SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0001 $01.25/0

THE COMPUTATIONAL COMPLEXITY OF CONTINUED FRACTIONS*

V. STRASSEN†

Abstract. The Knuth–Schönhage algorithm for expanding a quolynomial into a continued fraction is shown to be essentially optimal with respect to the number of multiplications/divisions used, uniformly in the inputs.

Key words. continued fraction, Euclidean representation, symbolic multiplication, computational complexity, computation tree, lower bound, degree method

1. Introduction. Let k be a field, $k[x]$ the polynomial ring over k in the indeterminate x. Let A_1, A_2 be polynomials, $A_2 \neq 0$. Applying the division algorithm successively (Euclid's algorithm) we get

$$\begin{aligned} A_1 &= Q_1 A_2 + A_3, \\ A_2 &= Q_2 A_3 + A_4, \\ &\cdots\cdots \\ A_{t-1} &= Q_{t-1} A_t, \end{aligned} \tag{1.1}$$

where $A_i \neq 0$, $\deg A_{i+1} < \deg A_i$ for $i > 1$. The sequence $(Q_1, \cdots, Q_{t-1})$ depends only on the quolynomial A_1/A_2 and is called the continued fraction of A_1/A_2. (For its significance in several branches of mathematics see [13], [26].) The name comes from the identity

$$A_1/A_2 = Q_1 + 1/(Q_2 + 1/(\cdots + 1/(Q_{t-2} + 1/Q_{t-1})\cdots)),$$

valid in $k(x)$, which follows from (1.1) by dividing the ith equation by A_{i+1} and eliminating all A_i/A_{i+1} with $i > 1$. $(Q_1, \cdots, Q_{t-1})$ determines A_1/A_2 uniquely.

Knuth [8] associates with (A_1, A_2) the extended sequence $(Q_1, \cdots, Q_{t-1}, A_t)$, which he calls the Euclidean representation of (A_1, A_2). It represents the pair (A_1, A_2) uniquely. In fact, one has a bijection between pairs of polynomials (A_1, A_2) such that $\deg A_1 \geqq \deg A_2 \geqq 0$ and finite sequences of polynomials $(Q_1, \cdots, Q_{t-1}, A_t)$ such that $t \geqq 2$, $\deg Q_1 \geqq 0$, $\deg Q_i > 0$ for $1 < i < t$ and $\deg A_t \geqq 0$. If we put $n = \deg A_1$, $m = \deg A_2$, then obviously $\sum_1^{t-1} \deg Q_i + \deg A_t = n$, $\sum_2^{t-1} \deg Q_i + \deg A_t = m$.

The Euclidean representation is rather informative. It contains the continued fraction of A_1/A_2 and the gcd A_t of A_1, A_2. Brown [3] and Collins [4] have exhibited the resultant of A_1, A_2 essentially as a power product of the leading coefficients of the Q_i and A_t. In particular, if $A_2 = d/dx\, A_1$, one gets the discriminant of A_1. If in addition $k = R$, one can read off from the Euclidean representation the number of zeros of A_1 in any real interval in linear time, since Sturm's algorithm may be carried out using the values of the Q_i and A_t at the endpoints of the interval.

In the sequel we will exclusively work with the Euclidean representation; however, our results will apply mutatis mutandis also to continued fractions.

* Received by the editors May 26, 1981, and in revised form December 21, 1981. The results of this paper were announced in Proc. Internat. Congress Math., Vancouver, 1974. Significant portions of this paper are reprinted with permission from "The Computational Complexity of Continued Fractions", by Volker Strassen, which was published in the Proceedings of the 1981 Symposium on Symbolic and Algebraic Computation, Copyright 1981, Association for Computing Machinery, Inc.

† Seminar für Angewandte Mathematik, Universität Zürich, Freiestrasse 36, CH-8032 Zürich, Switzerland.

How fast can we compute the sequence $(Q_1, \cdots, Q_{t-1}, A_t)$ from (A_1, A_2)? For simplicity and elegance we will allow in this paper additions, subtractions and multiplications by fixed scalars (which are thought to be stored in the program) for free and will thus only count "nonscalar" multiplications and divisions (Ostrowski's measure). For $n = m$, Euclid's algorithm requires in the worst and "normal" case, when $\deg Q_i = 1$ for $1 < i < t$ and $\deg A_t = 0$, about n^2 mult/div. The algorithm cannot be essentially improved if one insists on computing the A_i in addition to the Q_i (use a linear independence argument).

Lehmer [10] suggested employing the fact that for small $\deg Q_i$ only a small initial segment of A_i, A_{i+1} is needed to compute Q_i. Taking up this idea, Knuth [8] and Schönhage [16] constructed an ingenious $O(n \log n)$ algorithm for computing the Euclidean representation. Actually, all three authors were concerned with the number theoretic analogue of our situation ($\mathbb{Z}$ instead of $k[x]$). The translation to the somewhat simpler polynomial setting is due to Moenck [11].

In the present paper we will show that the Knuth–Schönhage algorithm is optimal up to a multiplicative constant. In fact we will prove this not only for the worst case, but in a strong sense uniformly over the set of input polynomials A_1, A_2, at least when k is algebraically closed.

We use the model of a computation tree, allowing tests of the form

$$\text{"if } a = 0 \text{ goto } i \text{ else goto } j\text{"}$$

free of charge. A computation tree computes a "collection" (φ, π), where φ is a function on the set J of inputs and π a finite partition of J (see § 5). Fixing n and m, we have in the case of the Euclidean representation

$$J = \{(A_1, A_2) \colon \deg A_1 = n, \deg A_2 = m\}$$
$$= \{(a_0, \cdots, a_n, b_0, \cdots, b_m) \in k^{n+m+2} \colon a_0 b_0 \neq 0\}$$

(with the identification $A_1 = \sum_0^n a_i t^{n-i}$, $A_2 = \sum_0^m b_j t^{m-j}$),

$$\varphi(A_1, A_2) = (Q_1, \cdots, Q_{t-1}, A_t)$$

(also represented by its sequence of coefficients of total length $n + t$),

$$\pi = \left\{ D(n_1, \cdots, n_t) \colon t \geqq 2, n_1, n_t \geqq 0, n_i > 0 \text{ for } 1 < i < t, \sum_1^t n_i = n, \sum_2^t n_i = m \right\},$$

where

$$D(n_1, \cdots, n_t) = \{(A_1, A_2) \in J \colon (\deg Q_1, \cdots, \deg Q_{t-1}, \deg A_t) = (n_1, \cdots, n_t)\}$$

is the set of inputs, whose Euclidean representation has the "format" $(n_1, \cdots, n_t)$. Our main result is the following (see §§ 4 and 6):

1. The Knuth–Schönhage algorithm computes the Euclidean representation with cost

$$\leqq 30n(H(n_1, \cdots, n_t) + 6.5)$$

on $D(n_1, \cdots, n_t)$ (H is the entropy function, see (2.1)).

2. Let k be algebraically closed. Any algorithm that computes the Euclidean representation has cost

$$\geqq n(H(n_1, \cdots, n_t) - 2)$$

on a Zariski dense open subset of $D(n_1, \cdots, n_t)$. (Any algorithm may, of course, be speeded up on particular inputs by a table-look-up procedure.)

In particular, for $n=m$, the order $n \log n$ of the worst case of the Knuth–Schönhage algorithm cannot be improved. The above result 2 remains true also for nonclosed fields, if the algorithm is assumed to yield the Euclidean representation over the algebraic closure as well. If this condition is not satisfied, we still get order-sharp lower bounds on those $D(n_1, \cdots, n_t)$ with $t \geqq (\frac{1}{2}+\varepsilon)m$. ($\varepsilon > 0$; this of course covers the worst case.) Similar results hold in the important situation of polynomials over a field $\mathbb{Z}_p$, where p is not known in advance.

For proving lower bounds we employ the geometric degree method (Strassen [23], see also Borodin–Munro [2], Schönhage [17], Heintz [6′], Schnorr [15]). For this reason the paper assumes some knowledge of the language of classical algebraic geometry (see Mumford [12], Shafarevich [18], Samuel [14]). Let k be algebraically closed. The degree of a closed irreducible set $X \subset k^n$ is the typical number of points of intersection of X with an affine subspace of k^n of complementary dimension. (This coincides with the degree of the closure of X in $\mathbb{P}^n$.) The degree of a closed, but reducible subset of k^n is the sum of the degrees of its components. The degree of a locally closed set X is the degree of its closure $\bar{X}$ (thus also the sum of the degrees of its components). We have Bezout's inequality

$$\deg(X \cap Y) \leqq \deg X \deg Y \tag{1.2}$$

for closed X, $Y \subset k^n$. We will use this inequality mainly in the case, when Y is an affine subspace of k^n, where it becomes $\deg(X \cap Y) \leqq \deg X$. If $f_1, \cdots, f_r \in k(x_1, \cdots, x_n)$ are rational functions, we denote by $\deg(f_1, \cdots, f_r)$ the degree of the locally closed graph $W \subset k^{n+r}$ of the rational map $k^n \to k^r$ defined by $(f_1, \cdots, f_r)$. Then if $L(f_1, \cdots, f_r)$ is the complexity of $f_1, \cdots, f_r$ with respect to the cost measure introduced above (see [23], [2]), we have the degree bound

$$L(f_1, \cdots, f_r) \geqq \log \deg(f_1, \cdots, f_r). \tag{1.3}$$

(In this paper log always means $\log_2$.)

2. Symbolic multiplication of several polynomials. The results of this section will be used later, but they are also of independent interest. Let $n_1, \cdots, n_t$ be nonnegative integers, $n = \sum_i n_i$. We denote the entropy of the probability vector $(n_1/n, \cdots, n_t/n)$ by $H(n_1, \cdots, n_t)$, i.e.,

$$H(n_1, \cdots, n_t) = -\sum_{n_i > 0} \frac{n_i}{n} \log \frac{n_i}{n}. \tag{2.1}$$

(In case $n=0$ we set $H(n_1, \cdots, n_t) = 0$.) Obviously the entropy does not change if we remove from $(n_1, \cdots, n_t)$ all n_i which are $=0$. We list a few properties of the entropy, some of which will be used in later sections (for detailed proofs see Fano [5]). We have $0 \leqq H(n_1, \cdots, n_t) \leqq \log n$ for $n > 0$, with both bounds attained. $nH(n_1, \cdots, n_t)$ is monotonic in each argument n_i (as one sees by differentiating): If $n_i \leqq n_i'$ for all i and if $n' = \sum_1' n_i'$, then

$$nH(n_1, \cdots, n_t) \leqq n'H(n_1', \cdots, n_t'). \tag{2.2}$$

Since inserting zeros into the sequence $(n_1, \cdots, n_t)$ does not change the entropy, this implies

$$\left(\sum_1^{t-1} n_i\right) H(n_1, \cdots, n_{t-1}) \leqq nH(n_1, \cdots, n_t). \tag{2.3}$$

The following crucial property is easily checked:

$$\left(\sum_1^s n_i\right) H(n_1, \cdots, n_s) + \left(\sum_{s+1}^t n_i\right) H(n_{s+1}, \cdots, n_t)$$
$$= n\left(H(n_1, \cdots, n_t) - H\left(\sum_1^s n_i, \sum_{s+1}^t n_i\right)\right). \tag{2.4}$$

It is convenient to extend the definition (2.1) by allowing nonnegative real numbers p_i in place of n_i. Since the entropy is invariant under scaling, we can reformulate (2.4) as follows: If $\sum_1^s n_i = pn$, then

$$pH(n_1, \cdots, n_s) + (1-p)H(n_{s+1}, \cdots, n_t) = H(n_1, \cdots, n_t) - H(p, 1-p). \tag{2.5}$$

In the sequel we will often write H for $H(n_1, \cdots, n_t)$.

LEMMA 2.1.

$$\log\left(\frac{n!}{n_1! \cdots n_t!}\right) \geqq n(H(n_1, \cdots, n_t) - 2).$$

Proof. We may assume that all n_i are positive. By Stirling's formula with error estimate, we have

$$\frac{n!}{n_1! \cdots n_t!} \geqq 2^{nH}(2\pi n)^{1/2} \prod_{j=1}^t (2\pi n_j)^{-1/2} e^{-1/(12n_j)}.$$

(See, e.g., Fano [5, (8.87)]; notice that Fano works with natural logarithms.) Because of $n_1 \cdots n_t \leqq (n/t)^t$, this implies

$$\log\left(\frac{n!}{n_1! \cdots n_t!}\right) \geqq nH + \frac{1}{2}\log(2\pi n) - \frac{t}{2}\log\frac{2\pi n}{t} - \frac{t}{12}\log e$$
$$\geqq nH - \frac{t}{2}\log\frac{2\pi n}{t} - \frac{t}{12}\log e.$$

Given n, the sum of the absolute values of the last two terms is maximal for

$$t = 2\pi n e^{-5/6},$$

and for this value of t, we have

$$\frac{t}{2}\log\left(\frac{2\pi n}{t}\right) + \frac{t}{12}\log e \leqq 2n.$$

Let k be an infinite field and let $x, p_{11}, \cdots, p_{1n_1}, \cdots, p_{t1}, \cdots, p_{tn_t}$ be indeterminates over k. We put

$$P_i = x^{n_i} + p_{i1}x^{n_i-1} + \cdots + p_{in_i} \tag{2.6}$$

and

$$A = P_1 \cdots P_t. \tag{2.7}$$

The polynomial A has the form

$$A = x^n + a_1 x^{n-1} + \cdots + a_n,$$

where the a_i are polynomials in the p_{js}, i.e., $a_i \in k[p_{11}, \cdots, p_{tn_t}] = k[\mathbf{p}]$. The following theorem holds true irrespective of whether we interpret the complexity $L(a_1, \cdots, a_n)$

in $k[\mathbf{p}]$ (not allowing division) or in the field of rational functions $k(\mathbf{p})$. In either case linear operations are not to be counted.

THEOREM 2.2.

$$n(H(n_1, \cdots, n_t)-2) \leqq L(a_1, \cdots, a_n) \leqq n(H(n_1, \cdots, n_t)+1).$$

In particular, as $H \to \infty$,

$$L(a_1, \cdots, a_n) \sim nH(n_1, \cdots, n_t).$$

Proof. We may assume that all n_i are positive.

Left inequality. Without loss of generality let k be algebraically closed. Choose $\alpha_1, \cdots, \alpha_n \in k$ such that the polynomial $x^n+\alpha_1 x^{n-1}+\cdots+\alpha_n$ has n simple roots in k, say $\theta_1, \cdots, \theta_n$. We will determine the number of solutions of the system of equations

$$a_1 = \alpha_1, \cdots, a_n = \alpha_n. \tag{2.8}$$

A point $(\kappa_{11}, \cdots, \kappa_{1n_1}, \cdots, \kappa_{t1}, \cdots, \kappa_{tn_t}) \in k^n$ is a solution of (2.8) if and only if

$$(x-\theta_1)\cdots(x-\theta_n) = (x^{n_1}+\kappa_{11}x^{n_1-1}+\cdots+\kappa_{1n_1})\cdots(x^{n_t}+\kappa_{t1}x^{n_t-1}+\cdots+\kappa_{tn_t}).$$

Therefore we have a bijection of the set of solutions of (2.8) and the set of partitions of $\{\theta_1, \cdots \theta_n\}$ into t classes with $n_1, \cdots, n_t$ elements, respectively. Thus they are exactly $n!/(n_1! \cdots n_t!)$ solutions. By (1.2) and Lemma 2.1 this implies

$$\log \deg (a_1, \cdots, a_n) \geqq \log\left(\frac{n!}{n_1! \cdots n_t!}\right) \geqq n(H-2), \tag{2.9}$$

and therefore by (1.3)

$$L(a_1, \cdots, a_n) \geqq n(H-2).$$

Right inequality. A word is a finite (possibly empty) sequence from the set $\{0, 1\}$. An s-code is a sequence $w_1, \cdots, w_s$ of s words such that for any $i \neq j$ the word w_i is not an initial segment of the word w_j. We will first show by induction on t (t being the number of polynomials to be multiplied symbolically, see (2.7)) that for any t-code $w_1, \cdots, w_t$

$$L(a_1, \cdots, a_n) \leqq \sum_1^t n_i \text{ length } (w_i). \tag{2.10}$$

This is clear for $t = 1$. Now let $t > 1$. It suffices to show (2.10) for a t-code $w_1, \cdots, w_t$ for which $\sum n_i$ length (w_i) is as small as possible. Because of $t > 1$ the code does not contain the empty word. We partition $\{1, \cdots, t\}$ into the set E of those i for which w_i begins with 0 and its complement F. E is nonempty. Otherwise all w_i would begin with 1. Deleting the initial 1 in each w_i would still leave us with a t-code, in contradiction to the assumption of minimality above. Similarly F is nonempty. We assume without loss of generality that $E = \{1, \cdots, s\}$. Deleting the initial zeros in $w_1, \cdots, w_s$ and the initial ones in $w_{s+1}, \cdots, w_t$ we obtain an s-code $\tilde{w}_1, \cdots, \tilde{w}_s$ and a $(t-s)$-code $\tilde{w}_{s+1}, \cdots, \tilde{w}_t$. We put $m = n_1+\cdots+n_s$ and define $b_1, \cdots, b_m, c_1, \cdots, c_{n-m} \in k[\mathbf{p}]$ by

$$x^m+b_1x^{m-1}+\cdots+b_m = P_1 \cdots P_s,$$

$$x^{n-m}+c_1x^{n-m-1}+\cdots+c_{n-m} = P_{s+1} \cdots P_t.$$

Obviously

$$x^n+a_1x^{n-1}+\cdots+a_n = (x^m+b_1x^{m-1}+\cdots+b_m)(x^{n-m}+c_1x^{n-m-1}+\cdots+c_{n-m}).$$

Our induction hypothesis implies

$$L(b_1, \cdots, b_m) \leqq \sum_1^s n_i \text{ length } (\tilde{w}_i),$$

$$L(c_1, \cdots, c_{n-m}) \leqq \sum_{s+1}^t n_i \text{ length } (\tilde{w}_i).$$

Since the symbolic multiplication of two monic polynomials of degrees m and $n-m$ can be achieved with n nonlinear operations (cf. [23, p. 244]) we conclude

$$\begin{aligned} L(a_1, \cdots, a_n) &\leqq L(b_1, \cdots, b_m) + L(c_1, \cdots, c_{n-m}) + n \\ &\leqq \sum_1^s n_i \text{ length } (\tilde{w}_i) + \sum_{s+1}^t n_i \text{ length } (\tilde{w}_i) + n \\ &= \sum_1^s n_i (\text{length } (\tilde{w}_i) + 1) + \sum_{s+1}^t n_i (\text{length } (\tilde{w}_i) + 1) \\ &= \sum_i^s n_i \text{ length } (w_i) + \sum_{s+1}^t n_i \text{ length } (w_i) \\ &= \sum_1^t n_i \text{ length } (w_i). \end{aligned}$$

Thus, we have proved (2.10) for an arbitrary t-code $w_1, \cdots, w_t$. Now we can always choose a t-code $w_1, \cdots, w_t$ such that

$$\sum_1^t n_i \text{ length } (w_i) \leqq n(H+1)$$

(see Fano [5, § 3.5]). Therefore,

$$L(a_1, \cdots, a_n) \leqq n(H+1).$$

This completes the proof of the theorem.

3. Conversion of a continued fraction into a rational fraction. We need the following:

LEMMA 3.1. *Let k be algebraically closed and let*

$$f_1(y, x_1, \cdots, x_n),$$

$$\cdots\cdots$$

$$f_r(y, x_1, \cdots, x_n)$$

be polynomials. For $\mu \in k$ let $W_\mu \subset k^{n+r}$ be the graph of the map

$$(\alpha_1, \cdots, \alpha_n) \mapsto (f_1(\mu, \boldsymbol{\alpha}), \cdots, f_r(\mu, \boldsymbol{\alpha})).$$

Then the function

$$\mu \mapsto \deg (W_\mu)$$

is Zariski lower semicontinuous (i.e., it equals its maximum value except on finitely many points).

Proof. Let W be the graph of the map

$$(\mu, \alpha_1, \cdots, \alpha_n) \mapsto (f_1(\mu, \boldsymbol{\alpha}), \cdots, f_r(\mu, \boldsymbol{\alpha})).$$

W is an $(n+1)$-dimensional closed subvariety of $k \times k^{n+r}$. For any $\mu \in k$ the graph W_μ is an n-dimensional closed subvariety of k^{n+r}, and we have

$$\{\mu\} \times W_\mu = W \cap (\{\mu\} \times k^{n+r}) = W \cdot (\{\mu\} \times k^{n+r}), \tag{3.1}$$

since W and $\{\mu\} \times k^{n+r}$ intersect transversally (see van der Waerden [27], Samuel [14], Hartshorne [6], Mumford [12]). Let $\bar{W}$ be the closure of W in $k \times \mathbb{P}^{n+r}$, $\bar{W}_\mu$ the closure of W_μ in $\mathbb{P}^{n+r}$. Since

$$\begin{aligned} \dim (\bar{W} \cap (\{\mu\} \times \mathbb{P}^{n+r})) &\leqq \dim ((\{\mu\} \times W_\mu) \cup (\bar{W} - W)) \\ &= \max \{\dim W_\mu, \dim (\bar{W} - W)\} = n, \end{aligned} \tag{3.2}$$

$\bar{W}$ and $\{\mu\} \times \mathbb{P}^{n+r}$ intersect properly. By conservation of number (see van der Waerden [27], Samuel [14]), the quantity

$$\deg (\bar{W} \cdot (\{\mu\} \times \mathbb{P}^{n+r}))$$

is independent of μ, say $= c$.

On the other hand we have

$$\{\mu\} \times \bar{W}_\mu = \bar{W} \cdot (\{\mu\} \times \mathbb{P}^{n+r}) \tag{3.3}$$

for all but finitely many points $\mu_1, \cdots, \mu_v$. For if we restrict both sides of (3.3) to affine space $k \times k^{n+r}$, we get equality by (3.1). So (3.3) can be invalidated only by the appearance of components of $\bar{W} \cap (\{\mu\} \times \mathbb{P}^{n+r})$ disjoint from $k \times k^{n+r}$. Now any such component lies in $\bar{W} - W$ and has dimension n by (3.2), so it is a component of $\bar{W} - W$. But $\{\mu\} \times \mathbb{P}^{n+r}$ can contain a component of $\bar{W} - W$ for only finitely many μ. Equation (3.3) implies

$$\deg (W_\mu) = \deg (\bar{W}_\mu) = \deg (\bar{W} \cdot (\{\mu\} \times \mathbb{P}^{n+r})) = c$$

for $\mu \notin \{\mu_1, \cdots, \mu_v\}$. Also we have

$$\deg (W_{\mu_i}) = \deg (\bar{W}_{\mu_i}) \leqq \deg (\bar{W} \cdot (\{\mu_i\} \times \mathbb{P}^{n+r})) = c$$

since $\bar{W}_{\mu_i}$ is always a component of $\bar{W} \cdot (\{\mu_i\} \times \mathbb{P}^{n+r})$. These two statements prove the lemma. (As has been observed by J. Heintz, the lemma can also be proved without using the principle of conservation of number.)

Now let k be an arbitrary infinite field, $t \geqq 2$ and $n_1, \cdots, n_t$ be nonnegative integers such that $n_i > 0$ for $1 < i < t$. Let x and

$$\begin{aligned} &q_{10}, \cdots, q_{1n_1}, \\ &\cdots\cdots\cdots \\ &q_{t0}, \cdots, q_{tn_t} \end{aligned}$$

be indeterminates over k. Put

$$Q_i = q_{i0} x^{n_i} + \cdots + q_{in_i}$$

for $1 \leqq i < t$ and

$$A_t = q_{t0} x^{n_t} + \cdots + q_{tn_t}.$$

Then the system of polynomial equations

$$A_1 = Q_1 A_2 + A_3,$$
$$A_2 = Q_2 A_3 + A_4,$$
$$\cdots\cdots$$
$$A_{t-1} = Q_{t-1} A_t$$

uniquely determines polynomials $A_1, \cdots, A_{t-1}$. We have

$$A_1 = a_0 x^n + \cdots + a_n, \qquad A_2 = b_0 x^m + \cdots + b_m,$$

where $n = \sum_1^t n_i$, $m = \sum_2^t n_i$ and $a_0, \cdots, a_n, b_0, \cdots, b_m \in k[\mathbf{q}]$.

In the following theorem we can interpret $L(a_0, \cdots, a_n, b_0, \cdots, b_m)$ either in $k[\mathbf{q}]$ (not allowing division) or in $k(\mathbf{q})$. As usual, linear operations are free.

THEOREM 3.2. *Let* $n > 0$. *Then*

$$n(H(n_1, \cdots, n_t) - 2) \leqq L(a_0, \cdots, a_n, b_0, \cdots, b_m) \leqq 8n(H(n_1, \cdots, n_t) + 7).$$

Proof. Left-hand inequality. Without loss of generality let k be algebraically closed. By induction one easily sees

$$\begin{pmatrix} A_i \\ A_{i+1} \end{pmatrix} = \begin{pmatrix} Q_i & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} Q_{t-1} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} A_t \\ 0 \end{pmatrix},$$

in particular,

$$\begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = \begin{pmatrix} Q_1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} Q_{t-1} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} A_t \\ 0 \end{pmatrix}. \tag{3.4}$$

Let $\mu \in k$ be different from 0. In (3.4) we make the substitution

$$q_{i0} \mapsto \frac{1}{\mu}, \qquad q_{ij} \mapsto \frac{1}{\mu} p_{ij}$$

(where $1 \leqq i \leqq t$, $1 \leqq j \leqq n_i$ and the p_{ij} are new indeterminates) and multiply both sides of (3.4) by μ^t. We get

$$\begin{pmatrix} a_0(\mu)x^n + a_1(\mu)x^{n-1} + \cdots + a_n(\mu) \\ b_0(\mu)x^m + b_1(\mu)x^{m-1} + \cdots + b_m(\mu) \end{pmatrix} = \begin{pmatrix} P_1 & \mu \\ \mu & 0 \end{pmatrix} \cdots \begin{pmatrix} P_{t-1} & \mu \\ \mu & 0 \end{pmatrix} \begin{pmatrix} P_t \\ 0 \end{pmatrix}, \tag{3.5}$$

where

$$P_i = x^{n_i} + p_{i1}x^{n_i - 1} + \cdots + p_{in_i}$$

and where $a_j(\mu), b_j(\mu)$ are obtained from a_j, b_j by the above substitution and subsequent multiplication by μ^t. Since the graph of the polynomial map defined by $a_i(\mu), \cdots, a_n(\mu)$ is essentially an affine linear section of the graph of $a_1, \cdots, a_n$, the degree of the former is less or equal than the degree of the latter. Thus, for any $\mu \neq 0$,

$$\begin{aligned} \log \deg (a_0, \cdots, a_n, b_0, \cdots, b_m) &\geqq \log \deg (a_1, \cdots, a_n) \\ &\geqq \log \deg (a_1(\mu), \cdots, a_n(\mu)). \end{aligned} \tag{3.6}$$

(3.5) shows that the $a_j(\mu)$ are polynomials in $\mathbf{p}$ which depend polynomially on the parameter μ. In particular, $a_j(\mu)$ make sense for $\mu = 0$ and (3.5) remains correct in this case, i.e.,

$$x^n + a_1(0)x^{n-1} + \cdots + a_n(0) = P_1 \cdots P_t. \tag{3.7}$$

By Lemma 3.1 we have

$$\log \deg (a_1(\mu), \cdots, a_n(\mu)) \geqq \log \deg (a_1(0), \cdots, a_n(0)) \tag{3.8}$$

for all but finitely many μ. Equation (3.7) together with (2.9) imply

$$\log \deg (a_1(0), \cdots, a_n(0)) \geqq n(H-2). \tag{3.9}$$

Equations (3.6), (3.8) and (3.9) yield

$$\log \deg (a_0, \cdots, a_n, b_0, \cdots, b_m) \geqq n(H-2). \tag{3.10}$$

Now (1.3) gives the left-hand inequality of the theorem.

Right-hand inequality. For any 2×2 matrix

$$G = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}$$

whose coefficients g_{ij} are polynomials in x with coefficients in $k(\mathbf{q})$, we put

$$\max \deg G = \text{maximum of the degrees with respect to } x \text{ of the } g_{ij},$$

$$L(G) = L \text{ (the set of the coefficients with respect to } x \text{ of the } g_{ij}).$$

Let

$$G_i = \begin{pmatrix} Q_i & 1 \\ 1 & 0 \end{pmatrix}$$

for $i \leqq t-1$ and

$$G_t = \begin{pmatrix} A_t & 0 \\ 0 & 0 \end{pmatrix}.$$

Then $\max \deg G_i = n_i$, and by (3.4), it suffices to show

$$L(G_1 \cdots G_t) \leqq 8n(H(n_1, \cdots, n_t)+7). \tag{3.11}$$

The problem of computing the matrix product $G_1 \cdots G_t$ is similar to the problem of computing the product of polynomials $P_1 \cdots P_t$ as in Theorem 2.2 (of course, in both cases we are dealing with symbolic computations, i.e., computations on coefficients), the main difference being that matrices do not commute. We replace (3.1)) by

$$L(G_1 \cdots G_t) \leqq cn(H(n_1, \cdots, n_t)+d)+7(t-1), \tag{3.12}$$

where we will choose c, $d \geqq 1$ at the end of the proof, which is by induction on t. (To carry out the induction, we will use the special form of the G_i only in as far as $\max \deg G_i \leqq n_i$ and the coefficients of the polynomials in x appearing in any G_i are either indeterminates or constants.)

The start ($t=2$) being clear, let $t>2$ and, therefore $n>0$. There is a unique s ($1 \leqq s \leqq t$) such that

$$\sum_1^{s-1} n_i \leqq \frac{n}{2}, \qquad \sum_1^{s} n_i > \frac{n}{2}.$$

Define p, p' by

$$pn = \sum_1^{s-1} n_i, \qquad p'n = \sum_1^{s} n_i.$$

Then we have $p \leqq \frac{1}{2} < p'$. Choose $0 < \varepsilon < \frac{1}{2}$.

Case $p' \leqq 1-\varepsilon$. We first compute $G_1 \cdots G_s$ and $G_{s+1} \cdots G_t$, and then, by one matrix multiplication, $G_1 \cdots G_t$. Using the matrix multiplication algorithm of [20] together with the fact that

$$\max \deg (G_1 \cdots G_t) \leqq n$$

and then the induction hypothesis and property (2.5) of the entropy function, we get

$$\begin{aligned} L(G_1 \cdots G_t) &\leqq L(G_1 \cdots G_s) + L(G_{s+1} \cdots G_t) + 7(n+1) \\ &\leqq cp'n(H(n_1, \cdots, n_s) + d) + 7(s-1) + c(1-p')n(H(n_{s+1}, \cdots, n_t) + d) \\ &\quad + 7(t-s-1) + 7(n+1) \\ &\leqq cn(H(n_1, \cdots, n_t) + d) + 7(t-1) + 7n - cnH(p', 1-p'). \end{aligned}$$

Now $H(p', 1-p') \geqq H(1-\varepsilon, \varepsilon)$. Thus, if the condition

$$7 \leqq cH(\varepsilon, 1-\varepsilon) \tag{3.13}$$

is satisfied, we have (3.12).

Case $p' > 1-\varepsilon$, $p \geqq \varepsilon$. We first compute $G_1 \cdots G_{s-1}$ and $G_s \cdots G_t$ and then $G_1 \cdots G_t$. Again (3.13) implies (3.12).

Case $p < \varepsilon$, $p' > 1-\varepsilon$. We first compute $G_1 \cdots G_{s-1}$ and $G_{s+1} \cdots G_t$ and then, by two matrix multiplications, $(G_1 \cdots G_{s-1})G_s(G_{s+1} \cdots G_t)$. Using induction hypothesis and properties (2.3) and (2.5) of the entropy function, we get

$$\begin{aligned} L(G_1 \cdots G_t) &\leqq L(G_1 \cdots G_{s-1}) + L(G_{s+1} \cdots G_t) + 14(n+1) \\ &\leqq cpn(H(n_1, \cdots, n_{s-1}) + d) + 7(s-2) + c(1-p')n(H(n_{s+1}, \cdots, n_t) + d) \\ &\quad + 7(t-s-1) + 14(n+1) \\ &\leqq cn(p'H(n_1, \cdots, n_s) + (1-p')H(n_{s+1}, \cdots, n_t) \\ &\qquad + (p+1-p')d) + 7(t-1) + 14n \\ &\leqq cn(H(n_1, \cdots, n_t) + 2\varepsilon d) + 7(t-1) + 14n. \end{aligned}$$

Thus in this case the condition

$$14 \leqq (1-2\varepsilon)cd \tag{3.14}$$

implies (3.12). Now we choose $\varepsilon = 0.325$, $c = 8$ and $d = 5$. Then (3.13) and (3.14) are satisfied and the theorem follows from (3.12), because $n_i > 0$ for $1 < i < t$ implies $t - 1 \leqq n + 1$.

4. Conversion of a rational fraction into a continued fraction: Analysis of the Knuth–Schönhage algorithm. Let k be an infinite field, $n \geqq m$ nonnegative integers. Given univariate polynomials A_1, A_2 over k with $\deg A_1 = n$, $\deg A_2 = m$, there are unique nonzero polynomials $Q_1, \cdots, Q_{t-1}, A_3, \cdots, A_t$ such that

$$\begin{aligned} A_1 &= Q_1 A_2 + A_3, \\ A_2 &= Q_2 A_3 + A_4, \\ &\cdots\cdots \\ A_{t-1} &= Q_{t-1} A_t \end{aligned} \tag{4.1}$$

and $\deg A_{i+1} < \deg A_i$ for $i \geqq 2$ (Euclid's algorithm). The sequence

$$(Q_1, \cdots, Q_{t-1}, A_t)$$

is called the Euclidean representation of (A_1, A_2) (Knuth [8]). We have $t \geqq 2$. If we put

$$n_i := \deg Q_i \quad (1 \leqq i \leqq t-1), \qquad n_t := \deg A_t, \tag{4.2}$$

then $n_i \geqq 0$ for all i and $n_i > 0$ for $1 < i < t$. Furthermore

$$n = \sum_1^t n_i, \qquad m = \sum_2^t n_i.$$

We define $A_{t+1} = 0$ and

$$M_i = \begin{pmatrix} 0 & 1 \\ 1 & -Q_i \end{pmatrix}. \tag{4.3}$$

Then

$$\begin{pmatrix} A_s \\ A_{s+1} \end{pmatrix} = M_{s-1} \cdots M_1 \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \tag{4.4}$$

for $s \leqq t$. (The reader will notice that (4.4) for $s = t$ is just the inverse relationship of (3.4) if there the indeterminates q_{ij} are replaced by elements of k.)

In this section we will show, using an algorithm that is essentially due to Knuth [8] and Schönhage [16] (see also Moenck [11]), how to compute the coefficients of A_1, A_2 in a rather efficient way. Since the length of the output (number of coefficients of the Q_i and of A_t) depends on the input (A_1, A_2), it is clear that our computational model has to be extended by allowing branching instructions, say of the form

"if $f = 0$, then goto i else goto j".

In this way we get the well-known model of a computation tree. For the purpose of proving lower bounds for the complexity of our problem, we will discuss this model in some detail in the next section. In the present section we will analyze the cost (number of nonlinear multiplications/divisions as a function of the input) of the Knuth-Schönhage algorithm and implicitly present the algorithm using an informal approach. The style is such, however, that it can easily be formalized with the help of Propositions 5.2 and 5.3. For completeness we will give an ALGOL-like formulation of the algorithm at the end of this section.

We need a preliminary result. Given a polynomial $A = a_0 x^q + \cdots + a_q \in k[x]$ of degree $q \geqq 0$ and an integer l, we set

$$A|l = \begin{cases} 0 & \text{if } l < 0, \\ a_0 x^l + \cdots + a_l & \text{if } 0 \leqq l \leqq q, \\ a_0 x^l + \cdots + a_q x^{l-q} & \text{if } l > q. \end{cases} \tag{4.5}$$

($A|l$ consists, so to speak, of the significant part of A of length $l+1$.) Obviously

$$(Ax^j)|l = A|l.$$

Given two pairs of polynomials (A, B) and (A', B') such that

$$\deg A \geqq \deg B \geqq 0, \qquad \deg A' \geqq \deg B' \geqq 0,$$

and an integer l, we say that (A, B) and (A', B') coincide up to l if and only if

$$A|l = A'|l,$$

$$B|(l-(\deg A-\deg B)) = B'|(l-(\deg A'-\deg B')).$$

Coincidence up to l is an equivalence relation. (A, B) and (Ax^j, Bx^j) coincide up to l for every $j \geqq 0$ (given that $\deg A \geqq \deg B \geqq 0$). If (A, B) and (A', B') coincide up to l and $l \geqq \deg A - \deg B$, then $\deg A - \deg B = \deg A' - \deg B'$. The qualitative idea of the following lemma is due to Lehmer [10].

LEMMA 4.1. *Besides* (4.1) *consider Euclid's algorithm for another pair* $A_1', A_2' \in k[x]$ *with* $\deg A_1' \geqq \deg A_2'$:

$$A_1' = Q_1'A_2' + A_3',$$

$$A_2' = Q_2'A_3' + A_4',$$

$$\cdots\cdots$$

$$A_{t'-1}' = Q_{t'-1}'A_{t'}'.$$

Let l *be a nonnegative integer and* $1 \leqq s \leqq t$ *be such that* $\sum_1^{s-1} n_i \leqq l$ *and either* $s = t$ *or* $\sum_1^s n_i > l$. *Define* s' *similarly (using* A_1', A_2' *instead of* A_1, A_2). *Then, if* (A_1, A_2) *and* (A_1', A_2') *coincide up to* $2l$, *we have* $s = s'$ *and* $Q_i = Q_i'$ *for* $1 \leqq i \leqq s-1$.

Proof. We show by induction on $1 \leqq j \leqq s$:

$$j \leqq s', \qquad Q_i = Q_i' \quad \text{for all } i < j,$$

and either $j = s$ or (A_j, A_{j+1}) and (A_j', A_{j+1}') coincide up to $2(l - \sum_1^{j-1} n_i)$. (This implies the lemma by symmetry.) The start of the induction is clear and the induction step is a consequence of the following statement:

Let (A, B) and (A', B') coincide up to $2l$, where $l \geqq \deg A - \deg B$, and let

$$\begin{aligned} A &= QB + C, \qquad \deg C < \deg B, \\ A' &= Q'B' + C', \qquad \deg C' < \deg B'. \end{aligned} \tag{4.6}$$

Then $Q = Q'$ and either $C = 0$ or $l - \deg Q < \deg B - \deg C$ or (B, C) and (B', C') coincide up to $2(l - \deg Q)$. To prove this statement, we may assume

$$\deg A = \deg A' > 2l$$

(by multiplying (A, B) and (A', B') with appropriate powers of x) and, therefore,

$$\deg (A - A') \leqq \deg A - 2l - 1,$$

$$\deg B \qquad = \deg B',$$

$$\deg (B - B') \leqq \deg A - 2l - 1.$$

Subtracting the equations (4.6), we get

$$A - A' = Q(B - B') + (Q - Q')B' + C - C'. \tag{4.7}$$

The polynomials $A - A'$, $Q(B - B')$ and $C - C'$ all have degrees $< \deg B$. Therefore,

$$\deg (Q - Q')B' < \deg B,$$

which implies $Q = Q'$. But then (4.7) gives

$$\deg (C - C') < \deg Q + \deg A - 2l. \tag{4.8}$$

Now assume $C \neq 0$, $l - \deg Q \geqq \deg B - \deg C$. Comparing this with (4.8) we get $\deg C = \deg C'$ (in particular $C' \neq 0$). But then (4.8) implies

$$C|2(l - \deg Q) - (\deg B - \deg C) = C'|2(l - \deg Q) - (\deg B' - \deg C').$$

This proves the statement and the lemma.

In the sequel when we speak of computing polynomials from polynomials, we always think of computing their coefficients from the coefficients of the given polynomials. At the end of such a computation, one will, of course, also know the degrees of the output polynomials. Similar remarks apply to matrices built up from polynomials.

LEMMA 4.2. *Let $n \geqq m \geqq 0$ and $l \geqq 0$. The function that assigns to any pair of polynomials A_1, A_2, with $\deg A_1 = n$, $\deg A_2 = m$, the sequence*

$$(Q_1, \cdots, Q_{s-1}, M_{s-1} \cdots M_1),$$

with $\sum_1^{s-1} n_i \leqq l$ and either $s = t$ or $\sum_1^s n_i > l$, is computable in time

$$cl\left(H\left(n_1, \cdots, n_{s-1}, l - \sum_1^{s-1} n_i\right) + d\right) + e(s-1) + 1,$$

where $c = 30$, $d = 5$ and $e = 16$.

Proof. Induction on l. The cases $l = 0$ or $l < n - m$ being clear, assume $1 \leqq l \leqq n$ without loss of generality and $l \geqq n - m$. By working with the initial segments $A_1|2l$ and $A_2|2l - (n-m)$, instead of A_1 and A_2 respectively, and using Lemma 4.1, we may assume without loss of generality that

$$\deg A_1 \leqq 2l, \qquad \deg A_2 \leqq 2l.$$

By the induction hypothesis (applied to $\lfloor l/2 \rfloor$ instead of l) and (2.2), we can compute

$$(A_1, A_2) \mapsto (Q_1, \cdots, Q_{r-1}, M_{r-1} \cdots M_1)$$

in time

$$c\frac{l}{2}\left(H\left(n_1, \cdots, n_{r-1}, \frac{l}{2} - \sum_1^{r-1} n_i\right) + d\right) + e(r-1) + 1,$$

where $\sum_1^{r-1} n_i \leqq l/2$ and either $r = t$ or $\sum_1^r n_i > l/2$. By (4.4) we can compute

$$(A_1, A_2, M_{r-1} \cdots M_1) \mapsto (A_r, A_{r+1})$$

in time

$$4\left(\left(\sum_1^{r-1} n_i\right) + 2l + 1\right) \leqq 10l + 4.$$

Therefore

$$(A_1, A_2) \mapsto (Q_1, \cdots, Q_{r-1}, M_{r-1} \cdots M_1, A_r, A_{r+1}) \tag{4.9}$$

is computable in time

$$c\frac{l}{2}\left(H\left(n_1, \cdots, n_{r-1}, \frac{l}{2} - \sum_1^{r-1} n_i\right) + d\right) + e(r-1) + 10l + 5.$$

Now, if $r = s$ (this can be tested at no cost since $r = s$ if and only if $A_{r+1} = 0$ or $\sum_1^{r-1} n_i + \deg A_r - \deg A_{r+1} > l$), no further computation is necessary. Otherwise, we can compute

$$(A_r, A_{r+1}) \mapsto (Q_r, A_{r+1}, A_{r+2})$$

in time

$$6n_r+1+2l+1\leqq 8l+2$$

(by a division with remainder, using Sieveking [19], Strassen [23], Kung [9]). If $A_{r+2}\neq 0$, we apply the induction hypothesis to (A_{r+1}, A_{r+2}) instead of (A_1, A_2) and $l-\sum_1^r n_i$ instead of l. Thus,

$$(A_{r+1}, A_{r+2})\mapsto(Q_{r+1}, \cdots, Q_{s-1}, M_{s-1}\cdots M_{r+1})$$

is computable in time

$$c\Big(l-\sum_1^r n_i\Big)\Big(H\Big(n_{r+1}, \cdots, n_{s-1}, \Big(l-\sum_1^r n_i\Big)-\sum_{r+1}^{s-1} n_i\Big)+d\Big)+e(s-r-1)+1.$$

If $A_{r+2}=0$ we have $r=s-1$. So in any case (if $r<s$) we can compute

$$\begin{aligned}&(Q_1, \cdots, Q_{r-1}, M_{r-1}\cdots M_1, A_r, A_{r+1})\\ &\quad\mapsto(Q_1, \cdots, Q_{s-1}, M_{s-1}\cdots M_{r+1}, M_{r-1}\cdots M_1)\end{aligned}\tag{4.10}$$

in time

$$c\Big(l-\sum_1^r n_i\Big)\Big(H\Big(n_{r+1}, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\Big)+d\Big)+e(s-r-1)+8l+3.$$

By two matrix multiplications, taking into account the special form (4.3) of M_r, we can compute

$$(M_{s-1}\cdots M_{r+1}, M_r, M_{r-1}\cdots M_1)\mapsto(M_{s-1}\cdots M_1)$$

in time

$$2(l+1)+7(l+1)=9(l+1).$$

Together with (4.10) we see that

$$(Q_1, \cdots, Q_{r-1}, M_{r-1}\cdots M_1, A_r, A_{r+1})\mapsto(Q_1, \cdots, Q_{s-1}, M_{s-1}\cdots M_1)$$

is computable in time

$$c\Big(l-\sum_1^r n_i\Big)\Big(H\Big(n_{r+1}, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\Big)+d\Big)+e(s-r-1)+17l+12,$$

when $r<s$. Finally, we combine this with (4.9). Thus, (in any case),

$$(A_1, A_2)\mapsto(Q_1, \cdots, Q_{s-1}, M_{s-1}\cdots M_1)$$

is computable in time

$$c\frac{l}{2}\Big(H\Big(n_1, \cdots, n_{s-1}, \frac{l}{2}-\sum_1^{s-1} n_i\Big)+d\Big)+e(s-1)+10l+5=:t_1$$

if $r=s$, or in time

$$\begin{aligned}&c\frac{l}{2}\Big(H\Big(n_1, \cdots, n_{r-1}\frac{l}{2}-\sum_1^{r-1} n_i\Big)+d\Big)\\ &\quad+c\Big(l-\sum_1^r n_i\Big)\Big(H\Big(n_{r+1}, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\Big)+d\Big)+e(s-1)+27l+1=:t_2\end{aligned}$$

if $r<s$. (We have used $e=16$.) To complete the induction, we have to show

$$t_i \leqq cl\left(H\left(n_1, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\right)+d\right)+e(s-1)+1. \tag{4.11}$$

Now by (2.2)

$$t_1 \leqq cl\left(H\left(n_1, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\right)+d\right)-c\frac{l}{2}d+e(s-1)+10l+5,$$

implying (4.11) easily for $c=30$, $d=5$. To estimate t_2, choose $0<\varepsilon<1/2$.

Case $\sum_1^r n_i<(1-\varepsilon)l$. By (2.2) and (2.4) we have

$$\begin{aligned} t_2 &\leqq c\sum_1^r n_i(H(n_1, \cdots, n_r)+d)+c\left(l-\sum_1^r n_i\right)\left(H\left(n_{r+1}, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\right)+d\right) \\ &\quad +e(s-1)+27l+1 \\ &\leqq cl\left(H\left(n_1, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\right)+d\right)+e(s-1)+1+l(27-cH(1-\varepsilon, \varepsilon)). \end{aligned}$$

Thus, in this case (4.11) is a consequence of the condition

$$27 \leqq cH(1-\varepsilon, \varepsilon). \tag{4.12}$$

Case $\sum_1^r n_i \geqq (1-\varepsilon)l$. Here we have

$$\begin{aligned} t_2 &\leqq c\sum_1^r n_i(H(n_1, \cdots, n_r)+d)+c\left(\varepsilon-\frac{1}{2}\right)ld \\ &\quad +c\left(l-\sum_1^r n_i\right)\left(H\left(n_{r+1}, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\right)+d\right)+e(s-1)+27l+1 \\ &\leqq cl\left(H\left(n_1, \cdots, n_{s-1}, l-\sum_1^{s-1} n_i\right)+d\right)+e(s-1)+1+27l+c\left(\varepsilon-\frac{1}{2}\right)ld. \end{aligned}$$

In this case (4.11) is implied by

$$27 \leqq \left(\frac{1}{2}-\varepsilon\right)cd. \tag{4.13}$$

Now (4.12) and (4.13) are satisfied for $\varepsilon=0.32$, $c=30$, $d=5$.

THEOREM 4.3. *Let* $n \geqq m \geqq 0$, $n>0$. *The function that assigns to any pair of polynomials* A_1, A_2 *with* $\deg A_1=n$, $\deg A_2=m$ *their Euclidean representation* $(Q_1, \cdots, Q_{t-1}, A_t)$ *is computable in time*

$$30n(H(n_1, \cdots, n_t)+6.5).$$

Proof. Lemma 4.2 with $l=n$ shows that

$$(A_1, A_2) \mapsto (Q_1, \cdots, Q_{t-1}, M_{t-1}\cdots M_1)$$

is computable in time

$$30n(H(n_1, \cdots, n_t)+5)+16(t-1)+1.$$

By (4.4),

$$(A_1, A_2, M_{t-1}\cdots M_1) \mapsto A_t$$

is computable in time $2(n+1)$. Now use $t-1 \leqq n+1$. For completeness we now give

an ALGOL-like procedure for the function that appears in Lemma 4.2.

procedure SCH $(A, B, u; z, Q, M)$:
  if $B = 0$ or $u < \deg A - \deg B$ **then**
    begin $z := 1$;
      $M := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$;
      $Q := \emptyset$;
    end
  else
    begin $F := A|2u$;
      $G := B|(2u - (\deg A - \deg B))$;
      $v := \lfloor u/2 \rfloor$;
      SCH $(F, G, v; z, Q, M)$;
      $\binom{F}{G} := M\binom{F}{G}$;
      if $\left(\sum_{i<z} \deg Q[i]\right) + \deg F - \deg G \leqq u$ **and** $G \neq 0$ **then**
        begin $Q[z] := \operatorname{div}(F, G)$;
          $H := \operatorname{rem}(F, G)$;
          $F := G$;
          $G := H$;
          $v := u - \sum_{i \leqq z} \deg Q[i]$;
          SCH $(F, G, v; z', Q', M')$;
          for $i := 1$ **to** $z' - 1$ **do** $Q[i + z] := Q'[i]$;
          $M := M'\begin{pmatrix} 0 & 1 \\ 1 & -Q[z] \end{pmatrix} M$;
          $z := z + z'$
        end
    end

Remarks. 1. u, v, z, z', i are variables for numbers, A, B, F, G, H are variables for polynomials, M, M' are variables for 2×2 matrices of polynomials, Q, Q' are variables of sequences of polynomials.

2. Given A, B such that $B \neq 0$, we have

$$A = \operatorname{div}(A, B) \cdot B + \operatorname{rem}(A, B)$$

such that $\deg \operatorname{rem}(A, B) < \deg B$.

3. Let the contents of A, B, u be A_1, A_2, l respectively (see Lemma 4.2). Then, after running the procedure SCH, the contents of A, B, u will be unchanged, the content of z will be s, the content of $Q[i]$ will be Q_i for $1 \leq i < s$ and the content of M will be $M_{s-1} \cdots M_1$ (hopefully).

5. The computational model. We will discuss here the notion of a computation tree in some generality. Let Ω, P be disjoint types (sets together with arity functions). The $\omega \in \Omega$ are called operational symbols, the $\rho \in P$ are called relational symbols. A structure of type (Ω, P) is a set A together with an interpretation for each $m \geqq 0$ of any m-ary $\omega \in \Omega$ as an m-ary partial operation in A and of any m-ary $\rho \in P$ as an m-ary relation in A. Notationally we will not distinguish between a symbol and its interpretation.

Example. $\Omega = \{0, 1, +, -, *, /\}$, $P = \{\leqq\}$, $A = \mathbb{R}$. (See also [21], [22].)

Let $s_1, s_2, \cdots$ be variables (symbols which denote storage locations). A *computation tree* of type (Ω, P) is a binary tree B (see [1]) together with

1. A function that assigns:
   - to any vertex with exactly one son (simple vertex) an operational instruction of the form
   $$s_i := \omega(s_{j_1}, \cdots, s_{j_m}),$$
   where $m \geqq 0$, $i, j_1, \cdots, j_m > 0$ and $\omega \in \Omega$ m-ary,
   - to any vertex with two sons (branching vertex) a test instruction of the form
   $$\rho(s_{j_1}, \cdots, s_{j_m}),$$
   where $m \geqq 0$, $j_1, \cdots, j_m > 0$ and $\rho \in P$ m-ary,
   - to any leaf an output instruction of the form
   $$(s_{j_1}, \cdots, s_{j_q}),$$
   where $q \geqq 0$, $j_1, \cdots, j_q > 0$;
2. A partition σ of the set of leaves such that the length q of the assigned output instructions is constant on σ-classes.

The purpose of the partition is to collect the relevant part of the information gathered by the various tests of the tree, as is most easily visualized in the case of a decision tree, i.e., a computation tree with $\Omega = \phi$. Computation trees will have inputs as well as outputs of the form

$$(A, a_1, \cdots, a_n),$$

where A is a structure of type (Ω, P) and $a_1, \cdots, a_n \in A$. n is called the length of the input (output). Let J be a set of inputs of length n (A may vary).

A *collection* for J is a pair (φ, π), where φ is a function that assigns to any input $\mathbf{a} = (A, a_1, \cdots, a_n) \in J$ an output $\varphi(\mathbf{a}) = (A, b_1, \cdots, b_q)$ (the structure of $\varphi(\mathbf{a})$ is the same as that of $\mathbf{a}$, q may vary as a function of $\mathbf{a}$) and where π is a finite partition of J such that the length of $\varphi(\mathbf{a})$ is constant on π-classes.

Examples. If it is clear from the context which structure is in front of an input (or output) we will often neglect to write it.

(5.1) *Matrix inversion.* Let k be a field, $J = k^{n \times n}$,

$$\varphi((a_{ij})) = \begin{cases} (a_{ij})^{-1} & \text{if } \det(a_{ij}) \neq 0, \\ \phi & \text{otherwise,} \end{cases}$$

$\pi = \{\{\text{regular matrices}\}, \{\text{singular matrices}\}\}$. Then (φ, π) is a collection for J. Other interesting input sets for the same problem are, e.g.,

$$J = \bigcup_{p \text{ prime}} \{\mathbb{Z}_p\} \times \mathbb{Z}_p^{n \times n},$$

$$J = \{(a_{ij}) \in k^{n \times n} : (a_{ij}) \text{ orthogonal}\}.$$

(5.2) *Knapsack.* Consider $\mathbb{R}$ as an ordered field and let $J = \mathbb{R}^n$,

$$\varphi(\mathbf{a}) = \phi \quad \text{for all } \mathbf{a} \in \mathbb{R}^n,$$

$$\pi = \{E, \mathbb{R}^n \backslash E\}, \quad \text{where}$$

$$E = \left\{ \mathbf{a} \in \mathbb{R}^n : \exists I \subset \{1, \cdots, n\} \sum_{j \in I} a_i = 1 \right\}.$$

Then (φ, π) is a collection for J.

(5.3) *Euclidean representation.* Let k be an infinite field, considered as a commutative k-division algebra with equality $(\Omega = \{0, 1, +, -, *, /\} \sqcup k$, where $\lambda \in k$ is interpreted as multiplication by λ, $P = \{=\})$. Let $n \geqq m \geqq 0$, $J = (k^\times \times k^n) \times (k^\times \times k^m) \subset k^{n+m+2}$ $(k^\times = k \setminus \{0\})$. Think of inputs $\in J$ as pairs of polynomials A_1, A_2 such that $\deg A_1 = n$, $\deg A_2 = m$. Put

$$\varphi(A_1, A_2) = (Q_1, \cdots, Q_{t-1}, A_t)$$

(see (4.1)) and let π be the partition of J into the fibres of the map

$$(A_1, A_2) \mapsto (\deg Q_1, \cdots, \deg Q_{t-1}, \deg A_t).$$

In other words, given nonnegative numbers $n_1, \cdots, n_t$ such that $n_i > 0$ for $1 < i < t$ and

$$\sum_1^t n_i = n, \qquad \sum_2^t n_i = m,$$

let $D(n_1, \cdots, n_t) \subset J$ be the set of those (A_1, A_2) for which $\deg Q_i = n_i$ $(i < t)$, $\deg A_t = n_t$. Then the $D(n_1, \cdots, n_t)$ are just the π-classes. (φ, π) is a collection for J, which we call the Euclidean representation.

We will skip a detailed semantics of computation trees and just state the following conclusion: An input $\mathbf{a} = (A, a_1, \cdots, a_n)$ fed into a computation tree B of the same type may or may not produce a leaf together with an output $\mathbf{b} = (A, b_1, \cdots, b_q)$. (At the root of B the variables are assigned the values $(a_1, a_2, \cdots, a_n, \infty, \infty, \cdots)$. Then a directed path starting from the root together with an assignment to the variables for any vertex of the path is constructed.) If it does, we say that B is defined on $\mathbf{a}$. If B is defined on a set J of inputs of the same length, we let $\varphi(\mathbf{a}) = \mathbf{b}$, where $\mathbf{b}$ is the output produced by B on $\mathbf{a}$, and we let π be such that $\mathbf{a}$ and $\mathbf{a}'$ are in the same π-class if and only if the leaves that B produces at $\mathbf{a}$ and $\mathbf{a}'$ are in the same σ-class. Then (φ, π) is a collection for J. We say that B *computes* (φ, π). Now let a cost function

$$z : \Omega \cup P \to \mathbb{R}^+$$

be given. By adding the costs of the various instructions encountered when going from the root of B to a leaf, we may define the cost of any leaf of B. If B is defined on J, this gives us a function

$$t : J \to \mathbb{R}^+,$$

the *cost of* B *on* J ($\mathrm{t}(\mathbf{a})$ is the cost of the leaf of B produced by the input $\mathbf{a}$). Finally, given J, a collection (φ, π) for J and a function

$$t : J \to \mathbb{R}^+,$$

we say that (φ, π) *is computable in time* t if there is a computation tree B which computes (φ, π) and has cost $\leqq t$ on J. We also say that (φ, π) is strictly computable in time t if in addition B is required to have $\sigma = \{\{v\}: v \text{ leaf of } B\}$ (i.e., B has to output all the information gained by performing tests). Obviously, (φ, π) is computable in time t if and only if there is a partition π' of J finer than π such that (φ, π') is a collection, strictly computable in time t.

In order to eliminate the clumsy notion of a computation tree, we will now axiomatically characterize the correct statements of the form

(5.4) "(φ, π) is computable in time t".

To this end fix (Ω, P), z and J.

AXIOM. (id, $\{J\}$) *is computable in time* 0.

Rules of inference. Let (φ, π) be a collection for J, computable in time t, $D \in \pi$ and

$$\varphi \restriction D = (\varphi_0, \cdots, \varphi_q).$$

(Thus for $\mathbf{a} \in D$ we have $\varphi_0(\mathbf{a}) = A$, $\varphi_i(\mathbf{a}) \in A$ for $i \geqq 1$.)

(I) If $t \leqq t'$, then (φ, π) is computable in time t'.

(II) If $\varphi' = \varphi$ on $J \backslash D$ and

$$\varphi' \restriction D = (\varphi_0, \varphi_{i_1}, \cdots, \varphi_{i_p}),$$

where $1 \leqq i_1, \cdots, i_p \leqq q$, then (φ', π) is computable in time t.

(III) If $\varphi' = \varphi$ on $J \backslash D$ and

$$\varphi' \restriction D = (\varphi_0, \varphi_1, \cdots, \varphi_q, \omega(\varphi_1, \cdots, \varphi_m)),$$

where $m \leqq q$, $\omega \in \Omega$ m-ary and $(\varphi_1(\mathbf{a}), \cdots, \varphi_m(a)) \in Def\omega$ for any $\mathbf{a} \in D$, then (φ', π) is computable in time $t + z(\omega)1_D$ (where 1_D = indicator of D).

(IV) If

$$\pi' = (\pi \backslash \{D\}) \cup \{\{\mathbf{a} \in D : (\varphi_1(\mathbf{a}), \cdots, \varphi_m(\mathbf{a})) \in \rho\}, \{\mathbf{a} \in D : (\varphi_1(\mathbf{a}), \cdots, \varphi_m(\mathbf{a})) \notin \rho\}\},$$

where $m \leqq q$, $\rho \in P$ m-ary, then (φ, π') is computable in time $t + z(\rho)1_D$.

(V) If

$$\pi' = (\pi \backslash \{D, E\}) \cup \{D \cup E\},$$

where $E \in \pi$, $E \neq D$ but length $(\varphi \restriction E) = q$, then (φ, π') is computable in time t.

THEOREM 5.1. *The correct statements of the form "(φ, π) is computable in time t" are exactly those which can be deduced from the above axiom with the Rules of Inference* (I)–(V). (*Similarly for "strictly computable" and Rules* (I)–(IV).)

Proof. "Deducible $\Rightarrow$ correct": Straightforward. "Correct $\Rightarrow$ deducible": Show that if (φ, π) is strictly computable in time t, then the corresponding statement may be deduced from the axiom by the Rules (I)–(IV), using induction on the size of a "strict" computation tree which computes (φ, π).

A convenient way to use Theorem 5.1 is by *axiomatic induction*: Given z and J, let $\mathfrak{A}$ be a statement about triples (φ, π, t) such that the above axiom and the Rules (I)–(V) hold when all statements of the form (5.4) have been replaced by the corresponding statements $\mathfrak{A}(\varphi, \pi, t)$. Then we can conclude

$$(\varphi, \pi) \text{ computable in time } t \Rightarrow \mathfrak{A}(\varphi, \pi, t).$$

As simple applications of axiomatic induction, the following two propositions can be proved.

PROPOSITION 5.2. *Let (φ, π) be a collection for J, computable in time t and let $D_1 \in \pi$. Moreover, let (φ_1, π_1) be a collection for J_1, computable in time t_1, and assume $\varphi(D_1) \subset J_1$. Put*

$$\tilde{\varphi} = \begin{cases} \varphi & \text{on } J \backslash D_1, \\ \varphi_1 \circ \varphi & \text{on } D_1, \end{cases}$$

$$\tilde{\pi} = (\pi \backslash \{D_1\}) \cup \varphi^{-1}\pi_1,$$

$$\tilde{t} = t + (t_1 \circ \varphi) \cdot 1_{D_1}.$$

Then $(\tilde{\varphi}, \tilde{\pi})$ is computable in time $\tilde{t}$.

PROPOSITION 5.3. *Let (φ_i, π_i) be collections for J, computable in time t_i $(i = 1, 2)$. Define φ by*

$$\varphi(\mathbf{a}) = (A, b_1, \cdots, b_{q_1}, c_1, \cdots, c_{q_2}),$$

where $\varphi_1(\mathbf{a}) = (A, b_1, \cdots, b_{q_1})$, $\varphi_2(\mathbf{a}) = (A, c_1, \cdots, c_{q_2})$, *and let*

$$\pi = \pi_1 \wedge \pi_2.$$

Then (φ, π) *is a collection for* J, *computable in time* $t_1 + t_2$.

Both propositions remain correct when everywhere "computable" is replaced by "strictly computable".

Application. Consider the collection (φ, π) of Example (5.3) (Euclidean representation). If we take the cost function $z = 1_{\{*,/\}}$, i.e., if we allow linear operations and tests for free and count the remaining multiplications/divisions, then Theorem 4.3 (or rather its proof) shows that

(5.5) $\qquad (\varphi, \pi)$ is strictly computable in time $30n(H + 6.5)$.

In fact, using Propositions 5.2 and 5.3, the proof of § 4 can easily be formalized for the present model.

6. Conversion of a rational fraction into a continued fraction: Lower bounds. Let k be an infinite field, considered as a k-field with equality ($\Omega = \{0, 1, +, -, *, /\} \sqcup k$ with constants 0, 1 and unary $\lambda \in k$, $P = \{=\}$). As before, let $z = 1_{\{*,/\}}$.

THEOREM 6.1. *Assume that* k *is algebraically closed. Let* $J \subset k^n$ *be Zariski open and* (φ, π) *be a collection for* J, *strictly computable in time* T. *Let* $D \in \pi$ *and* $\varphi \restriction D = (\varphi_1, \cdots, \varphi_q)$ (disregarding $\varphi_0 = k$). *Then*:

1. D *is a Zariski locally closed* (see [12]) *subset of* $k^n \cdot \varphi \restriction D$ *is the restriction to* D *of a rational map* $k^n \to k^q$, *whose domain of definition includes* D. *Graph* $(\varphi \restriction D)$ *is a locally closed subset of* k^{n+q}.

2. $T \restriction D \geqq \log \deg \text{graph } (\varphi \restriction D)$.

Proof. By axiomatic induction we first show: If (φ, π) is strictly computable in time T, then for any $\tilde{D} \in \pi$ with $\varphi \restriction \tilde{D} = (\tilde{\varphi}_1, \cdots, \tilde{\varphi}_u)$ (say) there are rational functions $F_1, \cdots, F_u, G_1, \cdots, G_v, H_1, \cdots, H_w$ on k^n such that

(6.1) $\qquad \tilde{D} = \{G_1 = \cdots = G_v = 0, H_1 \cdots H_w \neq 0\} \cap J,$

(6.2) $\qquad$ the F_i are defined on $\tilde{D}$ and $\tilde{\varphi}_i = F_i \restriction \tilde{D}$,

(6.3) $\qquad T \restriction \tilde{D} \geqq L(F_1, \cdots F_u, G_1, \cdots, G_v, H_1, \cdots, H_w).$

(The domain of definition of a rational function F is the set of points in k^n for which the reduced denominator does not vanish. The condition $G = 0$ is satisfied at a point $\boldsymbol{\alpha}$ if and only if $G(\boldsymbol{\alpha})$ is defined and equals 0. Similarly $H(\boldsymbol{\alpha}) \neq 0$ is meant to imply that $H(\boldsymbol{\alpha})$ is defined.) The inductive proof is quite straightforward, and we content ourselves with giving two instances of treating the rules of inference ((V) excluded). First, Rule (III) with $\omega = /$: The case $\tilde{D} \neq D$ being clear, let $\tilde{D} = D$. Then $u = q$ and the rational functions $F_1, \cdots, F_u, F_1/F_2, G_1, \cdots, G_v, H_1, \cdots, H_w$ will do. Second, Rule (IV): Take $F_1, \cdots, F_u, G_1, \cdots, G_v, F_1 - F_2, H_1, \cdots, H_w$ in case $\tilde{D} = D \cap \{\varphi_1 = \varphi_2\}$ and $F_1, \cdots, F_u, G_1, \cdots, G_v, H_1, \cdots, H_w, F_1 - F_2$ in case $\tilde{D} = D \cap \{\varphi_1 \neq \varphi_2\}$.

Equations (6.1) and (6.2) imply the first assertion of the theorem. By (6.3), (1.3) and (1.2) (intersection with a linear space), we have

$$\begin{aligned} T \restriction \tilde{D} &\geqq L(F_1, \cdots, F_u, G_1, \cdots, G_v, H_1, \cdots, H_w) \\ &\geqq \log \deg (F_1, \cdots, F_u, G_1, \cdots, G_v) \\ &\geqq \log \sum_{\substack{C \text{ component of} \\ \text{graph } (F_1, \cdots, F_u) \cap (\{G_1 = \cdots = G_v = 0\} \times k^u)}} \deg C. \end{aligned}$$

Now by (6.1) and (6.2)

$$\text{graph}\,(\varphi \restriction \tilde{D}) = \text{graph}\,(F_1, \cdots, F_u)$$
$$\cap\,(\{G_1 = \cdots = G_v = 0\} \times k^u) \cap (\{H_1 \cdots H_w \neq 0\} \cap J) \times k^u).$$

Thus the closure of any component of the graph of $\varphi \restriction \tilde{D}$ is also the closure of a component of graph $(F_1, \cdots, F_u) \cap (\{G_1 = \cdots = G_v = 0\} \times k^u)$. Therefore,

$$T \restriction \tilde{D} \geqq \log \sum_{\substack{C \text{ component} \\ \text{of graph } (\varphi \restriction \tilde{D})}} \deg C,$$

proving assertion 2 of the theorem.

We remark that there are finite partitions π of k^n into locally closed subsets, for which (id, π) is not strictly computable: Take $k = \mathbb{C}$, $n = 2$ and $\pi = \{D, E, F\}$, where $D = \{y = x^2 - 1, x \neq 1\}$, $E = \{-y = x^2 - 1, x \neq -1\}$, $F = \mathbb{C}^2 \backslash (D \cup E)$.

Now for any field k let

$$J_k(n, m) = \{k\} \times (k^\times \times k^n) \times (k^\times \times k^m).$$

This is an open subset of k^{n+m+2}. A typical input $(k, a_0, \cdots, a_n, b_0, \cdots, b_m) \in J_k(n, m)$ is interpreted as a pair of polynomials $A_1 = \sum_0^n a_i x^{n-i}$, $A_2 = \sum_0^m b_j x^{m-j} \in k[x]$ of degrees n and m, respectively. Let (φ_k, π_k) be the Euclidean representation for $J_k(n, m)$ (see (5.3)). Then π_k consists of the classes $D_k(n_1, \cdots n_t)$ $(t \geqq 2, n_1 \geqq 0, n_2, \cdots, n_{t-1} > 0, n_t \geqq 0, \sum_1^t n_i = n, \sum_2^t n_i = m)$, which by φ_k are mapped bijectively onto

$$J_k(n_1, \cdots, n_t) = \{k\} \times \prod_{i=1}^{t} (k^\times \times k^{n_i}).$$

$J_k(n_1, \cdots, n_t)$ is an open subset of k^{n+t}.

LEMMA 6.2. *Let k be algebraically closed.*

1. *$D_k(n_1, \cdots, n_t)$ is a locally closed irreducible subvariety of k^{n+m+2}.*
2. *$\varphi_k \restriction D_k(n_1, \cdots, n_t)$ is an isomorphism of varieties*

$$D_k(n_1, \cdots, n_t) \to J_k(n_1, \cdots, n_t).$$

Its inverse is given by polynomials of degree $\leqq t$.

3. *Let $W_k \subset k^{n+m+2} \times k^{n+t}$ be the graph of $\varphi_k \restriction D_k(n_1, \cdots, n_t)$. Then W_k is locally closed, irreducible and*

$$\log \deg W_k \geqq n(H(n_1, \cdots, n_t) - 2).$$

4. *Put $N = (n+m+2)+(n+t)$, $z = (n+m+2)$. There are polynomials $F_1, \cdots, F_z \in k[x_1, \cdots, x_N]$ of degree $\leqq t$ such that*

$$\bar{W}_k = \{F_1 = \cdots = F_z = 0\}.$$

Proof. By (5.2) the Euclidean representation is strictly computable. Thus, by Theorem 6.1 $D_k(n_1, \cdots, n_t)$ is locally closed and $\varphi_k \restriction D_k(n_1, \cdots, n_t)$ is a morphism into $J_k(n_1, \cdots, n_t)$. Equation (3.4) interpreted as a function ψ on $J_k(n_1, \cdots, n_t)$ gives its inverse. So $\varphi_k \restriction J_k(n_1, \cdots, n_t)$ is an isomorphism, $D_k(n_1, \cdots, n_t)$ is irreducible and ψ is defined by polynomials of degree $\leqq t$. Since W_k is up to a permutation of the coordinates the same as graph ψ, (3.10) gives

$$\log \deg W_k \geqq n(H - 2).$$

Finally, let $f_1, \cdots, f_z \in k[x_{z+1}, \cdots, x_N]$ be the polynomials defining ψ. Put $F_i = x_i - f_i$ for $1 \leqq i \leqq z$. Then by (3.4) $\deg F_i \leqq t$ and $\bar{W}_k = \{F_1 = \cdots = F_z = 0\}$.

THEOREM 6.3. *Let k be algebraically closed and let the Euclidean representation for $J_k(n, m)$ be computable in time T. Then any $D_k(n_1, \cdots, n_t)$ contains a dense open subset U such that*

$$T \upharpoonright U \geqq n(H(n_1, \cdots, n_t) - 2).$$

Proof. There is a refinement π' of π_k such that (φ_k, π') is strictly computable in time T. π' further subdivides $D_k(n_1, \cdots, n_t)$ into $D_1, \cdots, D_p$, say. By Theorem 6.1 the D_i are locally closed subsets of the irreducible variety $D_k(n_1, \cdots, n_t)$, so one of them (call it U) is a dense open subset of $D_k(n_1, \cdots, n_t)$. Then the graph of $\varphi_k \upharpoonright U$ and the graph of $\varphi_k \upharpoonright D_k(n_1, \cdots, n_t)$ have the same closure. Therefore, the graph of $\varphi_k \upharpoonright U$ is irreducible and, by Lemma 6.2. 3,

$$\log \deg (\text{graph } \varphi_k \upharpoonright U) \geqq n(H - 2).$$

Theorem 6.1. 2 now yields $T \upharpoonright U \geqq n(H-2)$.

Next we will discuss the Euclidean representation for nonclosed fields. Let k be an infinite field, K its algebraic closure. A point of K^n that belongs to k^n is called rational. k^n has a Zariski topology (generated by the sets $\{f \neq 0\}$, where $f \in k[x_1, \cdots, x_n]$). This is also the topology induced by the Zariski topology in K^n. (If $f \in K[\mathbf{x}]$, let $\{f_1, \cdots, f_z\}$ be its orbit under the action of Gal (K/k) on $K[\mathbf{x}]$. Put $g = \prod_1^z f_i$ if char $k = 0$ and $g = (\prod_1^z f_i)^{p^e}$ with e sufficiently large if char $k = p$. Then $g \in k[\mathbf{x}]$ and $(\xi \in k^n : f(\xi) = 0\} = \{\xi \in k^n : g(\xi) = 0\}$.) Since k is infinite, k^n is dense in K^n, hence, it is an irreducible topological space. Since Euclid's algorithm is "field independent", we have

$$D_k(n_1, \cdots, n_t) = D_k(n_1, \cdots, n_t) \cap J_k(n, m), \tag{6.4}$$

$$\varphi_k \upharpoonright D_k(n_1, \cdots, n_t) = \varphi_k \upharpoonright D_k(n_1, \cdots, n_t) \tag{6.5}$$

(disregarding k and K in front of an input or output). Therefore, $D_k(n_1, \cdots, n_t)$ is mapped onto $J_k(n_1, \cdots, n_t)$ under φ_K. Since k^{n+t} is dense in K^{n+t}, $J_k(n_1, \cdots, n_t)$ is dense in $J_K(n_1, \cdots, n_t)$ and therefore $D_k(n_1, \cdots, n_t)$ is dense in $D_K(n_1, \cdots, n_t)$, in particular, irreducible. Also W_k is dense in W_K.

COROLLARY 6.4. *Let k be an infinite field, $n \geqq m \geqq 0$ and let B be a computation tree which computes the Euclidean representation for $J_k(n, m)$ in time T. Assume that B computes also the Euclidean representation for $J_K(n, m)$. Then any $D_k(n_1, \cdots, n_t)$ contains a dense open subset U such that*

$$T \upharpoonright U \geqq n(H(n_1, \cdots, n_t) - 2).$$

Proof. Let B compute (φ_K, π_K) in time $\tilde{T}$. Then

$$\tilde{T} \upharpoonright J_k(n, m) = T,$$

and by Theorem 6.3 there is a dense open subset $\tilde{U}$ of $D_K(n_1, \cdots, n_t)$ such that

$$\tilde{T} \upharpoonright \tilde{U} \geqq n(H - 2).$$

Since $D_k(n_1, \cdots, n_t)$ is dense in $D_K(n_1, \cdots, n_t)$, $U := \tilde{U} \cap D_k(n_1, \cdots, n_t)$ is dense open in $D_k(n_1, \cdots, n_t)$. Moreover, since $U \subset J_k(n, m)$ we have

$$T \upharpoonright U = \tilde{T} \upharpoonright U \geqq n(H - 2).$$

Next we try to free ourselves from the assumption that B computes the Euclidean representation also over the algebraic closure of k. To this end we will have to estimate the degree of an unknown rational map that extends $\varphi_k \upharpoonright D_k(n_1, \cdots, n_t)$. A result of

this nature is proved (for a similar purpose) in Strassen [24]. A more powerful method is contained in the proof of Theorem 1 of Heintz–Sieveking [7]. The following lemma comes out of their approach.

LEMMA 6.5. *Let K be algebraically closed and $X \subset K^N$ be a Zariski closed set, all of whose components have the same dimension. Assume that there are polynomials $F_1, \cdots, F_z \in K[x_1, \cdots, x_N]$ of degree $\leqq d$ such that any component of X is also a component of $\{F_1 = \cdots = F_z = 0\}$. Then if $X \subset Y \subset K^N$ and Y is closed, we have*

$$\deg Y \geqq \deg X \cdot d^{\dim X - \dim Y}.$$

Proof. Induction on $\dim Y$: The start $\dim Y = \dim X$ being clear, let $\dim Y > \dim X$. Cleaning Y of superfluous components we may assume that any component of Y contains a component of X. Let $C_1, \cdots, C_r$ be the components of Y of highest dimension. If all F_i would vanish on a C_ρ, then a component of X contained in C_ρ would not be a component of $\{F_1 = \cdots = F_z = 0\}$. Thus, for each ρ there is an $i_0 \leqq z$ and a $c_\rho \in C_\rho$ such that $F_{i_\rho}(c_\rho) \neq 0$. The set

$$\left\{(\lambda_1, \cdots, \lambda_r) \in K^r : \exists \sigma \leqq r \sum_{\rho=1}^{r} \lambda F_{i_\rho}(c_\sigma) = 0\right\}$$

is a union of r proper subspaces of K^r, therefore, not all of K^r. So we can choose $\lambda_1, \cdots, \lambda_r \in K$ such that $\sum_{\rho \leqq r} \lambda_\rho F_{i_\rho}$ does not vanish on any C_σ. Let

$$\tilde{Y} = Y \cap \left\{\sum_{\rho \leqq r} \lambda_\rho F_{i_\rho} = 0\right\}. \tag{6.6}$$

Then $\dim \tilde{Y} < \dim Y$ and $X \subset \tilde{Y}$. The induction hypothesis applied to $\tilde{Y}$ yields

$$\deg \tilde{Y} \geqq \deg X \cdot d^{\dim X - \dim \tilde{Y}}.$$

On the other hand, (6.6) and Bezout's inequality (1.2) gives

$$\deg \tilde{Y} \leqq \deg Y \cdot d.$$

The two last inequalities together complete the induction.

THEOREM 6.6. *Let k be infinite and $\varepsilon > 0$. If the Euclidean representation for $J_k(n, m)$ is computable in time T, then any $D_k(n_1, \cdots, n_t)$ with $t \geqq (\frac{1}{2} + \varepsilon)m$ contains a Zariski dense open subset U such that*

$$T \restriction U \geqq 2\varepsilon n H(n_1, \cdots, n_t) - 5n.$$

Proof. By axiomatic induction (over k) one easily shows: Let a collection (φ, π) for $J_k(n, m)$ be computable in time T. Then there is a collection $(\hat{\varphi}, \hat{\pi})$ for $J_k(n, m)$, strictly computable in time $\hat{T}$, such that

$$\hat{\varphi} \restriction J_k(n, m) = \varphi, \tag{6.7}$$

(6.8) for any π-class D there are $\hat{\pi}$-classes, $\hat{D}_1, \cdots, \hat{D}_p$ such that

$$D = \bigcup_{1}^{p} (\hat{D}_i \cap J_k(n, m)),$$

$$T = \hat{T} \restriction J_k(n, m). \tag{6.9}$$

(Only Rule (III) with $\omega = /$ requires some care: One has to insert a test as to whether the denominator takes the value 0 or not.)

We apply this to the Euclidean representation (φ_k, π_k) for $J_k(n, m)$. Unfortunately, $(\hat{\varphi}, \hat{\pi})$ need not be the Euclidean representation for $J_K(n, m)$. Let

$$D_k(n_1, \cdots, n_t) = \bigcup_1^p (\hat{D}_i \cap J_k(n, m)). \tag{6.10}$$

Without loss of generality $\hat{D}_i \cap J_k(n, m) \neq \phi$ for all i. Let $\hat{E}_i$ be the closure of $\hat{D}_i$. Then (6.10) implies

$$D_k(n_1, \cdots, n_t) \subset \bigcup_1^p \hat{E}_i.$$

Since $D_k(n_1, \cdots, n_t)$ is irreducible, it is contained in some $\hat{E}_i$, say $D_k(n_1, \cdots, n_t) \subset \hat{E}_1$. Since $\hat{D}_1 \in \hat{\pi}$ is open in $\hat{E}_1$ by Theorem 6.1 and $\hat{D}_1 \cap D_k(n_1, \cdots, n_t) \neq \phi$, the irreducibility of $D_k(n_1, \cdots, n_t)$ implies that

$$U := \hat{D}_1 \cap D_k(n_1, \cdots, n_t) \text{ is dense open in } D_k(n_1, \cdots, n_t). \tag{6.11}$$

From (6.7) we see that

$$V := \text{graph } (\varphi_k \restriction U) \subset \text{graph } (\hat{\varphi} \restriction \hat{D}_1).$$

Equation (6.11) implies that V is dense in W_k, thus also in W_K. Therefore,

$$X := \bar{W}_K \subset \text{closure graph } (\hat{\varphi} \restriction \hat{D}_1) := Y.$$

Now we apply Lemma 6.5. We have $\dim X = n+t$, $\dim Y \leqq n+m+2$, $N = (n+m+2)+(n+t)$. By Lemma 6.2. 4 (with k replaced by K) we can take $d = t$, $z = n+m+2$. Thus,

$$\begin{aligned} \log \deg Y &\geqq \log \deg X - (m+2-t) \log t \\ &\geqq n(H-2) - (m+2-t) \log t \end{aligned}$$

by Lemma 6.2. 3. Now Theorem 6.1. 2 applied to $(\hat{\varphi}, \hat{\pi})$ and $\hat{D}_1$ yields

$$\hat{T} \restriction \hat{D}_1 \geqq n(H-2) - (m+2-t) \log t,$$

and therefore, by (6.11) and (6.9),

$$T \restriction U \geqq n(H-2) - (m+2-t) \log t.$$

Now, since $n_i/n \geqq 1/n$ for $2 \leqq i \leqq t-1$, we have

$$H \geqq \sum_{i=2}^{t-2} \frac{1}{n} \log n \geqq \frac{t-3}{n} \log n.$$

Thus, for $t \geqq (\frac{1}{2}+\varepsilon)m \geqq m/2$ we have $m \log n \leqq 2nH + 6 \log n$ and therefore,

$$\begin{aligned} T \restriction U &\geqq n(H-2) - (m+2-(\tfrac{1}{2}+\varepsilon)m) \log n \\ &= n(H-2) - (\tfrac{1}{2}-\varepsilon)m \log n - 2 \log n \\ &\geqq n(H-2) - (1-2\varepsilon)nH - 5 \log n \\ &\geqq 2\varepsilon nH - 5n. \end{aligned}$$

In practice, one frequently wants to compute the Euclidean representation over a field $\mathbb{Z}_p$ (or several such fields, see [3]). Typically, p is not known in advance. So we are led to consider the type $\Omega = \{0, 1, +, -, *, /\}$, $P = \{=\}$ and the following set of

inputs:

$$J'(n, m) = \bigcup_{p \text{ prime}} J_{\mathbb{Z}_p}(n, m). \tag{6.12}$$

A typical input $(\mathbb{Z}_p, a_0, \cdots, a_n, b_0, \cdots, b_m)$ is interpreted as a pair of polynomials $\sum_0^n a_i x^{n-i}$, $\sum_0^m b_j x^{m-j} \in \mathbb{Z}_p[x]$ of degrees n and m, respectively. Let (φ', π') be the Euclidean representation for $J'(n, m)$. If we put

$$D'(n_1, \cdots, n_t) = \bigcup_{p \text{ prime}} D_{\mathbb{Z}_p}(n_1, \cdots, n_t), \tag{6.13}$$

we have

$$\varphi' = \bigcup_{p \text{ prime}} \varphi_{\mathbb{Z}_p},$$

$$\pi' = \left\{ D'(n_1, \cdots, n_t) : t \geqq 2,\ n_1 \geqq 0,\ n_2, \cdots, n_{t-1} > 0,\ n_t \geqq 0,\ \sum_1^t n_i = n,\ \sum_2^t n_i = m \right\}.$$

Given $n \geqq m \geqq 0$, Theorem 4.3 applies not only to infinite ground fields, but also to fields $\mathbb{Z}_p$ with p sufficiently large. Using a table-look-up procedure for small $\mathbb{Z}_p$, we obtain:

(6.14) The Euclidean representation (φ', π') for $J'(n, m)$ is computable in time

$$30n(H(n_1, \cdots, n_t) + 6.5).$$

If we dismiss in the Knuth–Schönhage algorithm the symbolic multiplication of polynomials by interpolation in favor of a slower direct method, the algorithm works for any $\mathbb{Z}_p$. Thus, (φ', π') is also strictly computable.

$J'(n, m)$ carries a Zariski topology, a basis being given by the sets

$$U_f := \{(\mathbb{Z}_p, \mathbf{a}, \mathbf{b}) \in J'(n, m) : f(\mathbf{a}, \mathbf{b}) \neq 0 \text{ in } \mathbb{Z}_p\},$$

where $f \in \mathbb{Z}[x_1, \cdots, x_{n+m+2}]$. Since $U_f \neq \phi$ for $f \neq 0$, and $U_f \cap U_g = U_{fg}$, any two non-empty open sets intersect. Thus $J'(n, m)$ is irreducible. Similarly,

$$J'(n_1, \cdots, n_t) := \bigcup_{p \text{ prime}} J_{\mathbb{Z}_p}(n_1, \cdots, n_t) \tag{6.15}$$

is irreducible in its Zariski topology.

LEMMA 6.7. 1. *Let a collection (φ, π) for $J'(n, m)$ be strictly computable. Then any $D \in \pi$ is locally closed and*

$$\varphi \restriction D : D \to \bigcup_{p \text{ prime}} \{\mathbb{Z}_p\} \times \mathbb{Z}_p^q$$

is Zariski continuous.

2. $$\varphi' : D'(n_1, \cdots, n_t) \to J'(n_1, \cdots, n_t)$$

is a homeomorphism. In particular, $D'(n_1, \cdots, n_t)$ is irreducible.

Proof. 1. Similar to the proof of Theorem 6.1. 1.

2. By 1,

$$\varphi' : D'(n_1, \cdots, n_t) \to \bigcup_p \{\mathbb{Z}_p\} \times \mathbb{Z}_p^{n+t}$$

is continuous. But φ' maps $D'(n_1, \cdots, n_t)$ into $J'(n_1, \cdots, n_t)$, whose Zariski topology is induced from the Zariski topology of $\bigcup_p \{\mathbb{Z}_p\} \times \mathbb{Z}_p^{n+t}$. Thus,

$$\varphi' : D'(n_1, \cdots, n_t) \to J'(n_1, \cdots, n_t)$$

is continuous. It is also bijective and its inverse is continuous (see (3.4)).

It follows easily from Lemma 6.7. 2, that all $D'(n_1, \cdots, n_t)$ are infinite and that any Zariski dense open subset U of $D'(n_1, \cdots, n_t)$ has asymptotic density one (along the decomposition (6.13)).

THEOREM 6.8. *Let $n \geqq m \geqq 0$ and B be a computation tree that computes the Euclidean representation for $J'(n, m)$ in time T. Assume that B also computes the Euclidean representation for $J_{\mathbb{C}}(n, m)$. Then any $D'(n_1, \cdots, n_t)$ contains a dense open subset U' such that*

$$T \upharpoonright U' \geqq n(H(n_1, \cdots, n_t) - 2).$$

Proof. We apply B to the "combined" input set

$$J''(n, m) := J'(n, m) \cup J_{\mathbb{C}}(n, m).$$

$J''(n, m)$ also has a Zariski topology (defined by polynomials $\in \mathbb{Z}[x_1, \cdots, x_{n+m+2}]$). The Zariski topology on $J'(n, m)$ is induced by that of $J''(n, m)$, and the Zariski topology on $J_{\mathbb{C}}(n, m)$ is finer than the induced topology. Similarly for

$$J''(n_1, \cdots, n_t) = J'(n_1, \cdots, n_t) \cup J_{\mathbb{C}}(n_1, \cdots, n_t).$$

B computes the Euclidean representation (φ'', π'') on $J''(n, m)$. Lemma 6.7 also holds for this situation, i.e.,

$$\varphi'' : D''(n_1, \cdots, n_t) \to J''(n_1, \cdots, n_t)$$

is a homeomorphism. φ'' restricts to φ' on $D'(n_1, \cdots, n_t)$ and to $\varphi_{\mathbb{C}}$ on $J_{\mathbb{C}}(n_1, \cdots, n_t)$. Since, obviously, $J_{\mathbb{C}}(n_1, \cdots, n_t)$ is dense in $J''(n_1, \cdots, n_t)$, $D_{\mathbb{C}}(n_1, \cdots, n_t)$ is dense in $D''(n_1, \cdots, n_t)$. Similarly $D'(n_1, \cdots, n_t)$ is dense in $D''(n_1, \cdots, n_t)$. Let T_1 be the cost of B on $J''(n, m)$. Then

$$T_1 \upharpoonright J'(n, m) = T. \tag{6.16}$$

Each $D''(n_1, \cdots, n_t)$ is subdivided into locally closed sets D_i, on which T_1 is constant. Since $D''(n_1, \cdots, n_t)$ is irreducible, one of the D_i is dense open in $D''(n_1, \cdots, n_t)$, say D_1. Then $D_1 \cap D_{\mathbb{C}}(n_1, \cdots, n_t)$ is $\neq \phi$ and open in $D_{\mathbb{C}}(n_1, \cdots, n_t)$ with respect to the induced topology, thus, also with respect to the Zariski topology. By Theorem 6.3 there is a dense open U in $D_{\mathbb{C}}(n_1, \cdots, n_t)$ such that

$$T_1 \upharpoonright U \geqq n(H - 2).$$

We have $U \cap D_1 \neq \phi$ and therefore

$$T_1 \upharpoonright D_1 \geqq n(H - 2). \tag{6.17}$$

Now $U' := D_1 \cap D'(n_1, \cdots, n_t)$ is dense open in $D'(n_1, \cdots, n_t)$, and by (6.16) and (6.17), we have

$$T \upharpoonright U' \geqq T_1 \upharpoonright U' \geqq n(H - 2).$$

THEOREM 6.9. *Let $\varepsilon > 0$, $n \geqq m \geqq 0$, and let the Euclidean representation for $J'(n, m)$ be computable in time T. Then any $D'(n_1, \cdots, n_t)$ with $t \geqq (\frac{1}{2} + \varepsilon)m$ contains a dense open subset U' such that*

$$T \upharpoonright U' \geqq 2\varepsilon n H(n_1, \cdots, n_t) - 5n.$$

Proof. Analogous to the proof of Theorem 6.8, with $\mathbb{C}$ replaced by $\mathbb{Q}$ and using Theorem 6.6 instead of Theorem 6.3. One only has to show that a computation tree which computes the Euclidean representation for $J'(n, m)$ also computes the Euclidean representation for $J_{\mathbb{Q}}(n, m)$. This is achieved by looking at large p.

REFERENCES

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

[2] A. BORODIN AND I. MUNRO, *Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[3] W. S. BROWN, *On Euclid's algorithm and the computation of polynomial greatest common divisors*, J. Assoc. Comput. Mach., 18 (1971), pp. 478–504.

[4] G. E. COLLINS, *Subresultants and reduced polynomial remainder sequences*, J. Assoc. Comput. Mach., 14 (1967), pp. 128–142.

[5] R. M. FANO, *Transmission of Information, a Statistical Theory of Communications*, John Wiley, New York, 1961.

[5'] W. HABICHT, *Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens*, Commentarii Mathematici Helvetici, 21, 2 (1948), pp. 99–116.

[6] R. HARTSHORNE, *Algebraic Geometry*, Springer, New York, 1977.

[6'] J. HEINTZ, *Definability bounds of first order theories of algebraically closed fields*, Fundamentals of Computation Theory, FCT 79, Berlin, 1979, pp. 160–166.

[7] J. HEINTZ AND M. SIEVEKING, *Lower bounds for polynomials with algebraic coefficients*, Theor. Comput. Science, 11 (1980), pp. 321–330.

[8] D. E. KNUTH, *The analysis of algorithms*, Proc. Internat. Congress Math. (Nice, 1970), Vol. 3, Gauthier-Villars, Paris, 1971, pp. 269–274.

[9] H. T. KUNG, *On computing reciprocals of power series*, Numer. Math., 22 (1974), pp. 341–348.

[10] D. H. LEHMER, in Amer. Math. Monthly, 45 (1937), pp. 227–233.

[11] R. MOENCK, *Fast computation of* GCD's, Proc. Fifth Symposium on Theory of Computing, ACM, New York, 1973, pp. 142–151.

[12] D. MUMFORD, *Introduction to Algebraic Geometry*, Chap. I, Harvard University Cambridge, MA (mimeogr. notes).

[13] O. PERRON, *Die Lehre von den Kettenbrüchen*, Teubner, Berlin, 1913.

[14] P. SAMUEL, *Méthodes d'algèbre abstraite en géométrie algebrique*, Springer, New York, 1967.

[15] C. P. SCHNORR, *An extension of Strassen's degree bound*, Proc. of the FCT Conference, Berlin/Wendisch-Rietz, 1979, L. Budach, ed. Akademie-Verlag, Berlin, pp. 404–416, 1979.

[16] A. SCHÖNHAGE, *Schnelle Berechnung von Kettenbruchentwicklungen*, Acta Inform., 1 (1971), pp. 139–144.

[17] ———, *An elementary proof of Strassen's degree bound*, Theor. Comput. Science, 3 (1976), pp. 267–272.

[18] I. R. SHAFAREVICH, *Basic Algebraic Geometry*, Part I, Springer, New York, 1974.

[19] M. SIEVEKING, *An Algorithm for Division of Power Series*, Computing, 10 (1972), pp. 153–156.

[20] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–456.

[21] ———, *Berechnung und Programm I*, Acta Inform., 1 (1972), pp. 320–335.

[22] ———, *Berechnung und Programm II*, ibid., 2 (1973), pp. 64–79.

[23] ———, *Die Berechnungskomplexität von elementarsymmetrischen Funktionen und von Interpolationskoeffizienten*, Numer. Math., 20 (1973), pp. 238–251.

[24] ———, *Computational complexity over finite fields*, this Journal, 5 (1976), pp. 324–331.

[25] ———, *Some results in algebraic complexity theory*, Proc. Internat. Congress Math., Vancouver, 1974.

[26] H. S. WALL, *Analytic Theory of Continued Fractions*, Van Nostrand, New York, 1948.

[27] B. L. VAN DER WAERDEN, *Einführung in die Algebraische Geometrie*, Springer, Berlin, 1973.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0002 $01.25/0

OPTIMAL SEARCH IN PLANAR SUBDIVISIONS*

DAVID KIRKPATRICK†

Abstract. A planar subdivision is any partition of the plane into (possibly unbounded) polygonal regions. The subdivision search problem is the following: given a subdivision S with n line segments and a query point P, determine which region of S contains P. We present a practical algorithm for subdivision search that achieves the same (optimal) worst case complexity bounds as the significantly more complex algorithm of Lipton and Tarjan, namely $O(\log n)$ search time with $O(n)$ storage. Our subdivision search structure can be constructed in linear time from the subdivision representation used in many applications.

Key words. computational geometry, analysis of algorithms, point location, planar graphs, hierarchical search

1. Introduction.

Any finite collection of finite, semi-infinite or infinite line segments induces a partition of the plane into polygonal regions. We will restrict our attention, for the present, to collections of line segments whose pairwise intersections are restricted to segment endpoints. We call such a collection (or the finite set of polygonal regions induced by the collection) a *(planar) subdivision.*

We define the *subdivision search problem* to be the following: Given a subdivision S with n line segments and an arbitrary query point P, determine which region of S contains P. Our subdivision search problem is equivalent to the "region-searching" problem of Dobkin and Lipton [6]. It is a slight (but, as we shall see, inconsequential) generalization of both the "point-location" problem studied by Lee and Preparata [14] and the "triangle" problem of Lipton and Tarjan [18]. The "point in polygon" problem [1], [3], [24] (given a simple polygon, does it contain a specified query point?), the "rectangle searching" problem [27] (given a set of nonoverlapping rectangles, which, if any, contains a specified query point?), and the "line searching" problem [6] (given a set of lines in the plane, which, if any, contains a specified query point?) can all be formulated as instances of our subdivision search problem.

Dobkin and Lipton [6] were the first to cast Knuth's [12] "post-office" problem (given a set of points in the plane, which is closest to a specified query point?) as a subdivision search problem. Shamos [25] (and independently Dewdney [4]) refined this formulation by introducing the Voronoi diagram of a point set, a planar subdivision of remarkable utility in connection with nearest neighbor and other related problems.

In many applications, a planar subdivision is the object of numerous location queries. For this reason, algorithms for point location are generally characterized by three attributes: i) *preprocessing time*—the time required to construct a search structure from a standard representation of S; ii) *space*—the storage used in the construction and representation of the search structures; and iii) *search time*—the time required to locate a specified query point, given the search structure. We restrict our attention here to the worst-case behaviour of these attributes.

Dobkin and Lipton [6] employ a projective technique to reduce subdivision search to linear search. The resulting algorithm is asymptotically optimal (among comparison-based algorithms) in terms of search time but may be quite expensive in terms of both preprocessing time and storage. Specifically, Dobkin and Lipton provide an $O(\lg n)$[1]

* Received by the editors November 24, 1981. This work was supported in part by the National Sciences and Engineering Research Council of Canada, grant A3593.

† Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5.

[1] lg denotes $\log_2$.

search-time, $O(n^2)$ space, and $O(n^2 \lg n)$ preprocessing-time algorithm for subdivision searching. Dobkin and Lipton were also the first to raise the question: Can subdivision searching be done with $O(\lg n)$ search-time and $O(n)$ (or even $O(n \lg n)$) space?

Shamos [25] introduces an $O((\lg n)^2)$ search-time, $O(n)$ space, and $O(n \lg n)$ preprocessing-time algorithm suitable for searching a class of subdivisions that includes Voronoi diagrams. Employing an $O(n \lg n)$ algorithm for constructing a Voronoi diagram on n points [28], this leads to an $O((\lg n)^2)$ search-time, $O(n)$ space, and $O(n \lg n)$ preprocessing-time solution of the "post-office" problem. Shamos' algorithm is generalized by Lee and Preparata [15] to an $O((\lg n)^2)$ search-time, $O(n)$ space, and $O(n \lg n)$ preprocessing-time algorithm for the location in arbitrary subdivisions. Lee and Preparata's approach is divide-and-conquer; each reduction of the subdivision is achieved by discrimination of the query point with respect to a monotone chain of edges that splits the subdivision (at a cost of $O(\lg n)$ comparisons, in the worst case).

The first affirmative answer to the question of Dobkin and Lipton was provided by Lipton and Tarjan [18]. Lipton and Tarjan's $O(\lg n)$ search time, $O(n)$ space, and $O(n \lg n)$ preprocessing time algorithm for search in arbitrary triangular subdivisions (each interior region of the subdivision is bounded by exactly three line segments) is one of many important applications of their planar separator theorem [18], [19]. That general subdivision search can be efficiently reduced to triangular subdivison search follows from the $O(n \lg n)$ polygon triangulation algorithm of Garey et al. [8]; the details of this reduction are discussed in § 4. Unfortunately, Lipton and Tarjan's algorithm is of primarily theoretical interest; to quote Lipton and Tarjan [19], "We do not advocate this algorithm as a practical one, but its existence suggests that there may be a practical algorithm with an $O(\lg n)$ time bound and $O(n)$ space bound".

A recent result of Preparata [21] claims to come "very close to providing a complete substantiation" of Lipton and Tarjan's conjecture. Preparata's algorithm, which he describes as an evolution of the approach of Dobkin and Lipton [6], uses $O(\lg n)$ search time, $O(n \lg n)$ space, and $O(n \lg n)$ preprocessing time.

The purpose of this paper is to affirm Lipton and Tarjan's conjecture; we present a new subdivision search algorithm with exactly the same asymptotic bounds as Lipton and Tarjan's algorithm. The simplicity of our approach (and the existence of an implementation) suggests that it may also deserve to be called practical. A discussion of the implementation and more detailed evaluation of our algorithm will be presented elsewhere.

In the next section we present some preliminary definitions and comments on the data structures used by our algorithm. Sections 3, 4 and 5 describe our algorithm and a number of its applications. Section 6 concludes the paper with a discussion of some related open problems.

2. Definitions and preliminaries. A *finite planar subdivision* is a planar subdivision each of whose line segments is finite. Such a subdivision is indistinguishable from a straight-line embedding of a planar graph. Thus we can refer without confusion, not only to the vertices, edges and regions (or faces) of a finite planar subdivision, but also to such graph-theoretic notions as degree, incidence and independence [10]. It is an immediate consequence of Euler's formula (cf. [10]) that the numbers of vertices and edges of a finite planar subdivision are linearly related, and hence either number serves to characterize the size of such a subdivision. Hereafter, $|S|$ will denote the number of vertices of the finite subdivision S.

Let S be a finite planar subdivision. We take as a starting point for our algorithm what we call an *edge-ordered representation* of S. Specifically:

(a) if x is a line segment joining vertex v to vertex w, then x is represented by the pair of directed edges $\{(v, w), (w, v)\}$;

(b) each vertex v has associated with it not only its coordinates but also a list, in counterclockwise order, of all directed edges whose source is v; and
(c) each directed edge (v, w) has associated with it a pointer to the edge (w, v) as well as the name of the region lying immediately to the right of (v, w).

An edge-ordered representation is provided either implicitly or explicitly by the representations taken as standard in a number of earlier papers [20], [23]. It differs from the basic (unordered) list of adjacencies chosen by Lee and Preparata [15] as their initial representation. However, it should be clear that:

i) it occupies $O(|S|)$ space;
ii) it can be constructed in $O(|S| \lg |S|)$ time from a list of adjacencies or other standard representations of planar graphs;
iii) it can be constructed in $O(|S|)$ time if the underlying planar graph has bounded degree; and
iv) it can be constructed in $O(|S|)$ time from the natural graph representation provided in certain applications (cf. § 5).

Thus our choice of representation for subdivisions is intended to allow a realistic estimate of actual preprocessing costs.

The obvious redundancy in an edge-ordered representation can be neatly exploited in the development of our hierarchical search structure. A detailed description of the data structures used in one efficient implementation or our algorithm will be presented elsewhere.

A finite planar subdivision S has exactly one unbounded region, called the *external region* of S. Its complement is called the *interior* of S. The edges bounding the external region define what we call the *boundary* of S.

A *convex subdivision* is any finite planar subdivision whose interior is convex and whose interior regions are all convex. A *triangular subdivision* is a special case of a convex subdivision in which each region (including the exterior region) is bounded by three line segments. It is easily confirmed that a triangular subdivision on $n \geqq 3$ vertices has exactly $3n-6$ edges and $2n-4$ regions (including the external region).

In § 3, we give a new constructive proof of the following:

THEOREM 3.1. *There is an $O(\lg n)$ search time, $O(n)$ space and $O(n)$ preprocessing time algorithm for the triangular subdivison search problem.*

This result is extended to arbitrary planar subdivisions in § 4.

3. Fast search in triangular subdivisions. Let S be an arbitrary triangular subdivision with n vertices. A *subdivision hierarchy* associated with S is a sequence $S_1, S_2, \cdots, S_{h(n)}$ of triangular subdivisions, where $S_1 = S$ and each region R of S_{i+1} is linked to each region R' of S_i for which $R' \cap R \neq \varnothing$ (the so-called *parents* of R in S_i), for $1 \leqq i < h(n)$. We call $h(n)$ the *height* of the subdivision hierarchy. Obviously the space required for a subdivision hierarchy is just the space required for the individual subdivisions ($O(\sum_{i=1}^{h(n)} |S_i|)$) plus the space used by the intersubdivision links.

Our basic point location algorithm involves a single pass through the subdivision hierarchy, locating the test point at each level. Let p denote an arbitrary test point.

ALGORITHM HIERARCHICAL SUBDIVISION SEARCH
CANDIDATES$_{h(n)} \leftarrow$ regions of $S_{h(n)}$
$R \leftarrow$ region in CANDIDATES$_{h(n)}$ containing p
$i \leftarrow h(n)-1$
while $i > 0$ **do**
    CANDIDATES$_i \leftarrow$ parents(R)
    $R \leftarrow$ region in CANDIDATES$_i$ containing p
    $i \leftarrow i-1$
report (region R)

Since membership in any triangular region can be tested in constant time, the complexity of this search procedure is $O(\sum_{i=1}^{h(n)}|\text{CANDIDATES}_i|)$. Obviously we are motivated to construct subdivision hierarchies in which both the height and the size of all CANDIDATE sets are minimized.

We start by constructing a subdivision hierarchy of height two.

LEMMA 3.1. *There exist positive constants c and d such that for any triangular subdivision S with* $|S|>3$, *a triangular subdivision T can be constructed in* $O(|S|)$ *time, satisfying*:

i) $|T|\leqq(1-1/c)|S|$, *and*

ii) *each region of T has at most d parents in S.*

Proof. Let v be any internal (nonboundary) vertex of S, and let $\deg(v)$ denote its degree. Then, exactly $\deg(v)$ regions of S are incident with v. The union of these regions, which we call the *neighborhood* of v, forms a star-shaped polygonal region with $\deg(v)$ bounding edges. Now, if v and its $\deg(v)$ incident edges are removed from S and the neighborhood of v is retriangulated (introducing $\deg(v)-3$ new edges) what results is a new triangular subdivision with $|S|-1$ vertices. It should be clear that, regardless of how the neighborhood of v is retriangulated, each new region intersects at most $\deg(v)$ regions of S. Of course, the simplification achieved by this vertex removal and retriangulation is minimal. However, if w is any vertex which is independent of (i.e. nonadjacent to) v in S, then the neighborhoods of v and w do not intersect except possibly along one or more edges of S. Hence, such a pair of vertices can be removed in parallel and the triangular subdivision that is created by retriangulating their vacated neighborhoods has the property that each of its regions intersects at most $\max\{\deg(v), \deg(w)\}$ regions of S. By identical reasoning, if $v_1, \cdots, v_t$ form an independent set of vertices in S, then the $|S|-t$ vertex triangular subdivision T formed by removing $v_1, \cdots, v_t$ and retriangulating all t vacated neighborhoods has the property that each of its regions intersects at most $\max\{\deg(v_j),\ 1\leqq j\leqq t\}$ regions of S. To complete the proof it suffices to show that if c and d are sufficiently large then an independent set $v_1, \cdots, v_t$ with $\deg(v_i)\leqq d$, $1\leqq i\leqq t$, and $t\geqq|S|/c$ can always be identified in $O(|S|)$ time. This is an immediate consequence of the following lemma. □

LEMMA 3.2. *There exist positive constants c and d such that every planar graph on n vertices has at least n/c independent vertices of degree at most d. Furthermore, at least n/c of these can be identified in* $O(n)$ *time.*

Proof. We make no attempt to optimize c and d here. (Their optimal values influence the asymptotic constants for each of space, preprocessing time, and search time, and some tradeoffs can be expected.) We have already noted that an n-vertex planar graph has at most $3n-6$ edges. Hence the average vertex degree is less than 6, and so less than half of the vertices have degree exceeding 11. Starting with the set V of vertices of degree at most 11 (which can be identified easily in linear time), a straightforward elimination procedure identifies an independent subset containing at least $|V|/12\geqq n/24$ vertices. □

Of course, a subdivision hierarchy of height two provides no significant simplification over the original subdivision. However, if Lemma 3.1 is applied iteratively, we are led to a subdivision hierarchy in which asymptotically improved (in fact, optimal) search is possible.

LEMMA 3.3. *There exist positive constants c and d such that, for any triangular subidivision S with n vertices, an associated subdivision hierarchy* $S_1, \cdots, S_{h(n)}$ *can be constructed in* $O(n)$ *time, satisfying*:

i) $|S_{h(n)}|=3$;

ii) $|S_{i+1}|\leqq(1-1/c)|S_i|$; *and*

iii) *each region of* S_{i+1} *has at most d parents in* S_i.

Proof. Immediate from Lemma 3.1. □

COROLLARY 3.1. *The subdivision hierarchy above has height* $h(n) = O(\lg n)$ *and uses* $O(n)$ *space in total.*

Proof. It suffices to note that the sequence $|S_1|, |S_2|, \cdots, |S_{h(n)}|$ forms a decreasing geometric progression. □

We now restate and prove our basic result.

THEOREM 3.1. *There is an* $O(\lg n)$ *search time,* $O(n)$ *space, and* $O(n)$ *preprocessing-time algorithm for the triangular subdivision search problem.*

Proof. We use the hierarchical subdivision search algorithm in conjunction with the subdivision hierarchy constructed in Lemma 3.3. By Lemma 3.3, the preprocessing time is $O(n)$. By Corollary 3.1, the total space is $O(n)$. By our earlier observations, the complexity of search is $O(|S_{h(n)}| + \sum_{i=1}^{h(n)-1} p_i)$ where $p_i = \max_{R \in S_{i+1}} (|\text{parents } (R)|)$. But, by Lemma 3.1 and Corollary 3.1, $S_{h(n)}$ and p_i are $O(1)$ and $h(n)$ is $O(\lg n)$, so the search time is $O(\lg n)$. □

4. Fast search in general subdivisions. In this section we consider the reduction of general subdivision searching problems to triangular subdivision search. Let S be an arbitrary planar subdivision. We can reduce the question of searching in S to searching in a finite planar subdivision by intersecting S with a sufficiently large triangle chosen to contain all intersections of line segments of S. The interior of this triangle is clearly a finite planar subdivision. The exterior can be searched using a straightforward generalization of binary search, exploiting the fact that none of the semi-infinite line segments intersect in this region. This reduction adds a factor of only $O(|S|)$ to both the preprocessing time and space and $O(\lg |S|)$ to the search time used in the resulting finite subdivision search problem. Hence the asymptotic complexities of general and finite subdivision searching are equivalent.

It remains to reduce finite subdivision searching to triangular subdivision searching. Let S be a finite planar subdivision. We can assume from the preceding reduction that the boundary of S is triangular. Let T be the subdivision formed from S by triangulating each interior region of S. The size of T remains proportional to the size of S and, since T is a refinement of S, the location of points in T immediately implies their location in S. In the general case T can be formed from S in time $O(|S| \lg |S|)$, using the general polygon triangulation algorithm of Garey et al. [8]. However, if the regions of S are all convex, or even star-shaped, a straightforward linear algorithm exists for constructing T. Thus, we have demonstrated the following:

THEOREM 4.1. *There is an* $O(\lg n)$ *search time,* $O(n)$ *space and* $O(n \lg n)$ *preprocessing time algorithm for the general subdivision search problem.*

THEOREM 4.2. *There is an* $O(\lg n)$ *search time,* $O(n)$ *space and* $O(n)$ *preprocessing time algorithm for the convex subdivision search problem.*

5. Applications. Earlier papers on subdivision search, notably [15], [21], have mentioned a number of applications. We recall and expand on a few of these here.

5.1. Point in polygon problem. A planar polygon is a special case of a finite planar subdivision. Theorem 4.1 gives an immediate $O(\lg n)$ search time, $O(n \lg n)$ preprocessing time and $O(n)$ space algorithm for testing the inclusion of an arbitrary point in an n vertex planar polygon. For convex or star-shaped polygons, or any other family of polygons that can be triangulated in $O(n)$ time, the preprocessing time is linear.

5.2. Point in convex polyhedron problem. Lee and Preparata [15] note that the problem of testing the inclusion of an arbitrary point in an n-vertex convex polyhedron can be reduced to convex subdivision search with $O(n)$ preprocessing. It follows, by

Theorem 4.2, that an $O(\lg n)$ search time, $O(n)$ preprocessing time and $O(n)$ space algorithm exists for the point in convex polyhedron problem.

By dualization, an algorithm with identical attributes can be formulated for the problem of testing for the intersection of an arbitrary plane and a polyhedron in 3-space [5].

5.3. Locating a set of points in a planar subdivision. Preparata [22] shows that a set of k points can be located on an n-vertex planar subdivision in $O(k \lg k + n + k \lg n)$ time, given $O(n \lg n)$ preprocessing time. This result is an immediate consequence of Theorem 4.1. Furthermore, by Theorem 4.2, the preprocessing time can be reduced to $O(n)$ for convex planar subdivisions (which arise in a principal application [20] of Preparata's batched point location algorithm).

5.4. Closest point problems. The problem of determining which of a set of data points is closest to a given test point has been extensively studied. Shamos [25] (and independently Dewdney [4]) show how this problem can be reduced to point location in a particular family of planar subdivisions known as Voronoi diagrams. Voronoi diagrams (in any L_p metric) can be constructed in $O(n \lg n)$ time [13]. While Voronoi diagrams in arbitrary L_p metrics may involve curved edges, every region is star-shaped and hence Voronoi point location can be solved using subdivision search followed by at most one test against a curved edge. Furthermore, only linear preprocessing is required following the construction of the Voronoi diagram. This fact can be exploited in the dynamic maintenance of Voronoi diagrams and dynamic solution of closest point problems [9].

By replacing Voronoi diagrams by what are called generalized Voronoi diagrams [11], [14] it is possible to use an analogous approach to solve the closest line problem (which of a set of lines or line segments is closest to a given test point?).

6. Open problems and conclusions. It is tempting to extend the approach of this paper to the location of points in higher-dimensional subdivisions. Such an extension is by no means obvious. The number of vertices, edges, faces and regions of three-dimensional subdivisions are not necessarily linearly related, and the analogue of triangulation (tetrahedralization) is not a straightforward process. A more detailed discussion of subdivision search in higher dimensions will be taken up elsewhere.

Our algorithm seems to depend on the fact that the given subdivision is formed out of straight line segments. While the algorithm can be adapted to certain other situations (for example, when all internal regions are star shaped), the general problem of optimal search in subdivisions formed from arbitrary curve segments may require a totally new approach. As a concrete example of such a subdivision, consider those subdivisions which arise in the so-called locus approach to the fixed-radius nearest neighbor search problem [2]. Such subdivisions are formed by the intersection of fixed-radius circles, and in general do not seem to admit a simple refinement using straight edges. Thus the fixed-radius nearest neighbor search problem still awaits an $O(\lg n)$ search time, $O(n^2)$ space and $O(n^2 \lg n)$ preprocessing time solution. A solution using $O(\log n)$ search time, $O(n^2 \log n)$ space and $O(n^2 \log n)$ preprocessing time is a byproduct of Preparata's subdivision search algorithm [21]. Edelsbrunner and Maurer [7] present search algorithms for subdivisions formed by segments other than straight lines.

We have described a new subdivision search algorithm which, as pointed out by Lipton and Tarjan [18], is optimal for both search time and space, assuming only binary decisions are possible. Our algorithm is based on the hierarchical decomposition

of an arbitrary subdivision. It is conjectured that this technique will find a number of other applications in computational geometry and elsewhere. On this point we should acknowledge the fact that this technique does not originate with this paper; Lipton and Miller [17] use a very similar idea in developing a fast algorithm for coloring planar graphs.

REFERENCES

[1] J. L. BENTLEY AND W. CARRUTHERS, *Algorithms for testing the inclusion of points in polygons*, in Proc. 18th Annual Allerton Conference on Communication, Control and Computing, 1980, pp. 11–19.

[2] J. L. BENTLEY AND H. A. MAURER, *A note on Euclidean near neighbour searching in the plane*, Inform. Process. Lett., 8 (1979), pp. 133–136.

[3] W. BURTON, *Representation of many-sided polygons and polygonal lines for rapid processing*, Comm. ACM, 16 (1973), pp. 230–236.

[4] A. K. DEWDNEY, *Complexity of nearest neighbour searching in three and higher dimensions*, Rep. 28, Dept. Computer Science, Univ. of Western Ontario, London, Ontario, 1977.

[5] D. P. DOBKIN AND D. G. KIRKPATRICK, *Fast detection of polyhedral intersections*, Proc. 9th International Colloquium on Automata, Languages and Programming, Aarhus, Denmark, 1982, to appear.

[6] D. P. DOBKIN AND R. J. LIPTON, *Multidimensional searching problems*, this Journal, 5 (1976), pp. 181–186.

[7] H. EDELSBRUNNER AND H. A. MAURER, *On region location in the plane*, Rep. 52, Institut für Informationsverarbeitung, Technical Univ. Graz, Graz, Austria, 1980.

[8] 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.

[9] I. G. GOWDA, D. G. KIRKPATRICK, D. T. LEE AND A. NAAMAD, *Dynamic Voronoi diagrams*, IEEE Trans. Inform. Theory, to appear.

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

[11] D. G. KIRKPATRICK, *Efficient computation of continuous skeletons*, in Proc. 20th Annual IEEE Symposium on Foundations of Computer Science, 1979, pp. 18–27.

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

[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 R. L. DRYSDALE III, *Generalization of Voronoi diagrams in the plane*, this Journal, 10 (1981), pp. 73–87.

[15] D. T. LEE AND F. P. PREPARATA, *Location of a point in a planar subdivision and its applications*, this Journal, 6 (1977), pp. 594–606.

[16] D. T. LEE AND C. C. YANG, *Location of multiple points in a planar subdivision*, Inform. Process. Lett. 9 (1979), pp. 190–193.

[17] R. J. LIPTON AND R. E. MILLER, *A batching method for coloring planar graphs*, Inform. Process. Lett., 7 (1978), pp. 185–188.

[18] R. J. LIPTON AND R. E. TARJAN, *Applications of a planar separator theorem*, in Proc. 18th Annual IEEE Symposium on Foundations of Computer Science, 1977, pp. 162–170.

[19] ———, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[20] D. E. MULLER AND F. P. PREPARATA, *Finding the intersection of two convex polyhedra*, Theoret. Comput. Sci., 7 (1978), pp. 217–236.

[21] F. P. PREPARATA, *A new approach to planar point location*, this Journal, 10 (1981), pp. 473–482.

[22] ———, *A note on locating a set of points in a planar subdivision*, this Journal, 8 (1979), pp. 542–545.

[23] 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.

[24] K. B. SALOMON, *An efficient point-in-polygon agorithm*, Comput. Geosci., 4 (1978), pp. 173–178.

[25] M. I. SHAMOS, *Geometric complexity*, in Proc. 7th Annual ACM Symposium on Theory of Computing, 1975, pp. 224–233.

[26] ———, *Computational geometry*, Ph.D. Thesis, Yale Univ., New Haven, CT, 1978.

[27] M. I. SHAMOS AND J. L. BENTLEY, *Optimal algorithms for structuring geographic data*, in Proc. Symposium on Topological Data Structures for Geographic Information Systems, Harvard Univ., Cambridge, MA, 1977, pp. 43–51.

[28] M. I. SHAMOS AND D. HOEY, *Geometric intersection problems*, in Proc. 17th Annual IEEE Symposium on Foundations of Computer Science, 1976, pp. 208–215.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0003 $01.25/0

TOOLS FOR TEMPLATE DEPENDENCIES*

RONALD FAGIN†, DAVID MAIER‡, JEFFREY D. ULLMAN§
AND MIHALIS YANNAKAKIS¶

Abstract. Template dependencies (TD's) are a class of data dependencies that include multivalued and join dependencies and embedded versions of these. A collection of techniques, examples and results about TD's are presented. The principal results are:

1) Finite implication (implication over relations with a finite number of tuples) is distinct from unrestricted implication for TD's.

2) There are, for TD's over three or more attributes, infinite chains of increasingly weaker and increasingly stronger full TD's.

3) However, there are weakest (nontrivial) and strongest full TD's over any given set of attributes.

4) Over two attributes, there are only three distinct TD's.

5) There is no weakest (not necessarily full) TD over any set of three or more attributes.

6) There is a finite relation that obeys every strictly partial TD but no full TD.

7) The conjunction of each finite set of full TD's is equivalent to a single full TD. However, the conjunction of a finite set of (not necessarily full) TD's is not necessarily equivalent to a single TD and the disjunction of a finite set of full TD's is not necessarily equivalent to a single TD.

8) There is a finite set of TD's with an infinite Armstrong relation but no finite Armstrong relation.

9) A necessary and sufficient condition for the existence of finite Armstrong relations for sets of TD's can be formulated in terms of the implication structure of TD's.

Key words. relational database, template dependency, finite implication, multivalued dependency, join dependency

1. Introduction. Template dependencies (TD's) were introduced by Sadri and Ullman [SU] and, independently, by Beeri and Vardi [BV2]. Both sets of authors introduced TD's to provide a class of dependencies (sentences about relations) that include join dependencies [Ri] and embedded multivalued dependencies [Fa2] and that also has a complete axiomatization (no complete axiomatization is known for either join dependencies or embedded multivalued dependencies). TD's are examples of the "tuple-generating dependencies" of Beeri and Vardi [BV2]. Tuple-generating dependencies, along with "equality-generating dependencies" (which include functional dependencies [Co]) together comprise Fagin's [Fa3] class of embedded implicational dependencies (which is equivalent to Yannakakis and Papadimitriou's [YP] class of algebraic dependencies). This paper is a compendium of techniques, examples and counterexamples for TD's.

In § 2, we present definitions. In § 3, we demonstrate the existence of a strongest TD and a weakest nontrivial full TD. (*Note.* Unless stated otherwise, TD's are not assumed to be full.) We show that there is no weakest TD. In § 4, we show that there are only three distinct TD's on two attributes. In § 5, we demonstrate a useful correspondence between TD's and graphs and introduce the notion of an lp-homomorphism (label-preserving homomorphism). In § 6, we utilize this correspondence to help prove the existence of infinite chains of progressively weaker and progressively stronger full TD's. In § 7, we show that for TD's, implication is distinct

* Received by the editors May 21, 1981, and in revised form March 15, 1982.

† IBM Research Laboratory, San Jose, California 95193.

‡ State University of New York, Center at Stony Brook, Stony Brook, New York, 11794. The research of this author was supported in part by the National Science Foundation under grant IST-79-18264.

§ Stanford University, Stanford, California 94305. The research of this author was supported in part by the National Science Foundation under grant MCS-79-04528.

¶ Bell Laboratories, Murray Hill, New Jersey 07974.

from implication restricted to finite relations. In § 8, we show that the conjunction of a finite set of full TD's is equivalent to a single full TD. However, we show that the conjunction of a finite set of TD's is not necessarily equivalent to a single TD, and the disjunction of a finite set of full TD's is not necessarily equivalent to a single TD. In § 9, we show that there is a finite relation that obeys every strictly partial TD but no nontrivial full TD. In § 10, we demonstrate a finite set of TD's with no finite Armstrong relation [Fa3] (although we know [Fa3] that there is an infinite Armstrong relation). We also give a necessary and sufficient condition for the existence of finite Armstrong relations for sets of TD's.

2. Definitions. A relational database scheme consists of a universal set of attributes U and a set of "dependencies". The attributes in U are names for the components (columns) of relations in the database. The most common forms of dependencies are functional dependencies, or FD's [Co], and multivalued dependencies, or MVD's [Fa2]. We shall not discuss FD's in this paper.

In database theory, a tuple is formally regarded as a mapping from attributes to values, rather than as a list of component values, although the latter viewpoint is handy when the order of the attributes in the list is understood. We often use $t[Z]$, where t is a tuple and Z is a set of attributes, to stand for t restricted to domain Z, that is, the components of t for the attributes in Z. If A is an attribute, then we call $t[A]$ the *A entry* or *A value* of t.

Multivalued dependencies are denoted syntactically by $X \twoheadrightarrow Y$. The meaning of this dependency is that if relation R obeys the dependency, and if t_1 and t_2 are tuples of R with $t_1[X] = t_2[X]$, then there exists t_3 in R such that:

1. $t_3[X] = t_1[X] = t_2[X]$,
2. $t_3[Y] = t_1[Y]$ and
3. $t_3[U - XY] = t_2[U - XY]$.

Intuitively, the set of Y-values associated with each given X-value is independent of the values in all other attributes. By XY in 3 above, we mean $X \cup Y$.

Example. Consider the relation R, in Fig. 1.1, where $U = \{A, B, C, D\}$.

A	B	C	D
0	1	2	3
0	2	1	4
0	1	1	4
0	2	2	3
5	1	3	2

FIG. 1.1. *The relation R.*

The MVD $A \twoheadrightarrow B$ holds in R. For example, if t_1 and t_2 are the first two tuples in Fig. 1.1, then we may check that the tuple t_3, where $t_3[A] = t_1[A] = t_2[A] = 0$, $t_3[B] = t_1[B] = 1$, and $t_3[CD] = t_2[CD] = 14$, is present; it is row three. (By 14, we mean the tuple with first entry 1 and second entry 4; we shall sometimes find this type of abbreviation convenient.)

Let Σ be a set of dependencies, and let σ be a single dependency. When we say that Σ *logically implies* σ or that σ *is a logical consequence of* Σ, we mean that whenever every dependency in Σ holds for a relation R, then σ also holds for R. That is, there is no "counterexample relation" such that every dependency in Σ holds for R, but such that σ fails in R. We write $\Sigma \models \sigma$ to mean that Σ logically implies σ. For example, if A, B, and C are attributes, then $\{A \twoheadrightarrow B, B \twoheadrightarrow C\} \models A \twoheadrightarrow C$.

It appears that FD's and MVD's are almost sufficient to describe the "real world," and thus could be used for a database design theory. However, there is at least one, more general form of dependency that appears naturally, and this form causes severe difficulties when we try to infer dependencies. This type of dependency, called an embedded multivalued dependency (EMVD), was first studied by Fagin [Fa2] and Delobel [De]. For disjoint X, Y and Z, we say $X \twoheadrightarrow Y \mid Z$ holds if, when any "legal" relation over the set of attributes is projected onto the set of attributes XYZ (we project by restricting tuples to these attributes), then the MVD $X \twoheadrightarrow Y$ holds. (Note that $X \twoheadrightarrow Y$ holds in XYZ if and only if $X \twoheadrightarrow Z$ holds [Fa2]).

Another way of looking at the EMVD $X \twoheadrightarrow Y \mid Z$ is that if the relation R over attributes U obeys the dependency, then whenever we have two tuples t_1 and t_2 in R, and $t_1[X] = t_2[X]$, it follows that there is some t_3 in R, where

1. $t_3[X] = t_1[X] = t_2[X]$,
2. $t_3[Y] = t_1[Y]$ and
3. $t_3[Z] = t_2[Z]$.

Note that $t_3[U - XYZ]$ can be arbitrary; we can assert nothing about the values t_3 has in these components.

Unfortunately, when we try to make inferences about EMVD's we appear to run into a stone wall. It is not known whether the decision problem for EMVD's is decidable (the *decision problem* for EMVD's is the problem of deciding whether $\Sigma \models \sigma$, when Σ is a set of EMVD's and σ is a single EMVD). Neither is a complete axiomatization for EMVD's known. It is known [SW], [CFP] that there is no k-ary complete axiomatization for EMVD's for any fixed k, and, in particular, no finite complete axiomatization.

To tackle these problems for EMVD's, some more general types of dependencies have been studied recently, with the hope that the more general class would have a complete axiomatization or would provide insights on the EMVD decision problem. In particular, Sadri and Ullman [SU] and, independently, Beeri and Vardi [BV2] introduced *template dependencies*, or TD's, and provided a complete axiomatization. TD's include as special cases (a) MVD's, (b) EMVD's, (c) subset dependencies [SW], (d) mutual dependencies [Ni], (e) generalized mutual dependencies [MM] and (f) join dependencies [Ri]. The class of TD's was studied independently by Beeri and Vardi [BV2] and by Paradaens and Jannsens [PJ], and still more general classes were considered by Fagin [Fa3] and Yannakakis and Papadimitriou [YP]. Vardi [Va1] and, independently, Gurevich and Lewis [GL] have recently shown that the decision problem for TD's is undecidable.

A template dependency is an assertion about a relation R, that if we find tuples $r_1, \cdots, r_k$ in R with certain specific equalities among the entries of these tuples, then we can find in R a tuple r that has certain of its entries equal to certain of the entries in $r_1, \cdots, r_k$. Other entries of r may be arbitrary. Formally, we write a template dependency as $r_1, \cdots, r_k / r$, or as

$$\begin{array}{c} r_1 \\ \vdots \\ r_k \\ \hline r \end{array}$$

where the r_i's and r are strings of abstract symbols (sometimes called *variables*). The length of the r_i's and r equals the number of attributes in the universal set, and positions in these strings are assumed to correspond to attributes in a fixed order. No symbol

may appear in two distinct components among the r_i's and r. It is, of course, permissible that one symbol appear in the same component of several of the r_i's or r.

Let R be a relation and let T be a TD. Let h be a homomorphism that maps symbols in T into entries of R. By saying that h is a homomorphism, we mean that $h(a_1 \cdots a_n)$ is defined to be $h(a_1) \cdots h(a_n)$. We call h a *valuation*. Relation R is said to obey TD T if whenever there is a valuation h on the symbols appearing in the r_i's such that $h(r_i)$ is a tuple in R for all i, then we can extend h to those symbols that appear in r but do not appear among the r_i's, in such a way that $h(r)$ is also in R.

Example. Let $U = \{A, B, C, D\}$ and let R be the relation previously given in Fig. 1.1. Let T be the TD

$$\begin{array}{cccc} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_1 & c_2 & d_2 \\ a_1 & b_2 & c_2 & d_3 \\ \hline a_2 & b_3 & c_2 & d_1 . \end{array}$$

Define h by: $h(a_1) = h(a_2) = 0; h(b_1) = h(c_1) = 1; h(b_2) = h(c_2) = 2; h(d_2) = h(d_3) = 3$, and $h(d_1) = 4$. Then $h(a_1 b_1 c_1 d_1) = 0114$, $h(a_2 b_1 c_2 d_2) = 0123$, and $h(a_1 b_2 c_2 d_3) = 0223$, which are rows three, one, and four of Fig. 1.1. Thus, we must exhibit a value b for $h(b_3)$ such that $h(a_2 b_3 c_2 d_1)$ is in the relation of Fig. 1.1, if that relation is to obey the TD T. However, for no value of b is $0b24$ a row of Fig. 1.1, so we may conclude without further ado that R does not obey T. Of course, if a value of b had been found, we would then have to check all other possible valuations that mapped the first three rows of T into rows of Fig. 1.1.

When we say that a relation is *finite* (respectively, *infinite*), we mean that it has a finite (respectively, infinite) set of tuples. Database theory is most concerned with finite relations; however, sometimes it is convenient to consider infinite relations. If Σ is a set of dependencies, such as TD's, then by SAT (Σ), we mean the collection of relations (finite or infinite) that obey all of Σ. Note that $\Sigma \models \sigma$ if and only if SAT $(\Sigma) \subseteq$ SAT (σ). If we wish to consider only finite relations, then we can write $\text{SAT}_{\text{fin}}(\Sigma)$ to mean the collection of finite relations that obey Σ. Similarly, we can define $\Sigma \models_{\text{fin}} \sigma$ to mean that every finite relation that obeys Σ also obeys σ. As above, $\Sigma \models_{\text{fin}} \sigma$ if and only if $\text{SAT}_{\text{fin}}(\Sigma) \subseteq \text{SAT}_{\text{fin}}(\sigma)$. Note that if $\Sigma \models \sigma$, then $\Sigma \models_{\text{fin}} \sigma$. As we shall show in § 7, the converse fails for TD's.

When we speak of two dependencies σ and τ being *equivalent*, we mean that SAT $(\sigma) =$ SAT (τ), or equivalently, that $\sigma \models \tau$ and $\tau \models \sigma$. Similarly, we can define equivalent *sets* of dependencies. We shall sometimes speak of conjunctions or disjunctions of TD's. A relation obeys the conjunction (respectively, disjunction) of a set of TD's precisely if it obeys all (respectively, at least one) of them. Thus,

$$\text{SAT}(\wedge\{\sigma : \sigma \in S\}) = \bigcap\{\text{SAT}(\sigma) : \sigma \in S\},$$

$$\text{SAT}(\vee\{\sigma : \sigma \in S\}) = \bigcup\{\text{SAT}(\sigma) : \sigma \in S\}.$$

The following terminology will prove helpful. If $r_1, \cdots, r_k/r$ is a TD, then $r_1, \cdots, r_k$ are called the *hypothesis rows*, or *hypotheses*, and r is the *conclusion row*, or simply the *conclusion*. Each symbol that appears in the conclusion is said to be *distinguished*. A TD is said to be *full* if each of its distinguished symbols also appears in the hypotheses; otherwise, it is said to be *strictly partial*. If T is a TD, and if V is exactly the set of attributes for which the hypothesis rows of T contain distinguished variables, then we may call T a *V-partial* TD (we allow the possibility that $V = U$,

the set of all attributes). A TD is *trivial* if it always holds (in relations over the appropriate attributes).

Remark. A V-partial TD is trivial precisely if some hypothesis row of T contains distinguished variables for every one of its V entries. For if no hypothesis row of T contains distinguished variables for every one of its V entries, then the relation that consists of all of the hypothesis rows of T but not the conclusion is a relation not in SAT (T); hence, T is nontrivial.

Example. Let $U = \{A, B, C, D\}$. Then the MVD $A \twoheadrightarrow B$ is synonymous with the TD:

a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_1	c_2	d_2.

The EMVD $A \twoheadrightarrow B \mid C$ is written:

a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_1	c_2	d_3.

Note that this EMVD is a strictly partial TD. However, MVD's are full TD's.

3. Strongest and weakest TD's. An important tool in the study of dependencies is the chase process [ABU], [MMS], [SU]. When TD's alone are involved, could the chase go on forever in a nontrivial way? The question of the existence of infinite chases where "things keep happening" can be related to the existence of certain infinite sequences of TD's as follows. The set of rows in the tableau at any time during a chase may be taken to be the hypothesis rows of a TD whose conclusion row is the goal row for the chase. It is easy to show that as the chase proceeds, these TD's get progressively weaker. If the chase is successful, then we eventually arrive at a TD so weak that it is trivial.

If the chase is unsuccessful, then we might obtain an infinite sequence of TD's that, although some could be equivalent to the previous TD, would include an infinite subsequence of strictly weaker TD's. Or, we might necessarily reach a point where all successive TD's were equivalent but not trivial, and if we knew that we had reached that point, then we could deduce that the chase was unsuccessful.

These observations lead to the consideration of the structure of the space of TD's. Are there infinite sequences of strictly weaker TD's? Can we construct such a sequence by showing that for every nontrivial TD there is a weaker nontrivial TD? The answers to these (yes and no, respectively) and related questions are contained in later sections.

THEOREM 3.1. *For each set of attributes, there is a strongest* TD. *That is, there is a* TD T *such that* $T \models T'$ *for each* TD T' *over the same set of attributes as* T.

Proof. The TD that states a relation is a Cartesian product is the strongest TD. For example, the Cartesian product TD over three attributes is

a_1	b_1	b_2
b_3	a_2	b_4
b_5	b_6	a_3
a_1	a_2	a_3.

The Cartesian product TD is strongest because each relation that is a Cartesian product is easily seen to obey every TD (over the same attributes). □

Recall that a TD is said to be V-partial if V is the set of attributes for which the hypothesis rows of T contain distinguished variables.

COROLLARY 3.2. *There is a strongest V-partial* TD. *That is, there is a V-partial* TD T *such that $T \models T'$ for every V-partial* TD T' *over the same attributes.*

Proof. The V-partial TD that says of a relation that its projection onto V is a Cartesian product is the strongest V-partial TD. Thus, if U is ABC and V is AB, then this TD is

$$\begin{array}{ccc} a_1 & b_1 & b_2 \\ b_3 & a_2 & b_4 \\ \hline a_1 & a_2 & a_3 . \end{array}$$

□

THEOREM 3.3. *Assume that V contains at least two attributes. Then there is a weakest nontrivial V-partial* TD. *That is, there is a nontrivial V-partial* TD T *such that $T' \models T$ for every nontrivial V-partial* TD T' *over the same attributes. In particular (when $V = U$) there is a weakest nontrivial full* TD.

Note. The assumption that V contains at least two attributes is necessary, since it is easy to see that if V contains 0 or 1 attribute, then every V-partial TD is trivial.

Proof. Assume that the attributes in V are $A_1, \cdots, A_m$. Denote by W the attributes not in V. (Possibly, W is empty.) Assume that the attributes in W are $A_{m+1}, \cdots, A_n$. The variables of T that appear in the column A_i $(1 \leqq i \leqq m)$ of T are a_i and b_i. The only variable that appears in the hypothesis rows of A_j, for $j > m$, is c_j. The projection of the hypothesis of T into V contains all possible rows $e_1 \cdots e_m$, where e_i is either a_i or b_i, *except* that the row of all a's does not appear. The conclusion row contains all a's. For example, if $V = A_1A_2A_3$ and $W = A_4A_5$, then T is

$$\begin{array}{ccccc} a_1 & a_2 & b_3 & c_4 & c_5 \\ a_1 & b_2 & a_3 & c_4 & c_5 \\ a_1 & b_2 & b_3 & c_4 & c_5 \\ b_1 & a_2 & a_3 & c_4 & c_5 \\ b_1 & a_2 & b_3 & c_4 & c_5 \\ b_1 & b_2 & a_3 & c_4 & c_5 \\ b_1 & b_2 & b_3 & c_4 & c_5 \\ \hline a_1 & a_2 & a_3 & a_4 & a_5 . \end{array}$$

Clearly, T is nontrivial (see the remark near the end of § 2). We now show that if T' is a nontrivial, V-partial TD, then $\mathrm{SAT}(T') \subseteq \mathrm{SAT}(T)$, that is, that $T' \models T$. Let r be a relation (over set of attributes U) that is not in $\mathrm{SAT}(T)$; we shall show that r is not in $\mathrm{SAT}(T')$. Let g be a valuation that maps every hypothesis row of T to a tuple in r, but such that $g(a_1 \cdots a_m)$ does not appear in the projection $r[V]$ of r onto V. We know that g exists since r is not in $\mathrm{SAT}(T)$. We define a valuation h on T' as follows. We assume for convenience that T' and T have the same distinguished variables $a_1, \cdots, a_n$. For each distinguished variable a, let $h(a) = g(a)$. For each nondistinguished variable d in T', if d is in the A_i column, for some A_i in V, then let $h(d) = g(b_i)$; if d is in the A_j column for A_j in W, then let $h(d) = g(c_j)$.

Since T' is nontrivial, no hypothesis row of T' contains $a_1 \cdots a_m$ as its V entries. Let w' be an arbitrary hypothesis row of T' and let w be the row in T that has a's

in its V entries exactly where w' does. Since those entries are not all a's, we know that w exists. By definition of h, we know that $h(w') = g(w)$, and so $h(w')$ is a tuple in r. However, $h(a_1 \cdots a_m) = g(a_1 \cdots a_m)$ is not in $r[V]$, so r violates T', as was to be shown. □

We shall conclude this section by showing that there is no weakest nontrivial TD (including full and strictly partial TD's) if the number of attributes is at least 3. We first need a preliminary result.

THEOREM 3.4. *Let Σ be a set of V_1-partial* TD*'s and let σ be a nontrivial V_2-partial* TD. *If $\Sigma \models \sigma$, then $V_2 \subseteq V_1$.*

Proof. Assume that $\Sigma \models \sigma$ and that it is false that $V_2 \subseteq V_1$; we shall derive a contradiction. Let T_1 be the strongest V_1-partial TD constructed in the proof of Corollary 3.2, and let T_2 be the weakest nontrivial V_2-partial TD constructed in the proof of Theorem 3.3. Since (a) $T_1 \models \Sigma$ (that is, $T_1 \models \tau$ for every τ in Σ), (b) $\Sigma \models \sigma$, and (c) $\sigma \models T_2$, it follows by transitivity of logical implication that $T_1 \models T_2$. Let r be the relation consisting of the hypothesis rows of T_2. Then r violates T_2. We shall show that r obeys T_1, a contradiction.

Since it is false that $V_2 \subseteq V_1$ there is an attribute A in V_2 but not V_1. It is easy to verify that the projection $r[U - A]$ of r onto every attribute except A is the Cartesian product of the projection of r onto each attribute in $U - A$ (see Fig. 3.1). So, r obeys T_1, which was to be shown. □

THEOREM 3.5. *Assume that there are at least three attributes. Then there is no weakest nontrivial* TD. *That is, there is no nontrivial* TD *T such that $T' \models T$ for every nontrivial* TD *T' over the same attributes.*

Note. The assumption that there are at least three attributes is necessary, as we shall see in § 4. Also, observe that unlike Theorem 3.3, which might seem superficially to contradict Theorem 3.5, we are not fixing our attention on V-partial TD's for a given V, but rather considering the whole class of TD's at once.

Proof. Assume that there are at least three attributes, and that a weakest nontrivial TD T exists. Then T is V-partial for some V (possibly $V = U$). Now V is nonempty, since each V-partial TD with $V = \emptyset$ is trivial. So V contains an attribute A. Let $W = U - A$. Then W contains at least two attributes, since U contains at least three attributes. So there is a nontrivial W-partial TD T'. By definition of T, we know that $T' \models T$. This implication contradicts Theorem 3.4, since V is not a subset of W. □

4. TD's over two attributes. In this section, we prove the following result.

THEOREM 4.1. *There are only three distinct* TD*'s (up to equivalence) on two attributes.*

Proof. The three TD's over two attributes are the following:

						a_1	b_2
				a_1	b_2	b_1	b_2
a_1	a_2			b_1	a_2	b_1	a_2
a_1	a_2			a_1	a_2	a_1	a_2
T_1				T_2		T_3	

TD T_1 is the trivial TD, obeyed by every relation. TD T_2 says that the relation is a Cartesian product; it is the strongest TD. T_3 is the weakest nontrivial TD over two attributes. It is easy to check that none of T_1, T_2, and T_3 are equivalent. We must show that every TD over two attributes, say $T = t_1, t_2, \cdots, t_n / a_1 a_2$ is equivalent to one of these.

Case 1. None of $t_1, \cdots, t_n$ has a_1 in the first column, or none of $t_1, \cdots, t_n$ has a_2 in the second column, or some t_i is a_1a_2. It is easy to show that T is trivial. Thus, every strictly partial TD over two attributes is trivial.

Case 2. Case 1 does not hold, but there is no sequence of rows among $t_1, \cdots, t_n$ of the form

$$
(*) \qquad \begin{array}{ll} a_1 & b_1 \\ b_2 & b_1 \\ b_2 & b_3 \\ b_4 & b_3 \\ \cdots & \cdots \\ b_k & b_{k-1} \\ b_k & a_2 \end{array}
$$

for any $b_1, \cdots, b_k$, with $k \geqq 2$. Then, we can divide $t_1, \cdots, t_n$ into two groups. The first group contains those "reachable" from a_1, in the sense that they appear in some sequence $a_1b_1, b_2b_1, b_2b_3, b_4b_3, \cdots$, and the second contains those that are not. Tuples in the second category may be "reachable" from a_2 or they may be "reachable" from neither a_1 nor a_2.

We now show that T and T_2 are equivalent. We know that $T_2 \models T$, since the proof of Theorem 3.1 shows that T_2 implies every TD over two attributes. To show that $T \models T_2$, we need only show that when we chase [MMS] the hypothesis rows of T_2, using T, we get the conclusion row of T_2 [SU]. But this chase needs only one step. Map all tuples of T in the first group to a_1b_2 and all others to b_1a_2. This mapping cannot map one symbol of T to two distinct symbols of T_2, or the groups are not defined correctly. That is, we cannot have some tuple $t_i = cd$ mapped to a_1b_2, and then have some tuple $t_j = ed$ or cf mapped to b_1a_2, because ed and cf would be in group 1.

Case 3. A sequence (*) exists, with $k \geqq 2$, and a_1a_2 is not a hypothesis row. Then T is nontrivial, so by the proof of Theorem 3.3, we know that $T \models T_3$ (since T_3 is the weakest nontrivial full TD).

To show that $T_3 \models T$, we can chase the hypotheses of T with T_3 to infer successively the rows $a_1b_3, a_1b_5, \cdots, a_1b_{k-1}$ and then a_1a_2. □

5. The correspondence between TD's and graphs. For the upcoming examples, it is useful to give a graphical interpretation to TD's and relations. The graph for a TD or relation will have a node for each row or tuple, and edges labeled with attribute symbols, indicating in which components the rows or tuples agree. More precisely:

Definition. Given relation r on relation scheme $R = \{A_1, A_2, \cdots, A_n\}$, the *graph of* r, denoted G_r, is defined as follows. Let $\{t_1, t_2, \cdots, t_m\}$ be the tuples in r; the nodes in G_r will also be $t_1, t_2, \cdots, t_m$. For nodes t_1 and t_2, there is an undirected edge (t_1, t_2) with label A (possibly among others) in R exactly when $t_1(A) = t_2(A)$.

Example. Let r be

	A	B	C
t_1:	0	0	1
t_2:	0	1	0
t_3:	0	1	1
t_4:	1	0	0
t_5:	1	0	1
t_6:	1	1	0.

Then G_r is as in Fig. 5.1. There is always a self-loop from each node to itself, labeled by all the attributes, but we shall omit drawing such edges. We can also omit drawing some of the edges implied by transitivity of equality, to help reduce the clutter. Figure 5.2 represents the same relation as Fig. 5.1 when transitivity of equality is considered.

The graph (denoted G_T) for a template dependency T is defined similarly, except that there is a node denoted (*) that represents the conclusion row.

Example. Let $T =$

w_1:	a	b_1	c_1
w_2:	a_1	b	c_1
w_3:	a_1	b_1	c
	a	b	c .

Then G_T is as in Fig. 5.3.

We can characterize when a relation obeys a TD in terms of certain homomorphisms between their respective graphs.

DEFINITION. An lp-*homomorphism* (label-preserving homomorphism) between labeled, undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is a mapping $h : V_1 \to V_2$ such that if (v, w) is an edge of E_1 with label A (possibly among others) then $(h(v), h(w))$ is an edge of E_2 with label A.

Example. Let G_r and G_T be the graphs in the last two examples. Define the mappings h_1 and h_2 as follows:

$$\begin{aligned} h_1(*) &= t_5, & h_2(*) &= t_3, \\ h_1(w_1) &= t_5, & h_2(w_1) &= t_3, \\ h_1(w_2) &= t_1, & h_2(w_2) &= t_3, \\ h_1(w_3) &= t_1, & h_2(w_3) &= t_3. \end{aligned}$$

Then h_1 and h_2 are each lp-homomorphisms from G_T to G_r.

The mapping

$$\begin{aligned} h_3(*) &= t_1, \\ h_3(w_1) &= t_3, \\ h_3(w_2) &= t_5, \\ h_3(w_3) &= t_6 \end{aligned}$$

is not an lp-homomorphism from G_T to G_r, since $(h(*), h(w_3)) = (t_1, t_6)$ does not exist in G_r, and thus certainly does not have label C, as $(*, w_3)$ does.

We can now interpret the criterion for a relation r to obey a TD T in terms of their respective graphs.

THEOREM 5.1. *Relation r obeys T if and only if every* lp-*homomorphism from* $G_T - \{*\}$ *to* G_r *can be extended to an* lp-*homomorphism from all of* G_T *to* G_r.

The straightforward proof of Theorem 5.1 is left to the reader.

Example. Let T and r be the TD and relation used in previous examples. Some lp-homomorphisms from $G_T - \{*\}$ to G_r can be extended, such as h_1 and h_2 below:

$$\begin{aligned} h_1(w_1) &= t_5, & h_2(w_1) &= t_3, \\ h_1(w_2) &= t_1, & h_2(w_2) &= t_3, \\ h_1(w_3) &= t_1, & h_2(w_3) &= t_3. \end{aligned}$$

In fact, any lp-homomorphism that maps $G_T - \{*\}$ to a single node in G_r can be extended to G_T. We shall later use this fact to show that a particular TD T is obeyed

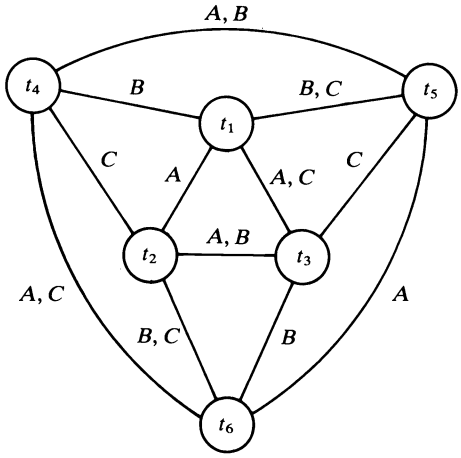


FIG. 5.1

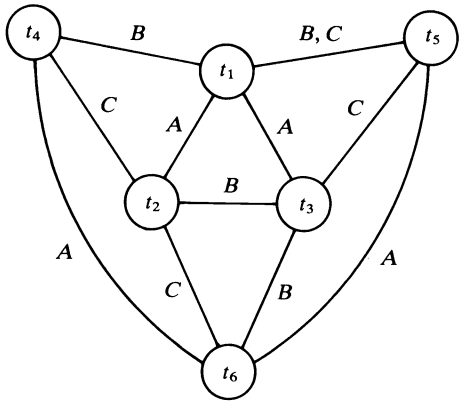


FIG. 5.2

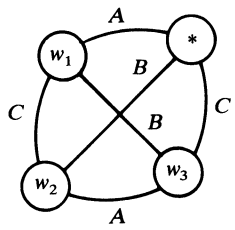


FIG. 5.3

by r, by showing that every lp-homomorphism from $G_T - \{*\}$ to r maps all of $G_T - \{*\}$ to a single node in r.

Relation r in our previous examples does not obey T, because there are lp-homomorphisms from $G_T - \{*\}$ to G_r that cannot be extended, such as

$$h_3(w_1) = t_3,$$
$$h_3(w_2) = t_5,$$
$$h_3(w_3) = t_6.$$

For, if $h_3(*) = t$, then t would have to agree with t_3 on A, with t_5 on B, and with t_6 on C. Then t would be (0, 0, 0), which is not in the relation r.

6. Chains of full TD's. We now use the correspondence between TD's and graphs to help prove the existence of infinite chains of progressively weaker and stronger full TD's.

LEMMA 6.1. *Let T' be a* TD *derived from* TD T *by the addition of hypothesis rows that use no distinguished symbols not already used in some hypothesis row. Then T is at least as strong as T'. That is, $T \models T'$.*

Proof. This result is easily verified by noting that any lp-homomorphism h' from $G_{T'} - \{*\}$ to a relation r can be restricted to an lp-homomorphism h from $G_T - \{*\}$ to r. Furthermore, if h cannot be extended to $G_{T'}$, then h cannot be extended to G_T. □

THEOREM 6.2 (progressively weaker chain). *There exists an infinite sequence of full* TD's $T_1, T_2, T_3, \cdots$ *such that* SAT $(T_i) \subset$ SAT (T_{i+1}) *for* $i \geqq 1$. *Thus, $T_i \models T_{i+1}$ for each i, and no T_i's are equivalent.*

Proof. Consider the *infinite* graph G (Fig. 6.1). Let T_i be the TD corresponding to the subgraph of G on nodes $*, 1, 2, \cdots, i+1$. By Lemma 6.1, SAT $(T_i) \subseteq$ SAT (T_{i+1}).

FIG. 6.1

To show proper containment, we need only exhibit a relation r in SAT (T_{i+1}) that does not obey T_i.

Relation r is simply the hypothesis rows of T_i considered as a relation. That is, r is any relation such that G_r is G restricted to nodes $1, 2, \cdots, i+1$. We see that r violates T_i, since the lp-homomorphism h from $G_{T_i} - \{*\}$ to G_r defined by $h(j) = j$, $1 \leqq j \leqq i+1$, cannot be extended to G_{T_i}.

We now show that r obeys T_{i+1}, that is, that each lp-homomorphism h from $G_{T_{i+1}} - \{*\}$ to G_r can always be extended to an lp-homomorphism from $G_{T_{i+1}}$ to G_r.

Case 1. For some nodes j and $j+1$ in $G_{T_{i+1}} - \{*\}$, we have $h(j) = h(j+1)$. Since in G, all odd nodes agree on A, and likewise all even nodes, if $h(j) = h(j+1)$ it follows that $h(p)$ and $h(q)$ agree on A for all p and q. In particular, $h(1)$, $h(2)$ and $h(3)$ agree on A, so we can extend h by letting $h(*) = h(2)$.

Case 2. No nodes j and $j+1$ are mapped to the same node in G_r by h. Let $h(1) = j$. There are 2 subcases, depending on whether j is even or odd.

Case 2a. j is odd. We shall show inductively that $h(k) = j + k - 1$ for $1 \leqq k \leqq i + 2$.

Assume $h(k-1) = j + k - 2$. Suppose k is odd. Since $k-1$ and k are connected by a C-labeled edge, $h(k-1)$ and $h(k)$ must be connected by a C-labeled edge. Since $j + k - 2$ is even, the only candidates for $h(k)$ are $j + k - 2$ and $j + k - 1$. The $j + k - 2$ choice is ruled out, since we are not in Case 1. Hence, $h(k) = j + k - 1$. A similar argument holds if k is even.

Now look at $h(i+2)$. By our inductive argument, $h(i+2) = j + i + 1 \geqq i + 2$, which is nonsense, since G_r contains only nodes $1, \cdots, i+1$. Thus, Case 2a cannot occur.

Case 2b. j is even. This case is very similar to Case 2a, except that we show inductively that $h(k) = j + 1 - k$, for $1 \leqq k \leqq i + 2$. Then $h(i+2) = j - i - 1 \leqq 0$, which is nonsense, since G_r contains only nodes $1, \cdots, i+1$. Thus, Case 2b cannot occur.

We have shown that Case 2 cannot occur. Thus, r obeys T_{i+1}, and the proof is complete. □

THEOREM 6.3 (progressively stronger chain). *There exists an infinite sequence of full* TD's $T_1, T_2, T_3, \cdots$ *such that* SAT $(T_{i+1}) \subset$ SAT (T_i). *That is,* $T_{i+1} \models T_i$ *for each* i, *and no two* T_i*'s are equivalent.*

Proof. Let T_i be the TD corresponding to the finite graph of Fig. 6.2, which we shall call G_i. G_i is just the graph for TD T_{2^i} in the last proof wrapped around with nodes 1 and $2^i + 1$ overlaid.

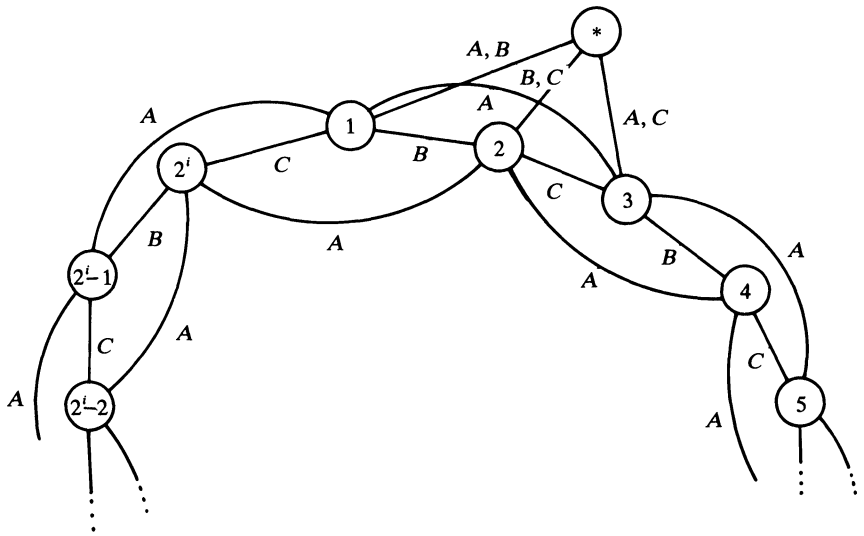


FIG. 6.2

The hard part of this proof is showing that SAT $(T_{i+1}) \subseteq$ SAT (T_i).

Let r be any relation in SAT (T_{i+1}); we shall show that r is in SAT (T_i). To prove this, let h be any lp-homomorphism from $G_i - \{*\}$ to G_r; we must show that h can be extended to an lp-homomorphism from G_i to G_r. We define an lp-homomorphism h' from $G_{i+1} - \{*\}$ to G_r in terms of h, by letting $h'(j)$ be $h(j)$, if $1 \leqq j \leqq 2^i$, and $h(j - 2^i)$ if $2^i < j \leqq 2^{i+1}$. Essentially, h' wraps G_{i+1} twice around the image of G_i in G_r under h. Since r is in SAT (T_{i+1}), we know that h' can be extended to G_{i+1}. The reader may check that h can be extended to G_i by letting $h(*) = h'(*)$.

The proof that SAT (T_{i+1}) is a *proper* subset of SAT (T_i) is by a counting argument similar to that used in the proof of Theorem 6.2. The relation r to use is one

corresponding to $G_{i+1}-\{*\}$. This relation is not in SAT (T_{i+1}). However, it is in SAT (T_i). For, any lp-homomorphism h from $G_i-\{*\}$ to G_r must map two nodes j and $j+1$ to the same node in G_r, which means the extension of h by $h(*)=h(2)$ will always work. □

7. Finite implication versus implication. In this section we show that finite implication (implication where we restrict our attention to finite relations) and unrestricted implication are distinct for TD's. Thus, the inference rules of Sadri and Ullman [SU] and of Beeri and Vardi [BV2] for TD's, which are complete for unrestricted implication, are incomplete when implication over finite relations only is considered. To state the result another way, let $\mathrm{SAT}_{\mathrm{fin}}(T)$ be the set of all *finite* relations that obey a TD T. We shall exhibit TD's $T_0, T_1, T_2, \cdots, T_k$ such that

$$\mathrm{SAT}_{\mathrm{fin}}(T_1, \cdots, T_k) \subseteq \mathrm{SAT}_{\mathrm{fin}}(T_0),$$

but

$$\mathrm{SAT}(T_1, \cdots, T_k) \not\subseteq \mathrm{SAT}(T_0).$$

Thus, $\{T_1, \cdots, T_k\} \models_{\mathrm{fin}} T_0$, but it is false that $\{T_1, \cdots, T_k\} \models T_0$. Further, we show that there can be no such example with $k=1$. That is, we show that if T_0 and T_1 are TD's, then $T_1 \models_{\mathrm{fin}} T_0$ if and only if $T_1 \models T_0$.

Apart from its inherent interest, we note another reason for studying the issue of whether finite and unrestricted implication are distinct. If finite implication and unrestricted implication were the same, then the decision problem would be decidable. That is, it would be decidable whether or not $\Sigma \models \sigma$, whenever Σ is a finite set of TD's and σ is a single TD. For, $\{(\Sigma, \sigma)$: Σ is finite and $\Sigma \models \sigma\}$ is r.e. (recursively enumerable), by Gödel's completeness theorem for first order logic [En] (or, in our special case, by the known [BV2], [SU] complete set of inference rules for TD's). Also, $\{(\Sigma, \sigma)$: Σ is finite and it is false that $\Sigma \models_{\mathrm{fin}} \sigma\}$ is r.e., since it is possible to systematically check for finite relations that obey Σ but not σ. Hence, if $\models$ and $\models_{\mathrm{fin}}$ were the same, then $\{(\Sigma, \sigma)$: Σ is finite and $\Sigma \models \sigma\}$ would be both r.e. and co-r.e., and hence decidable. As we have noted, Vardi [Va1] and, independently, Gurevich and Lewis [GL] have recently shown that the decision problem for TD's is undecidable.

THEOREM 7.1. *$\models$ and $\models_{\mathrm{fin}}$ are distinct. That is, implication of* TD*'s over the universe of all relations is distinct from implication of* TD*'s over the universe of finite relations.*

Proof. This proof draws its basic outline from a proof by Beeri and Vardi [BV3] of the same result for *untyped* TD's, that is, TD's in which a symbol may appear in more than one column. The construction used here is greatly more complicated than Beeri and Vardi's. We exhibit TD's T_0, T_1, T_2, T_3, T_4 for which there is an infinite relation that obeys $T_1, \cdots, T_4$ and violates T_0, but for which there is no such finite relation. The TD's $T_1, \cdots, T_4$ are given by graphs $G_1, \cdots, G_4$ in Fig. 7.1.

There is an underlying logic to these TD's. The intuition is that if we look at a relation r, we interpret the subgraph of G_r in Fig. 7.2 as representing a directed edge from t_1 to t_3. The relation r can then be interpreted as a directed graph D_r on some subset of its tuples. TD's T_1 and T_2 together say that if D_r has an edge $u \to v$ then for some w it has edge $v \to w$. That is, no node v is a sink. TD T_3 says roughly that D_r is transitively closed. What it actually tells us is that if we have the linked configuration of Fig. 7.3, then for some tuple t' we have Fig. 7.4, where t' is the tuple $*$ of G_3. As we shall see, TD T_4 applies nontrivially when D_r has an edge u such that $u \to u$.

The last TD, T_0, corresponds to graph G_0 in Fig. 7.5.

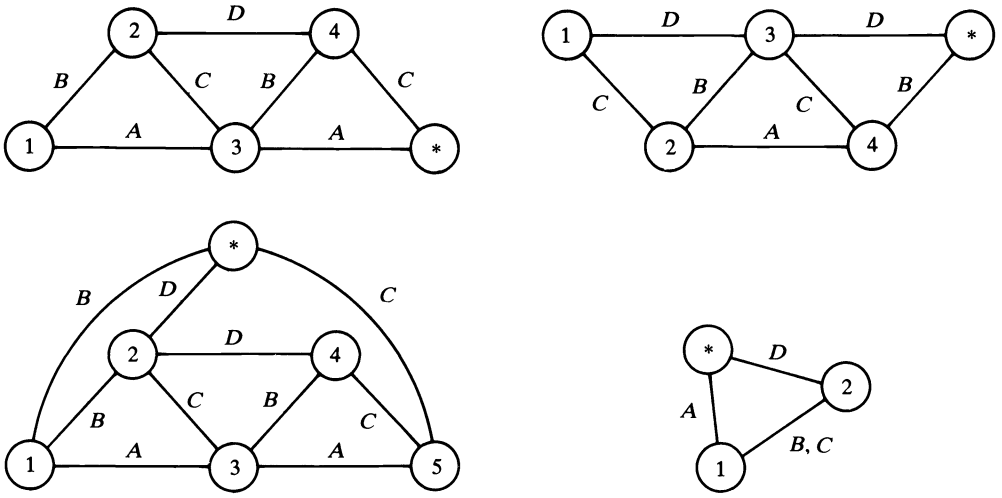


FIG. 7.1

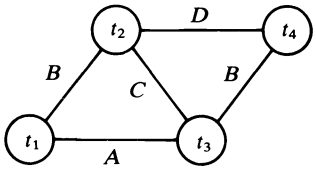


FIG. 7.2

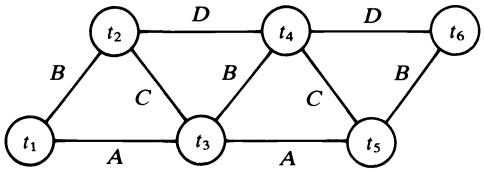


FIG. 7.3

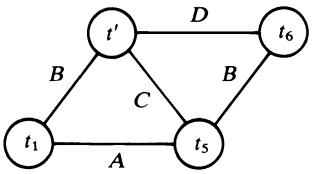


FIG. 7.4

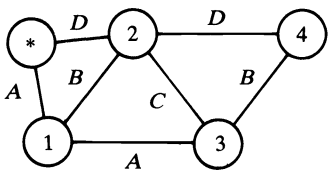


FIG. 7.5

The property of directed graphs we shall exploit is that any finite directed graph D that has no sinks and that is transitively closed has at least one loop edge. This statement is not true for infinite graphs; consider the graph on the natural numbers, where $i \to j$ is an edge if and only if $i < j$.

We now present an infinite relation r_I, and show that r_I obeys T_1, T_2, T_3 and T_4, but violates T_0. Thus, it is false that $\{T_1, T_2, T_3, T_4\} \models T_0$.

Let $r_I = \{(i, i, j, 0): 1 \leqq i < j\} \cup \{(0, i, i, i): 1 \leqq i\}$. We shall refer to tuples of r_I of the form $(i, i, j, 0)$ with $1 \leqq i < j$ as tuples of the first type and tuples $(0, i, i, i)$ with $1 \leqq i$ as tuples of the second type.

1. *r_I obeys T_1.* We shall show that if we chase r_I with T_1, then no new tuples appear. Consider the first time that a new tuple could appear. The only AC combinations not already present in r_I that could be forced by chasing with T_1 are those in which the A entry is i (we write this informally as $A = i$), $C = j$, and $i \geqq j \geqq 1$. To obtain such an AC combination, an application of T_1 must have $t_4 = (\cdot, b, j, \cdot)$ and $t_3 = (i, b, \cdot, \cdot)$. (By this we mean that t_3 and t_4 have the same B entry b, and the $\cdot$'s represent entries we don't care about now.) Since $i \geqq 1$, we know that t_3 is a tuple of the first type, so $b = i$. So t_4 is $(\cdot, i, j, \cdot)$ with $i \geqq j$. Thus, t_4 is a tuple of the second type, so $t_4 = (0, i, i, i)$. Since t_2 agrees with t_4 in D, we know that $t_2 = t_4$. Hence, t_4 agrees with t_3 in C (since t_2 agrees with t_3 in C). So the A and C entries of t_3 are both i, and hence equal. But in no tuple of r_I do the A and C entries agree. This is a contradiction, so chasing r_I with T_1 can produce no new AC entries. Hence, r_I obeys T_1, since T_1 is an AC-partial TD.

2. *r_I obeys T_2.* The only BD combinations that can be generated by chasing r_I with T_2 and that are missing have $B = i$, $D = j$, $i \neq j$ and $j \neq 0$. So $t_3 = (\cdot, \cdot, c, j)$, and $t_4 = (\cdot, i, c, \cdot)$. Since $j \neq 0$, we know that $t_3 = (0, j, j, j)$. Since $j = c \neq i$, we know $t_4 = (i, i, j, 0)$. Now t_2 agrees with t_4 in A, so $t_2 = (i, i, \cdot, 0)$. Thus, t_2 does not agree with t_3 in B, a contradiction.

3. *r_I obeys T_3.* Since t_2 and t_4 agree on D, they are both tuples of the first type or they are both tuples of the second type. If they are both tuples of the second type then they are equal, since they agree on D. In this case, either can serve as $*$ ($*$ must have C from t_4, and BD from t_2). So we can assume that t_2 and t_4 are both of the first type. The only way that no tuple of r_I can serve as $*$ is if the B entry of t_1 (and t_2), say i, is greater than or equal to the C entry of t_5 (and t_4), say j. So assume $i \geqq j$. Let $t_3 = (a, i', j', \cdot)$. Since $t_2 = (i, i, j', 0)$, we know that $i < j'$. Similarly, $t_4 = (i', i', j, 0)$ and $i' < j$. There are now two cases. *Case* 1. $a \neq 0$. Then, t_1, t_3 and t_5 are all of the first type. Since t_3 is of the first type, $a = i'$. Now, the B entry of t_1 is i, so the A entry of t_1 is i. Thus, $a = i$, so $i = i'$. Since $i' < j$, it follows that $i < j$, a contradiction. *Case* 2. $a = 0$. Then $i' = j'$, so $i < j' = i' < j$, a contradiction.

4. *r_I obeys T_4.* Since t_1 and t_2 agree on B and C, it follows easily that $t_1 = t_2$. Thus, $*$ can be taken to be t_1.

5. *r_I violates T_0.* Let $t_1 = (0, 1, 1, 1)$, $t_2 = (1, 1, 2, 0)$, $t_3 = (0, 2, 2, 2)$ and $t_4 = (2, 2, 3, 0)$. Then $*$ must be $(0, \cdot, \cdot, 0)$, and r_I contains no such tuple.

We now show that no finite relation r_F in SAT (T_1, T_2, T_3, T_4) violates T_0. Suppose r_F violates T_0. Then, G_{r_F} contains the configuration in Fig. 7.6 (ignoring X and its edges), where no tuple in r_F can serve as the node marked X (and so $t_1 \neq t_2$), even if we allow other edges connecting X to $t_1, \cdots, t_4$. By TD's T_1 and T_2, we know that r_F must also contain tuples t_5 and t_6 such that G_{r_F} contains the subgraph in Fig. 7.7. We do not require that the tuples be distinct. Further applications of T_1 and T_2 give the subgraph in Fig. 7.8, which we shall abbreviate as in Fig. 7.9. We remarked before that the tuples need not be distinct. Actually, if we extend this chain far enough they

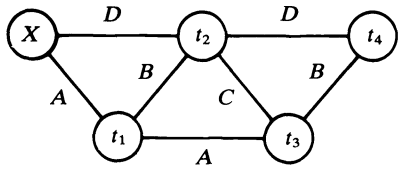


FIG. 7.6

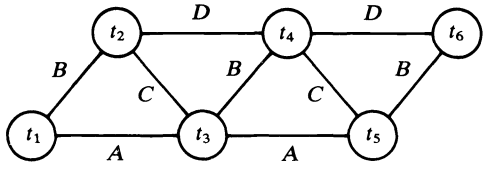


FIG. 7.7

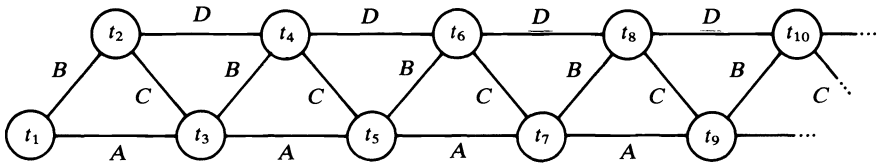


FIG. 7.8

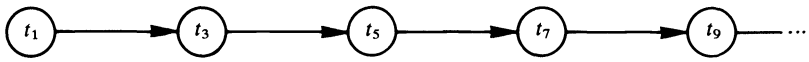


FIG. 7.9

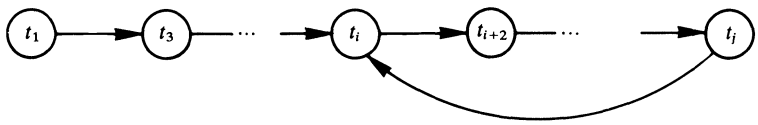


FIG. 7.10

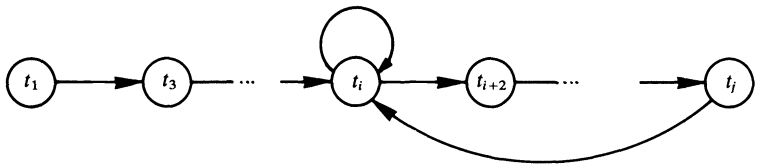


FIG. 7.11

cannot be distinct, since r_F is finite. The chain must eventually loop back on itself (Fig. 7.10). By repeated application of the "transitivity" TD, T_3, we eventually get an edge from t_i to itself (Fig. 7.11). The self-loop from t_i to itself means the same as the configuration shown in Fig. 7.12, where t_i appears twice, and where the exact identities of t' and t'' do not matter (except that $t'[D] = t_{i+1}[D] = t_2[D]$). As we see, t' agrees with t_i on both B and C. T_4 now applies to give us a tuple t where Fig. 7.13 holds. But $t_i[A] = t_1[A]$ and $t'[D] = t_2[D]$, so Fig. 7.14 holds. Hence, t serves as the

slot marked by X in the original figure, a contradiction. Relation r_F cannot violate T_0, concluding the proof. □

Although, as we just proved, there are TD's $T_0, T_1, \cdots, T_k$ such that $\{T_1, \cdots, T_k\} \models_{\text{fin}} T_0$ but for which $\{T_1, \cdots, T_k\} \models T_0$ fails, we now show that this is impossible if $k = 1$.

THEOREM 7.2. *Let T_0 and T_1 be* TD*'s. Then $T_1 \models_{\text{fin}} T_0$ if and only if $T_1 \models T_0$.*

Proof. It is immediate that if $T_1 \models T_0$, then $T_1 \models_{\text{fin}} T_0$. So assume that $T_1 \models_{\text{fin}} T_0$. We must show that $T_1 \models T_0$. Assume that T_1 is V_1-partial, and that T_0 is V_0-partial. Now Theorem 3.4 holds when "$\models$" is replaced by "$\models_{\text{fin}}$", by the same proof. So, since $T_1 \models_{\text{fin}} T_0$, it follows that $V_0 \subseteq V_1$. So, when we use T_1 to chase the hypothesis rows of T_0, it is easy to see that we never need to add a new row whose projection onto V_1 is already present. No new variables are added in the V_1 columns during the chase, so the chase terminates after a finite number of steps. Thus, as in the theory of the chase for full TD's [MMS], if there is a "counterexample" relation that obeys T_1 but not T_0, then there is a finite such counterexample. The result follows. □

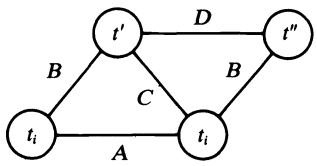


FIG. 7.12

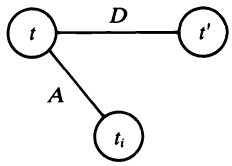


FIG. 7.13

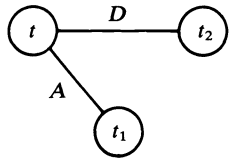


FIG. 7.14

We note that Theorem 7.2 was proven by Sadri [Sa] in the case where T_0 and T_1 are EMVD's. Also, Beeri and Vardi [BV1] showed if Σ is a set of V-partial TD's and σ a TD, then $\Sigma \models \sigma$ if and only if $\Sigma \models_{\text{fin}} \sigma$. This implies Theorem 7.2.

8. Closure of full TD's under conjunction. In this section, we show that full TD's are closed under finite conjunction. That is, we show that if Σ is a finite set of full TD's, then there is a single full TD T that is equivalent to Σ (in other words, SAT (T) = SAT (Σ)). The same result was obtained independently by Beeri and Vardi [BV2]. However, we show that the conjunction of a finite set of TD's (not necessarily full) is not necessarily equivalent to a single TD, and the disjunction of a finite set of full TD's is not necessarily equivalent to a single TD.

Since every multivalued dependency is equivalent to a full TD, it follows in particular that (the conjunction of) every set of multivalued dependencies is equivalent to a TD. However, sets of multivalued dependencies that are not only equivalent to a TD, but even to a join dependency (which are special cases of TD's), are quite special [BFMMUY], [BFMY], [FMU].

Our main tool is the direct product construction of Fagin [Fa3]. Let r and r' be relations, each with attributes $U = A_1 \cdots A_n$. The direct product $r \otimes r'$ has the same set U of attributes. The possible entries in the A_i column of $r \otimes r'$ are elements $\langle a, a' \rangle$, where a is an entry in the A_i column of r, and a' is an entry in the A_i column of r'. A tuple $(\langle a_1, a_1' \rangle, \cdots, \langle a_n, a_n' \rangle)$ is a tuple of the direct product if and only if $(a_1, \cdots, a_n)$ is a tuple of r and $(a_1', \cdots, a_n')$ is a tuple of r'. Fagin [Fa3] shows that if T is a TD (or even more generally, an embedded implicational dependency), and if r and r' are nonempty relations, then T holds for $r \otimes r'$ if and only if T holds for each of r and r'. This property is called *faithfulness* of T.

THEOREM 8.1. *Full* TD*'s are closed under finite conjunction.*

Proof. It is sufficient to prove that if T_1 and T_2 are full TD's, then there is a TD T that is equivalent to their conjunction; the result then follows by an easy induction. We use the direct product construction on hypothesis rows of the TD's T_1 and T_2. That is, let T_1 be

$$\begin{array}{cccc} c_{11} & c_{12} & \cdots & c_{1n} \\ & & \vdots & \\ c_{r1} & c_{r2} & \cdots & c_{rn} \\ \hline a_1 & a_2 & \cdots & a_n \end{array}$$

and let T_2 be

$$\begin{array}{cccc} d_{11} & d_{12} & \cdots & d_{1n} \\ & & \vdots & \\ d_{s1} & d_{s2} & \cdots & d_{sn} \\ \hline a_1 & a_2 & \cdots & a_n . \end{array}$$

We now define a new TD T, that we shall prove is equivalent to $T_1 \wedge T_2$. The hypothesis rows of T are the direct product of the hypothesis rows of T_1 (treated as a relation) and the hypothesis rows of T_2 (treated as a relation). Thus, let the symbols for the kth column of T be the product symbols $\langle c_{ik}, d_{jk} \rangle$ for $1 \leqq i \leqq r$ and $1 \leqq j \leqq s$, with $\langle a_k, a_k \rangle$ being the distinguished symbol for column k. The rs hypothesis rows of T are all of the rows of the form

$$\langle c_{i1}, d_{j1} \rangle \langle c_{i2}, d_{j2} \rangle \cdots \langle c_{in}, d_{jn} \rangle$$

for all i and j. The conclusion row of T is $\langle a_1, a_1 \rangle \langle a_2, a_2 \rangle \cdots \langle a_n, a_n \rangle$, of course.

$T \models T_1$, as we can show in one step of a chase by using the mapping that sends $\langle c_{ij}, d \rangle$ to c_{ij} for each d. Similarly, $T \models T_2$.

We shall show, by chasing the hypothesis rows of T, that $\{T_1, T_2\} \models T$. First, for each (fixed) j, apply T_1 to the r hypothesis rows of the form $\langle c_{i1}, d_{j1} \rangle \cdots \langle c_{in}, d_{jn} \rangle$ for $1 \leqq i \leqq r$ to infer the rows of the form $\langle a_1, d_{j1} \rangle \cdots \langle a_n, d_{jn} \rangle$ for $1 \leqq j \leqq s$. Then apply T_2 to these rows to infer $\langle a_1, a_1 \rangle \cdots \langle a_n, a_n \rangle$. □

Although the finite conjunction of full TD's is equivalent to a single TD, we now show that the finite conjunction of TD's (not necessarily full) is not necessarily equivalent to a single TD.

THEOREM 8.2. *There is a pair of* TD*'s whose conjunction is not equivalent to a single* TD.

Proof. It is sufficient to show that there is a finite set $T_1, \cdots, T_k$ of TD's such that $T_1 \wedge \cdots \wedge T_k$ is not equivalent to a single TD. For, if the conjunction of a pair of TD's were always equivalent to a single TD, then by induction, the conjunction of a finite set of TD's would be equivalent to a single TD.

Let $T_0, T_1, \cdots, T_4$ be the TD's of § 7 (for which $\{T_1, \cdots, T_4\} \models_{\text{fin}} T_0$ but for which $\{T_1, \cdots, T_4\} \models T_0$ fails). If $T_1 \wedge \cdots \wedge T_4$ were equivalent to a single TD T, then $T \models_{\text{fin}} T_0$, since $\{T_1, \cdots, T_4\} \models_{\text{fin}} T_0$. By Theorem 7.2, it follows that $T \models T_0$. So, $\{T_1, \cdots, T_4\} \models T_0$. This is a contradiction. □

Vardi [Va2] has posed the interesting question as to whether the conjunction of a pair of V-partial TD's (for the same V) is necessarily equivalent to a TD.

We now prove a result that implies (by Corollary 8.4 below) that TD's are not closed under finite disjunction.

THEOREM 8.3. *Let T_1 and T_2 be incomparable* TD*'s (that is, neither $T_1 \models T_2$ nor $T_2 \models T_1$). Then the disjunction $T_1 \vee T_2$ is not equivalent to a single* TD.

Proof. Let r_1 be a relation that obeys T_1 but not T_2, and let r_2 be a relation that obeys T_2 but not T_1. Let r be the direct product $r_1 \otimes r_2$. Then by faithfulness of T_1, we know that r does not obey T_1, since r_2 does not obey T_1. Similarly, r does not obey T_2, and so r does not obey $T_1 \vee T_2$. However, each of r_1 and r_2 obeys $T_1 \vee T_2$, since r_1 obeys T_1 and r_2 obeys T_2. If $T_1 \vee T_2$ were equivalent to a TD T, then the faithfulness of T would be violated. □

COROLLARY 8.4. *There are full* TD*'s T_1 and T_2 such that $T_1 \vee T_2$ is not equivalent to a single* TD.

Proof. Let T_1 and T_2 be incomparable full TD's. For example, over three attributes ABC, let T_1 be the MVD $A \twoheadrightarrow B$ and let T_2 be the MVD $B \twoheadrightarrow A$. By Theorem 8.3, it follows that $T_1 \vee T_2$ is not equivalent to a TD. □

We note that Ginsburg and Zaiddan [GZ] have considered questions similar to those discussed in this section, but for FD's instead of TD's, by studying intersections and unions of "functional dependency databases." Classes SAT (Σ), where Σ is a set of FD's, are called *functional dependency classes* by Fagin [Fa3]. *Functional dependency databases* differ from functional dependency classes by explicitly defining the domains for each attribute.

9. A set of strictly partial TD's cannot imply a full TD. In this section, we prove the following result.

THEOREM 9.1. *There is a finite relation that obeys every strictly partial* TD *but no nontrivial full* TD. *In particular, if Σ is a set of strictly partial* TD*'s and σ is a nontrivial full* TD, *then it is false that $\Sigma \models \sigma$ (or even that $\Sigma \models_{\text{fin}} \sigma$).*

We give two proofs of Theorem 9.1, since both proofs are amusing and both give additional information.

Proof 1. This proof is in the spirit of Sadri's [Sa] proof that there is a finite relation that obeys every EMVD that is not a MVD but violates every MVD. Let R be the relation that contains every tuple consisting only of 0's and 1's *except* the tuple of all 0's. For example, if there are three attributes, then R is

0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1.

This relation obeys every strictly partial TD, since the projection onto each proper subset V of the attributes U is the Cartesian product of the projection onto each attribute of V. However, R clearly violates the weakest nontrivial full TD T constructed in the proof of Theorem 3.3. Hence, R violates every nontrivial full TD (if R obeyed a nontrivial full TD T', then R would obey T, since $T' \models T$ by Theorem 3.3). □

Proof 2. Let $\mathscr{A}_n$ be the set of all relations (with attributes U) such that every entry of the relation is a member of $\{1, \cdots, n\}$. Thus, $\mathscr{A}_n$ contains 2^{n^u} members, where u is the number of attributes (that is, the size) of U. If P is a property of relations, then we say that "almost all relations have property P" (or "a random relation has property P") if the fraction of members of $\mathscr{A}_n$ with property P converges to 1 as $n \to \infty$. Fagin [Fa1] showed that if P is a first-order property of relations, then either almost all relations have property P or almost all relations fail to have property P. Using his techniques, it is easy to show that if σ is a strictly partial TD, then almost all relations obey σ, while if σ is a nontrivial full TD, then almost all relations violate σ.

Let T_V be the strongest V-partial TD (which exists by Corollary 3.2), and let $\Sigma = \{T_V : V \text{ is a proper subset of } U\}$. Then Σ is a finite set of TD's, since U contains only a finite number of subsets. By the above remarks, for each TD T_V in Σ, almost all relations obey T_V (since T_V is strictly partial). Since Σ is finite, it follows from elementary probability theory that almost all relations simultaneously obey every member of Σ. Furthermore, if σ is the weakest nontrivial full TD, whose existence is guaranteed by Theorem 3.3 (with $V = U$), then it follows by our earlier remarks that almost all relations violate σ (since σ is full). Thus, almost all relations obey Σ and violate σ. If a relation R obeys Σ, then it obeys every strictly partial TD, since if T is a V-partial TD, then T is implied by T_V, which is in Σ, if V is a proper subset of U. Further, if a relation R violates the weakest nontrivial full TD σ, then it violates every nontrivial full TD T (since $T \models \sigma$). Thus, almost all relations simultaneously obey every strictly partial TD and violate every nontrivial full TD. This is even stronger than the statement of Theorem 9.1. □

10. Finite Armstrong relations. Let Σ be a set of TD's. Let Σ^*_{fin} be $\{\sigma : \Sigma \models_{\text{fin}} \sigma\}$. Thus, Σ^*_{fin} is the set of all TD's that hold in every finite relation obeying Σ. A *finite Armstrong relation* [Fa3] for Σ is defined to be a finite relation that obeys Σ^*_{fin} but no other TD's. The following facts are easy consequences of results by Fagin [Fa3].

Fact 1. There is an Armstrong relation (not necessarily finite) for Σ. This fact can be interpreted in two distinct ways, both of which are correct. One meaning is that there is a relation (not necessarily finite) that obeys every TD in $\Sigma^* = \{\sigma : \Sigma \models \sigma\}$, but no other TD's. The second meaning is that there is a relation (not necessarily finite) that obeys every TD in Σ^*_{fin} but no other TD's; this is true because $(\Sigma^*_{\text{fin}})^* = \Sigma^*_{\text{fin}}$.

Fact 2. Let $\mathscr{S}$ be a fixed finite set of TD's (such as the set of all EMVD's over some fixed set of attributes). Then, there is a finite relation that obeys every TD in Σ^*_{fin} but violates every TD in $\mathscr{S}$ that is not in Σ^*_{fin}.

In this section, we shall show (Theorem 10.1 below) that the second sentence of Fact 2 is not necessarily true if $\mathscr{S}$ is the set of all TD's (this set is infinite by § 6, if there are at least three attributes). Also, we note that Fagin shows [Fa3] that the second sentence of Fact 2 is false if "TD" is replaced by "EID" (embedded implicational dependency) and if $\mathscr{S}$ is the set of all EID's.

By Theorem 10.1 below, there is a finite set Σ of TD's that have no finite Armstrong relation (although Σ has an infinite Armstrong relation, by Fact 1 above).

However, there are certainly some sets Σ of TD's that do have a finite Armstrong relation; for example, if Σ is the set of all TD's, then Σ has a finite Armstrong relation, namely, any one-tuple relation. Also, we show at the end of this section that if Σ is the empty set, then Σ has a finite Armstrong relation. In Theorem 10.2 below, we give several characterizations of those sets Σ of TD's that have a finite Armstrong relation.

THEOREM 10.1. *There is a finite set Σ of* TD*'s such that Σ has no finite Armstrong relation (with respect to* TD*'s). That is, there is no finite relation that obeys Σ^*_{fin} and no other* TD*'s.*

Proof. Let Σ be $\{T_3, T_4\}$, where T_3 and T_4 are as in the proof of Theorem 7.1. We shall show that there is no finite Armstrong relation for Σ. Let T^k be the TD that looks like T_0 of Theorem 7.1, except that the quadrangle is repeated k times; i.e., T^k is the TD shown in Fig. 10.1.

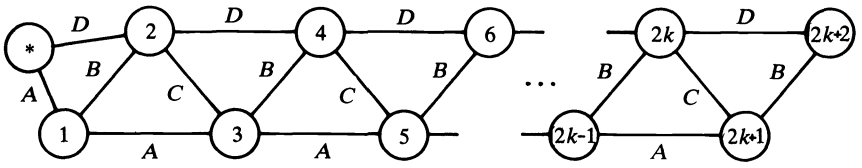


FIG. 10.1

We shall show that 1) for every k, it is false that $\Sigma \models_{\text{fin}} T^k$, and 2) every finite relation obeying Σ also obeys some T^k. It follows easily from 1) and 2) that there is no finite Armstrong relation for Σ.

1) *holds.* Let r^k be the relation $\{(i, i, j, 0): 1 \leqq i < j \leqq k+2\} \cup \{(0, i, i, i): 1 \leqq i \leqq k+1\}$. Then r^k is roughly the truncation of the relation r_I in the proof of Theorem 7.1 to the first $k+2$ positive integers. Now r^k obeys Σ. The proof is exactly the same as the proof in Theorem 7.1 that r_I satisfies T_3 and T_4. However, r^k violates T^k. For, let $t_{2i-1} = (0, i, i, i)$, and $t_{2i} = (i, i, i+1, 0)$ for $i = 1, \cdots, k+1$. Then the role of $*$ in the TD T^k must be filled by $(0, \cdot, \cdot, 0)$, although r^k contains no such tuple. We have shown that r^k obeys Σ but not T^k. This proves 1).

2) *holds.* Let r be a finite relation that obeys Σ and that has exactly k tuples. Consider the TD T^k. Every lp-homomorphism from the graph $G_{T^k} - \{*\}$ to G_r must map two distinct nodes $2i+1$, $2j+1$ to the same node (since there are $k+1$ odd-numbered nodes in $G_{T^k} - \{*\}$ and only k nodes in G_r). Then, as in the proof of Theorem 7.1, we can show that there is a tuple of r that can play the role of $*$. Therefore, r obeys T^k. This completes the proof of 2), and hence the proof of the theorem. □

An alternative proof of Theorem 10.1 can be obtained by using Vardi's result [Va1] that there is a single finite set Σ of TD's such that the set of all TD's σ for which $\Sigma \models_{\text{fin}} \sigma$ is not recursive. This result implies that there is no finite Armstrong relation for Σ, since we could test whether or not $\Sigma \models_{\text{fin}} \sigma$ by simply checking whether or not the finite Armstrong relation obeys σ.

THEOREM 10.2. *Let Σ be a set of* TD*'s. The following are equivalent*:

(a) *There is a finite relation that obeys Σ^*_{fin} and no other* TD*'s* ("*Σ has a finite Armstrong relation*").

(b) *There is a finite set $\mathcal{T}$ of* TD*'s, disjoint from Σ^*_{fin}, such that for each* TD *T not in Σ^*_{fin} there is a* TD *T' in $\mathcal{T}$ where $T \models T'$.*

(c) *There is a finite set $\mathscr{T}$ of* TD's, *disjoint from* Σ^*_{fin}, *such that* $T \models \vee\{T' : T' \in \mathscr{T}\}$ *for each* TD T *not in* Σ^*_{fin}.

(d) *There is a finite set $\mathscr{T}$ of* TD's, *disjoint from* Σ^*_{fin}, *such that* $\vee\{T : T \notin \Sigma^*_{\text{fin}}\}$ *is equivalent to* $\vee\{T' : T' \in \mathscr{T}\}$.

Note that $\{T' : T' \in \mathscr{T}\}$ in (d) is a finite subset of $\{T : T \notin \Sigma^*_{\text{fin}}\}$ in (d). So, (d) is a kind of compactness result, that says that a certain set has a finite subcover (that is, it says that a finite number of disjuncts of $\vee\{T : T \notin \Sigma^*_{\text{fin}}\}$ "covers" all of it).

Proof. (a)$\Rightarrow$(b). Let R be a finite relation that obeys Σ^*_{fin} and no other TD's. We now define a finite set $\mathscr{T}$ of TD's, each of which R violates. For each set P of rows of R and for each set V ($V \subseteq U$) of attributes, let $\mathscr{T}$ contain every V-partial TD with P as its hypothesis rows that is false about R. It is easy to see that $\mathscr{T}$ is a finite set of TD's. The set $\mathscr{T}$ is disjoint from Σ^*_{fin}, since R obeys Σ^*_{fin} and violates every member of $\mathscr{T}$. Now let T be a TD not in Σ^*_{fin}. We must show that there is a TD T' in $\mathscr{T}$ where $T \models T'$. Assume that T is V-partial. Since T is not in Σ^*_{fin}, we know that R violates T. So, there is a valuation h that maps the hypothesis rows of T onto rows of R such that there is no way to extend h to get the conclusion row of T mapped onto a row of R.

Let T' be the V-partial member of $\mathscr{T}$ whose hypothesis rows are the images under h of the hypothesis rows of T, and such that for each attribute A in V, the A entry of the conclusion row of T' is the image under h of the A entry of the conclusion row of T. We now show that $T \models T'$. For, assume that a relation S obeys T; we must show that S obeys T'. To show this, assume that the hypothesis rows of T' can be mapped by a valuation h' onto rows of S. We must show that h' can be extended to a mapping from the conclusion row of T' onto a row of S. Now $h \circ h'$ is a valuation from the hypothesis rows of T onto these same rows of S. Then $h \circ h'$ is already defined on the V entries of the conclusion row of T, and (since T holds for S) can be extended to map all of the conclusion row of T onto a row of S. This gives us an extension of h' to map all of the conclusion row of T' onto the same row of S, by mapping the A entry of the conclusion row of T' (for each A not in V) onto the same entry of S as the extension of $h \circ h'$ maps that entry. This was to be shown. So, $T \models T'$, as desired.

(b)$\Rightarrow$(c). Let the set $\mathscr{T}$ of (c) equal the set $\mathscr{T}$ of (b). Take T not in Σ^*_{fin}. By (b), there is some T' in $\mathscr{T}$ such that $T \models T'$. Hence, $T \models \vee\{T' : T' \in \mathscr{T}\}$.

(c)$\Rightarrow$(d). Let the set $\mathscr{T}$ of (c) equal the set $\mathscr{T}$ of (d). It is obvious that $\vee\{T' : T' \in \mathscr{T}\} \models \vee\{T : T \notin \Sigma^*_{\text{fin}}\}$, since $\{T' : T' \in \mathscr{T}\} \subseteq \{T : T \notin \Sigma^*_{\text{fin}}\}$. Conversely, we must show that $\vee\{T : T \notin \Sigma^*_{\text{fin}}\} \models \vee\{T' : T' \in \mathscr{T}\}$. Let R be a relation that obeys $\vee\{T : T \notin \Sigma^*_{\text{fin}}\}$; we must show that R obeys $\vee\{T' : T' \in \mathscr{T}\}$. Since R obeys $\vee\{T : T \notin \Sigma^*_{\text{fin}}\}$, this means that R obeys some T not in Σ^*_{fin}. By (c), we know that $T \models \vee\{T' : T' \in \mathscr{T}\}$, so R obeys $\vee\{T' : T' \in \mathscr{T}\}$, which was to be shown.

(d)$\Rightarrow$(a). Assume that (d) holds. By Fact 2 above, there is a finite relation R that obeys every TD in Σ^*_{fin} but violates every member of $\mathscr{T}$. Since R violates every member of $\mathscr{T}$, we know that R violates $\vee\{T' : T' \in \mathscr{T}\}$. By assumption, $\vee\{T' : T' \in \mathscr{T}\}$ is equivalent to $\vee\{T : T \notin \Sigma^*_{\text{fin}}\}$. Thus, R violates $\vee\{T : T \notin \Sigma^*_{\text{fin}}\}$. Hence, R obeys Σ^*_{fin} and no other TD's, which was to be shown. □

As a simple application of Theorem 10.2, we now show that there is a finite Armstrong relation for the empty set, that is, that there is a finite relation that violates every nontrivial TD. Let $\mathscr{T}$ be the set of weakest nontrivial V-partial TD's, one for every subset V, with at least two members, of the set U of attributes. These weakest

nontrivial V-partial TD's exist by Theorem 3.3. But this set $\mathscr{T}$ can play the role of $\mathscr{T}$ in (b) of Theorem 10.2. Hence, (a) of Theorem 10.2 holds, and so the empty set has a finite Armstrong relation.

11. Acknowledgment. The authors are grateful to Moshe Vardi for several useful suggestions.

REFERENCES

[BFMMUY] C. BEERI, R. FAGIN, D. MAIER, A. O. MENDELZON, J. D. ULLMAN AND M. YANNAKAKIS, *Properties of acyclic database schemes*, Proc. Thirteenth Annual ACM Symposium on the Theory of Computing, 1981, pp. 355–362.

[BFMY] C. BEERI, R. FAGIN, D. MAIER AND M. YANNAKAKIS, *On the desirability of acyclic database schemes*, J. Assoc. Comput. Mach., to appear.

[BV1] C. BEERI AND M. Y. VARDI, *A proof procedure for data dependencies*, Technical Report, Hebrew Univ. of Jerusalem, August 1980.

[BV2] ———, *Formal systems for tuple and equality generating dependencies*, Technical Report, Hebrew Univ. of Jerusalem, April 1981.

[BV3] ———, *The implication problem for data dependencies*, Proc. 8th ICALP, Acre, Israel, July 1981, in Lecture Notes in Computer Science 115, Springer-Verlag, New York, 1981, pp. 73–85.

[CFP] M. A. CASANOVA, R. FAGIN AND C. PAPADIMITRIOU, *Inclusion dependencies and their interaction with functional dependencies*, Proc. First ACM SIGACT-SIGMOD Principles of Database Systems, 1982, pp. 171–176.

[Co] E. F. CODD, *Further normalization of the data base relational model*, in Courant Computer Science Symposia 6: Data Base Systems, May 24–25, 1971, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1971, pp. 33–64.

[De] C. DELOBEL, *Normalization and hierarchical dependencies in the relational data model*, ACM Trans. Database Systems, 3 (1978), pp. 201–222.

[En] H. B. ENDERTON, *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.

[Fa1] R. FAGIN, *Probabilities on finite models*, J. Symbolic Logic, 41 (1976), pp. 50–58.

[Fa2] ———, *Multivalued dependencies and a new normal form for relational databases*, ACM Trans. Database Systems, 2 (1977), pp. 262–278.

[Fa3] ———, *Horn clauses and database dependencies*, Proc. 1980 ACM SIGACT Symposium on Theory of Computing, pp. 123–134. J. Assoc. Comput. Math., to appear.

[FMU] R. FAGIN, R. A. O. MENDELZON AND J. D. ULLMAN, *A simplified universal relation assumption and its properties*, ACM Trans. Database Systems, to appear.

[GZ] S. GINSBURG AND S. M. ZAIDDAN, *Properties of functional dependency families*, J. Assoc. Comput. Math., 29 (1982), pp. 678–698.

[GL] Y. GUREVICH AND H. R. LEWIS, *The inference problem for template dependencies*, Proc. First ACM SIGACT-SIGMOD Principles of Database Systems, 1982, pp. 221–229.

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

[MM] A. MENDELZON AND D. MAIER, *Generalized mutual dependencies*, in Proc. 1979 Very large Data Bases Conference, pp. 75–82.

[Ni] J.-M. NICOLAS, *Mutual dependencies and some results on undecomposable relations*, in Proc. 1978 Very Large Data Bases Conference, pp. 360–367.

[PJ] J. PARADAENS AND D. JANNSENS, *Decomposition of relations: A comprehensive approach*, in Formal Bases for Databases, J. Minker and H. Gallaire, ed., Plenum, New York, 1978.

[Ri] J. RISSANEN, *Theory of relations for databases—A tutorial survey*, Proc. 7th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, 64, Springer-Verlag, New York, pp. 537–551.

[Sa] F. SADRI, Personal communication.

[SU] F. SADRI AND J. D. ULLMAN, *Template dependencies: a large class of dependencies in relational databases and its complete axiomatization*, J. Assoc. Comput. Mach., 29 (1982), pp 363–372.

[SW] Y. SAGIV AND S. WALECKA, *Subset dependencies and a completeness result for a subclass of embedded multivalued dependencies*, J. Assoc. Comput. Mach., 29 (1982), pp. 103–117.

[Va1] M. Y. VARDI, *The implication and finite implication problems for typed template dependencies*, in Proc. First ACM SIGACT-SIGMOD Principles of Database Systems, 1982, pp. 230–238.

[Va2] ———, Private communication, Oct. 1981.

[YP] M. YANNAKAKIS AND C. PAPADIMITRIOU, *Algebraic dependencies*, Proc. 1980 IEEE Symposium on Foundations of Computer Science, pp. 328–332.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0004 $01.25/0

BOUNDS FOR MULTIFIT SCHEDULING ON UNIFORM PROCESSORS*

D. K. FRIESEN† AND M. A. LANGSTON‡

Abstract We examine the nonpreemptive assignment of N independent tasks to a system of M uniform processors with the objective of reducing the makespan, or the time required from the start of execution until all tasks are completed. Since the problem of finding a minimal makespan has been shown to be NP-hard, and hence unlikely to permit an efficient solution procedure, near-optimal heuristic algorithms have been studied. It is known that LPT (longest processing time first) schedules are within twice the length of the optimum. We analyze a variation of the MULTIFIT algorithm derived from bin packing, and prove that its worst-case performance bound is within 1.4 of the optimum.

Key words. multiprocessor scheduling, worst-case performance, heuristic algorithms, bin packing, uniform processors, near-optimal schedules, independent tasks

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 $L=\{a_1, a_2, \cdots, a_N\}$ 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 demonstrated 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 results.

When all elements of P are exactly the same, we say that we have an identical multiprocessor system. For this special case the LPT (Largest Processing Time first) algorithm has been analyzed [Gr] and proved to have a "tight" worst-case performance bound of $\frac{4}{3}-\frac{1}{3}M$. Informally, this means that the length of an LPT schedule can be no more than about 33% longer than the optimum. Techniques derived from bin packing are used in the MULTIFIT algorithm [CGJ]. It has been shown [Fr] that the worst-case performance of MULTIFIT lies between $\frac{13}{11}=1.1818$ and 1.2.

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 $\{r_1, r_2, \cdots, r_m\}$, 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]. It is known from [GIS] that the LPT algorithm can be implemented such that its worst-case bound is between 1.5 and 2 times optimal. In this paper we modify the MULTIFIT algorithm for the uniform multiprocessor case and prove that its worst-case performance is substantially better than the LPT algorithm, with a worst-case bound lying between 1.341 and 1.4.

Our work is organized as follows. The next section introduces the necessary notation and discusses some implementation details and preliminary results. In § 3,

* Received by the editors July 17, 1980, and in final revised form April 25, 1982.

† Division of Computer Science, Department of Industrial Engineering, Texas A&M University, College Station, Texas 77843.

‡ Department of Computer Science, Washington State University, Pullman, Washington 99164.

we assume the existence of a counterexample to our desired bound of 1.4, and hence the existence of a minimal counterexample whose properties we analyze. Section 4 contains the proof of our main result. We show that MULTIFIT can do no worse than 1.4 times the optimum by establishing a contradiction based on the presumed existence of a counterexample.

The final section of this paper contains the worst example we have found and some remarks and suggestions for further research.

2. Notation and preliminary results. In this section we describe the notation we use in the proof of Theorem 4.1 and some results needed to state it. To transform a scheduling problem to a bin packing problem, we consider each processor as a bin and, in the case of identical processors, the size of the bin corresponds to the schedule length or deadline (see [CGJ] for more details). Extending this idea to uniform processors, we fix the bin size corresponding to one of the processors, say P_1, the slowest one. Then for the ith processor P_i we let its bin size be r_i times that of P_1, where r_i is the ratio of the speed of P_i to the speed of P_1. We assume that $r_1 \leqq r_2 \leqq \cdots \leqq r_M$.

The FFD bin packing algorithm arranges the list of items in nonincreasing order of size. Then each item in the list is placed in the first bin (the P_i with smallest subscript) in which it will fit. To change deadlines, all bin sizes are multiplied by a constant, or expansion factor. We would like to find the smallest expansion factor, R_0, such that any list that can be packed in a set of bins of sizes $\{\alpha_1, \alpha_2, \cdots, \alpha_M\}$, will be successfully packed by the FFD algorithm when the bin sizes are multiplied by the expansion factor R_0.

If there are M processors, we use $P = \{P_1, P_2, \cdots, P_M\}$ to denote the M bins of the FFD packing and P^* to denote those of the optimal packing. We let α_i denote the size of P_i^*, let R be the expansion factor (i.e. the ratio of the size of P_i to that of P_i^*), and let $\beta_i = R\alpha_i$. We use FFD(β_1, L) to describe the success or failure of the packing. If the FFD packing of a list meets the deadline, then FFD(β_1, L) = succeed. If the packing fails to pack all items of L, we set FFD(β_1, L) = fail. Our main result states that FFD(β_1, L) = succeed if $\beta_1 \geqq \frac{7}{5}\alpha_1$.

To implement the algorithm MULTIFIT, we need an upper bound and a lower bound for the FFD schedule length. Then a binary search can be used to find an acceptable schedule. Note that the binary search is not guaranteed to find the *least* FFD binsize which works for a particular list. But since any β_1 greater than or equal to $R_0\alpha_1$ will suffice (see [CGJ] for a discussion of this "monotonicity" property), we know that the binary search will converge to a binsize $\leqq R_0\alpha_1$.

We utilize the results on LPT schedules to obtain the needed lower and upper bounds. We know from [GIS] that LPT can do no worse than twice the optimum. We thus first apply the LPT algorithm and then use:

$$\text{lower bound} = \max\left\{\frac{S(L)}{r_1 + \cdots + r_M}, \frac{1}{2}\text{LPT}\right\},$$

$$\text{upper bound} = \text{LPT}.$$

Naturally, if each iteration of our binary search fails to produce an acceptable FFD packing and the binsize converges on the upper bound, we select the LPT schedule since it must already be at most R_0 times the optimal schedule length.

Following the discussion in [CGJ] and using the techniques described in [Jo], k iterations of MULTIFIT can be performed in $O(N \log N + kN \log M)$ time, comparable to the LPT timing of $O(N \log N + N \log M)$. For large N, the time required for

both algorithms is dominated by the $O(N \log N)$ term of the initial sort. Using a binary search scheme, no more than seven iterations of MULTIFIT are necessary to produce a schedule whose finish time is less than or equal to $(R_0 + .01)$ times optimal.

3. Properties of a minimal counterexample. We suppose now that $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_M\}$ is a set of optimal bin sizes and that there is a list of items L such that FFD $(\beta_1 = \frac{7}{5}\alpha_1, L) =$ fail. To simplify our argument we assume that L is minimal—that no set of fewer than M bin sizes can be used to provide a counterexample and that, given M, no list with fewer than L items will fail to be packed into $\frac{7}{5}\alpha$ by FFD.

Thus we can assume that the FFD packing with expansion factor $\frac{7}{5}$ packed all items but the last. For convenience we normalize all bin sizes and item sizes so that the final item has size 1. Then every bin P_i is filled to more than $\beta_i - 1$.

In this minimal counterexample, we use the concept of domination from [CGJ] to prove our cancellation lemma, Lemma 3.1. Let I and J be ordered sets of indices representing sets of bins in P or P^*. We say that I dominates J (or alternatively, that the bins of I dominate the bins of J) if there is a one-to-one mapping f from items packed in the bins represented by J to items packed in the bins represented by I such that for any l in a bin of J, $l = f(l)$ if l is in a bin of I and $s(l) \leqq s(f(l))$ otherwise.

LEMMA 3.1. *Let $I, J \subseteq \{1, 2, \cdots, M\}$ represent ordered sets of bins. If $|I| = |J| = n > 0$ and $I_k \leqq J_k$ for $1 \leqq k \leqq n$, then $\bigcup_{i \in I} P_i$ cannot dominate $\bigcup_{j \in J} P_j^*$.*

Proof. Suppose we do have such domination. Consider the removal of all items in $\bigcup_{i \in I} P_i$ from both packings, leaving n empty bins in the FFD packing. Next we modify the optimal packing as follows: (1) move each remaining element l of $\bigcup_{j \in J} P_j^*$ to the position formerly occupied by $f(l)$, leaving n empty bins since $f(l)$ does not belong to a bin of J, and (2) for $k = 1, 2, \cdots, n$, move all items in $P_{I_k}^*$ to $P_{J_k}^*$. Now delete $\bigcup_{i \in I} \alpha_i$ from α. Thus we have constructed $L' = L - \bigcup_{i \in I} P_i < L$ and $M' = M - n < M$ such that FFD $(\beta_1, L') =$ fail, contradicting the presumed minimality of L and M. □

As a simple consequence we can show that each bin P_i^* for our minimal counterexample must contain at least two elements.

LEMMA 3.2. *If P_i^* is any bin in the optimal packing of L, then $|P_i^*| \geqq 2$.*

Proof. Suppose $P_i^* = \{x\}$. Since x would fit in P_i either the first item placed in P_i is as large as x, and P_i dominates P_i^*, or x was not available when P_i was packed. Then $x \in P_j, j < i$ and P_j dominates P_i^*. Thus Lemma 3.1 is contradicted in either case. □

Since $s(a) \geqq 1$ for all a, we can conclude from this lemma that $\alpha_i \geqq 2$ for all i. For any set A of items, let $S(A) = \sum_{a \in A} s(a)$.

LEMMA 3.3. *If $|P_i| = 1$, then $s(P_i) > \frac{9}{5}$.*

Proof. Since $\alpha_i \geqq 2$ and $s(P_i) > \frac{7}{5}\alpha_i - 1$, we have $s(P_i) > \frac{7}{5}(2) - 1 = \frac{9}{5}$. □

LEMMA 3.4. *If $s(P_i) < s(P_i^*)$, then $s(P_i) < \alpha_i < \frac{5}{2}$.*

Proof. If $s(P_i) < s(P_i^*)$, then $\frac{7}{5}\alpha - 1 < \alpha_i$ and $s(P_i) < s(P_i^*) \leqq \alpha_i < \frac{5}{2}$. □

LEMMA 3.5. *If P_i is any bin of the* FFD *packing such that $|P_i| \geqq 2$, then $s(P_i) \geqq s(P_i^*)$.*

Proof. Assume $s(P_i) < s(P_i^*)$. Then $\alpha_i < \frac{5}{2}$ by Lemma 3.4. Thus by Lemma 3.2 and the fact that any element has size $\geqq 1$ we have $|P_i| = |P_i^*| = 2$. Let $P_i^* = \{a, b\}$, $s(a) \geqq s(b)$, $P_i = \{u, v\}$, $s(u) \geqq s(v)$.

Suppose $s(a) > s(u)$. Then a must have been packed before u. Thus $a \in P_j$, $j < i$. By Lemma 3.2, $\beta_j = \frac{7}{5}\alpha_j \geqq \frac{14}{5}$. Moreover $s(a) + s(b) < \frac{5}{2} < \frac{14}{5}$ and hence b would fit in P_j. Thus $b \in P_k$, $k < j$, else P_j would dominate P_i^*. But then P_k must also contain an item at least as large as a since $\beta_k \leqq \frac{14}{5}$ and $a \leqq \frac{5}{2}$. Hence P_k dominates P_i^*. In either case, we contradict Lemma 3.1.

Suppose now that $s(a) \leqq s(u)$. If $s(b) \leqq s(v)$, then P_i would dominate P_i^*. Thus $s(b) > s(v)$ and either $b \in P_j$, $j < i$, or $s(u) + s(b) > \beta_i$. Noting that $s(v) \geqq s(b) - \frac{1}{4}$, since $s(b) \leqq \frac{1}{2}\alpha_i \leqq \frac{5}{4}$ and $s(v) \geqq 1$, we then have in the latter case, $s(u) + s(v) > \beta_i - \frac{1}{4} = \frac{7}{5}\alpha_i - \frac{1}{4} > \alpha_i \geqq s(P_i^*)$, contradicting our original hypothesis.

In the former case, u would have fit in P_j since $s(u) < s(P_i) < s(P_i^*) < \frac{5}{2} < \frac{14}{5} \leqq \frac{7}{5}\alpha_j = \beta_j$ by Lemma 3.2. Thus P_j contains both b and some item t such that $s(t) \geqq s(u) \geqq s(a)$.Hence P_j dominates P_i^*, again contradicting Lemma 3.1. □

The bins of P will be classified by type according to the following scheme. If, after P_i receives its first item, there is a total of k items in P_i when the next item is placed in a bin that follows P_i, then P_i is called a k-bin. (Note that this excludes the possibility of having 0-bins.) If no additional items are placed in P_i it is called a regular k-bin, otherwise it is called a fallback k-bin and the subsequent item, or items, are called fallback items. Items in a regular k-bin will be called items of type X_k, the first k items in a fallback k-bin will be of type Y_k and fallback items of type F.

The final step in preparation for the proof of the main result is to define a weight function w. We do this by assigning the weight $w(a)$ of an item, a, to be $s(a)$ if a is an item of type X_1 and $s(a) \leqq \frac{14}{5}$. $w(a)$ will be $s(a) - \frac{1}{5}$ if one of the following holds:

(1) a is a fallback item in a fallback 1- or 2-bin;

(2) a is in a regular 2-bin P_i, and P_i^* contains no items of type X_1;

(3) a is an item of type Y_1 in a fallback 1-bin P_i, and P_i^* contains no items of type X_1;

(4) a is an item of type Y_2 in a fallback 2-bin P_i such that $s(P_i) > s(P_i^*) + \frac{3}{5}$.

In all other cases $w(a) = s(a) - \frac{1}{10}$. This information is summarized in Table 3.1.

We extend the function w to sets of items by $w(A) = \sum_{a \in A} w(a)$. In the next section we will show that $w(P_i) \geqq w(P_i^*)$ for almost all bins. Using this we can prove that the FFD algorithm will pack the items of L in bins at most $\frac{7}{5}$ the size of the optimal packing.

TABLE 3.1
Item types and weights

Bin type	Item types	Restriction	Weights
Regular 1	X_1	$s(P_i) \leqq \frac{14}{5}$	s
		$s(P_i) > \frac{14}{5}$	$s - \frac{1}{10}$
Fallback 1	Y_1, F	P_i^* contains no X_1	$s - \frac{1}{5}, s - \frac{1}{5}$
		P_i^* contains an X_1	$s - \frac{1}{10}, s - \frac{1}{5}$
Regular 2	X_2	P_i^* contains no X_1	$s - \frac{1}{5}$
		P_i^* contains an X_1	$s - \frac{1}{10}$
Fallback 2	Y_2, F	$s(P_i) \leqq s(P_i^*) + \frac{3}{5}$	$s - \frac{1}{10}, s - \frac{1}{5}$
		$s(P_i) > s(P_i^*) + \frac{3}{5}$	$s - \frac{1}{5}, s - \frac{1}{5}$
Other	Other	–	$s - \frac{1}{10}$

4. Proof of the main result. In this section we prove that using the MULTIFIT algorithm for scheduling uniform processors produces a schedule whose length is at most $\frac{7}{5}$ times the minimal schedule length, that is, FFD $(\beta_1 = \frac{7}{5}\alpha_1, L) =$ succeed for any list L. In a sequence of lemmas preceding the result, we narrow the possibilities that could occur in a minimal counterexample. Using the weight function w described in § 3, we show that in almost all cases, the sum of the weights of the items in an FFD-packed bin P_i is at least as great as the sum of those in the corresponding optimal

bin P_i^*. Moreover, in all cases but one, if $w(P_i) < w(P_i^*)$ then the loss is compensated for by a gain in an easily specified bin P_j.

LEMMA 4.1. *If* $|P_i| = 1$, *then* $w(P_i) \geqq w(P_i^*)$.

Proof. If $\alpha_i < \frac{14}{5}$, then by Lemmas 3.2 and 3.3, P_i^* cannot contain an item of type X_1. Hence $w(P_i^*) \leqq \alpha_i - 2(\frac{1}{10})$. In this case $w(P_i) = s(P_i) \geqq \frac{7}{5}\alpha_i - 1$ and $\frac{7}{5}\alpha_i - 1 \geqq \alpha_i - \frac{1}{5}$ since $\alpha_i \geqq 2$, by Lemma 3.2.

If $\alpha_i \geqq \frac{14}{5}$, $w(P_i) \geqq s(P_i) - \frac{1}{10} \geqq \frac{7}{5}\alpha_i - 1 - \frac{1}{10}$. Thus all we need to show is that $\frac{7}{5}\alpha_i - \frac{11}{10} > \alpha_i$. This is true for all $\alpha_i > \frac{11}{4}$, and $\alpha_i \geqq \frac{14}{5} > \frac{11}{4}$. □

LEMMA 4.2. *If* P_i *is a fallback* 1*-bin, then* $w(P_i) \geqq w(P_i^*)$.

Proof. Suppose $w(P_i) < w(P_i^*)$. (Note that the first item of P_i must be larger than half the bin size.)

Case 1. Suppose $|P_i| = k \geqq 3$. Since P_i is a fallback 1-bin, the first item placed must be larger than the sum of the other $k-1$ items. Hence $s(P_i) > 2(k-1)$ and also $s(P_i) > \frac{7}{5}\alpha_i - 1$.

Case 1A. Suppose P_i^* contains no X_1 items. Then $w(P_i) \geqq \max(\frac{7}{5}\alpha_i - 1 - \frac{1}{5}k,$ $2(k-1) - \frac{1}{5}k)$ and $w(P_i^*) \leqq \alpha_i - \frac{1}{5}$ since $|P_i^*| \geqq 2$ by Lemma 3.2. If $w(P_i^*) > w(P_i)$ then

$$\tfrac{7}{5}\alpha_i - 1 - \tfrac{1}{5}k < \alpha_i - \tfrac{1}{5}$$

and hence

$$\alpha_i < 2 + \tfrac{1}{2}k.$$

Also

$$2(k-1) - \tfrac{1}{5}k < \alpha_i - \tfrac{1}{5}$$

and hence

$$\tfrac{9}{5}k - \tfrac{9}{5} < \alpha_i.$$

Combining these yields

$$\tfrac{9}{5}k - \tfrac{9}{5} < 2 + \tfrac{1}{2}k,$$

implying

$$k < \tfrac{38}{13} < 3, \quad \text{a contradiction.}$$

Case 1B. Suppose that there is exactly one X_1 item in P_i^*. Then $w(P_i) \geqq$ $\max(\frac{7}{5}\alpha_i - 1 - \frac{1}{5}(k-1) - \frac{1}{10}, 2(k-1) - \frac{1}{5}(k-1) - \frac{1}{10})$ and $w(P_i^*) \leqq \alpha_i - \frac{1}{10}$. As in Case 1A above, we derive $\frac{9}{5}k - \frac{9}{5} < 2 + \frac{1}{2}k$ and thus $k < 3$, a contradiction.

Case 1C. Suppose P_i^* contains two or more X_1 items. Then $w(P_i) \geqq$ $\max(\frac{7}{5}\alpha_i - 1 - \frac{1}{5}(k-1) - \frac{1}{10},\ 2(k-1) - \frac{1}{5}(k-1) - \frac{1}{10})$ and $w(P_i^*) \leqq \alpha_i$. If $w(P_i^*) > w(P_i)$ then $\frac{7}{5}\alpha_i - 1 - \frac{1}{5}(k-1) - \frac{1}{10} < \alpha_i$ and hence $\alpha_i < \frac{9}{4} + \frac{1}{2}k$. Also $2(k-1) - \frac{1}{5}(k-1) - \frac{1}{10} < \alpha_i$ and hence $\frac{9}{5}k - \frac{19}{10} < \alpha_i$. Combining these yields $k < \frac{83}{26} < 4$. Thus $k = 3$. From Lemma 3.3 we know that $\alpha_i \geqq \frac{18}{5}$ since P_i contains two X_1 items. This implies $\beta_i \geqq \frac{126}{25}$. Hence $s(P_i) > \frac{63}{25} + 2 = \frac{113}{25}$ since the first item is at least $\frac{1}{2}$ the bin size. From the table, $k = 3$ implies $w(P_i) \geqq s(P_i) - \frac{1}{2}$. Thus $w(P_i) > \frac{113}{25} - \frac{1}{2} > \frac{9}{4} + \frac{1}{2}k > \alpha_i > w(P_i^*)$.

Case 2. Suppose $|P_i| = 2$. Let $P_i = \{l, m\}$, $s(l) \geqq s(m)$.

Case 2A. Suppose P_i^* contains an item of type X_1. Then according to the table, $w(P_i) = s(P_i) - \frac{3}{10} \geqq \frac{7}{5}\alpha_i - \frac{13}{10}$. Since $w(P_i^*) \leqq \alpha_i$ in any case, we are done if $\frac{7}{5}\alpha_i - \frac{13}{10} \geqq \alpha_i$ (i.e. $\alpha_i \geqq \frac{13}{4}$). So assume $\alpha_i < \frac{13}{4}$. Then, since $2(\frac{9}{5}) > \frac{13}{4}$, P_i^* cannot contain two items of type X_1 by Lemma 3.3. Thus $w(P_i^*) \leqq s(P_i^*) - \frac{1}{10}$. If $w(P_i) < w(P_i^*)$, then

$$\tfrac{7}{5}\alpha_i - \tfrac{13}{10} < \alpha_i - \tfrac{1}{10}$$

and hence

$$\alpha_i < 3.$$

Thus the size of the X_1-item, x, is less than 2 by Lemma 3.2.

Since P_i is a fallback 1-bin, $2s(l) > \frac{7}{5}\alpha_i$ and hence

$$s(l) > \tfrac{7}{10}\alpha_i \geqq \tfrac{7}{10}(s(x)+1) > s(x).$$

We must now have $s(l) > \frac{14}{5}$ or l would be an item of type X_1 packed before x since it would fit in any bin. Hence

$$w(P_i) \geqq s(l) + s(m) - \tfrac{3}{10} > \tfrac{35}{10} > 3 > \alpha_i.$$

Case 2B. Suppose P_i^* contains no item of type X_1. In this case, $w(P_i) = s(P_i) - \frac{2}{5}$ and $w(P_i^*) \leqq s(P_i^*) - \frac{1}{5}$. If $w(P_i) < w(P_i^*)$, then

$$\tfrac{7}{5}\alpha_i - 1 - \tfrac{2}{5} < \alpha_i - \tfrac{1}{5}$$

and

$$\alpha_i < 3.$$

Thus $|P_i^*| = 2$ and we let $P_i^* = \{a, b\}$, $s(a) \geqq s(b)$.

Since P_i is a fallback 1-bin, $s(l) > \frac{1}{2}(\frac{7}{5}\alpha_i) = \frac{7}{10}\alpha_i$. If $s(a) > s(l)$, then $s(a) > \frac{7}{10}\alpha_i$ and $1 \leqq s(b) < \frac{3}{10}\alpha_i$. From this it would follow that $\alpha_i > \frac{10}{3}$, contradicting $\alpha_i < 3$. We conclude that $s(a) \leqq s(l)$. Also, we note that if $s(l) > \frac{14}{5}$, $s(P_i) > \frac{19}{5}$ and $w(P_i) > \frac{17}{5} > \alpha_i > w(P_i^*)$. Thus $s(l) \leqq \frac{14}{5}$ and every P_k, $k < i$, contains an item $\geqq l$ in size.

If $s(b) \leqq s(m)$, P_i would dominate P_i^*, so we assume $s(b) > s(m)$. Suppose $b \in P_k$, $k < i$. Then P_k contains an item c with $s(c) \geqq s(l) \geqq s(a)$ and P_k dominates P_i^*. Thus b must have been available when m was packed. Since it wasn't used, b must not have fit and

$$s(l) + s(b) > \tfrac{7}{5}\alpha_i$$

while

$$s(l) + s(m) - \tfrac{2}{5} \leqq w(P_i) < w(P_i^*) \leqq \alpha_i - \tfrac{1}{5}.$$

Subtracting, we obtain

$$s(b) > \tfrac{2}{5}\alpha_i + s(m) - \tfrac{1}{5} \geqq \tfrac{8}{5},$$

and

$$\alpha_i > s(a) + s(b) > 2s(b) > 3, \quad \text{a contradiction.} \qquad \square$$

LEMMA 4.3. *If* $|P_i| = 2$ *and* P_i *is regular, then* $w(P_i) \geqq w(P_i^*)$.

Proof. Suppose $w(P_i) < w(P_i^*)$.

Case 1. Suppose P_i^* contains an item of type X_1. Then $\alpha_i \geqq \frac{14}{5}$ since $s(P_i^*) \geqq \frac{9}{5} + 1$ by Lemma 3.3. $w(P_i) = s(P_i) - \frac{1}{5} \geqq \frac{7}{5}\alpha_i - \frac{6}{5}$. If $\alpha_i \geqq 3$, $\frac{7}{5}\alpha_i - \frac{6}{5} \geqq \alpha_i$ so we can assume $\alpha_i < 3$, $|P_i^*| = 2$, and P_i^* contains just *one* item of type X_1 (by Lemma 3.3), but then

$w(P_i^*) \leqq \alpha_i - \frac{1}{10}$ and

$$\tfrac{7}{5}\alpha_i - \tfrac{6}{5} < \alpha_i - \tfrac{1}{10}$$

would imply

$$\alpha_i < \tfrac{11}{4}, \quad \text{contradicting } \alpha_i \geqq \tfrac{14}{5}.$$

Case 2. Suppose P_i^* contains no item of type X_1. Then

$$w(P_i) \geqq \tfrac{7}{5}\alpha_i - \tfrac{7}{5} \quad \text{and} \quad w(P_i^*) \leqq \alpha_i - \tfrac{1}{5}.$$

Solving for α_i we get $\alpha_i < 3$ if $w(P_i) < w(P_i^*)$. Thus $|P_i^*| = 2$. Let $P_i^* = \{a, b\}$, $s(a) \geqq s(b)$ and $P_i = \{u, v\}$, $s(u) \geqq s(v)$.

Suppose $s(a) > s(u)$. Since $\alpha_i < 3$, $s(a) < 2$. Then $a \in P_j$, $j < i$. If P_j does not dominate P_i^*, then P_j cannot contain a second item as large as b. Thus either $b \in P_k$, $k < j$, or $s(a) + s(b) > \beta_j \geqq \frac{14}{5}$. In the former case, P_k dominates P_i^* since P_k must contain an item at least as large as a since a is available and would have fit in P_k. Thus $s(a) + s(b) > \beta_j$ and P_j is a fallback 1-bin since a is not an X_1 item by assumption. By Lemma 3.2, $|P_j^*| \geqq 2$. If P_j^* contains an X_1 item, $\alpha_j \geqq \frac{14}{5}$ by Lemma 3.3 and $s(a) + s(b) > \frac{7}{5}\alpha_j > 3 > \alpha_i$, which is impossible. Thus P_j^* can contain no X_1 items and $w(a) = s(a) - \frac{1}{5}$. Hence $w(P_i^*) \leqq \alpha_i - \frac{3}{10}$. Since $w(P_i) \geqq \frac{7}{5}\alpha_i - \frac{7}{5}$, if $w(P_i) < w(P_i^*)$ we would have $\alpha_i < \frac{11}{4}$. But now $s(a) + s(b) < \frac{11}{4} < \frac{14}{5}$ and $s(a) + s(b) < \beta_j$, a contradiction.

Suppose now that $s(a) \leqq s(u)$. If $s(b) \leqq s(v)$, then P_i dominates P_i^*, so we conclude $s(b) > s(v)$. Moreover, if $b \in P_j$, $j < i$, P_j would dominate P_i^* since P_j must contain an item at least as large as a. Thus

$$\text{(i)} \qquad s(u) + s(b) > \beta_j$$

since b was available and not used. But $s(u) + s(v) - \frac{2}{5} \leqq w(P_i) < w(P_i^*) < \alpha_i - \frac{1}{5}$ so

$$\text{(ii)} \qquad s(u) + s(v) < \alpha_i + \tfrac{1}{5}.$$

Subtracting (ii) from (i) we get $s(b) > \frac{2}{5}\alpha_i - \frac{1}{5} + s(v) \geqq \frac{8}{5}$ and $\alpha_i \geqq 2s(b) > \frac{16}{5} > 3$, a contradiction. □

LEMMA 4.4. *If P_i is a fallback 2-bin, then $w(P_i) \geqq w(P_i^*)$.*

Proof. Assume $w(P_i) < w(P_i^*)$.

Case 1. Suppose $|P_i| = k \geqq 4$. Since P_i is a fallback 2-bin, the size of the second item packed in P_i is more than the sum of the sizes of the $k-2$ smallest items. Hence $s(P_i) > 3(k-2)$ and $w(P_i) > 3(k-2) - \frac{1}{5}k$. Also, as usual, $w(P_i) \geqq \beta_i - 1 - \frac{1}{5}k$. If $w(P_i^*) > w(P_i)$, then $\frac{7}{5}\alpha_i - 1 - \frac{1}{5}k < \alpha_i$ and hence $\alpha_i < \frac{5}{2} + \frac{1}{2}k$. Also $3(k-2) - \frac{1}{5}k < \alpha_i$ and hence $\frac{14}{5}k - 6 < \alpha_i$. Thus $\frac{14}{5}k - 6 < \frac{5}{2} + \frac{1}{2}k$. Solving this inequality for k yields $k < \frac{85}{23} < 4$, contradicting the assumption of Case 1.

Case 2. Suppose $|P_i| = 3$. Let $P_i = \{u, v, m\}$, $s(u) \geqq s(v) \geqq s(m)$.

Case 2A. $s(P_i) - s(P_i^*) \leqq \frac{3}{5}$. Then $w(P_i) = s(P_i) - \frac{2}{5} \geqq \frac{7}{5}\alpha_i - \frac{7}{5}$. If $\alpha_i \geqq \frac{7}{2}$, then $w(P_i) \geqq \frac{7}{5}\alpha_i - \frac{7}{5} \geqq \alpha_i \geqq w(P_i^*)$. Hence we may assume $\alpha_i < \frac{7}{2}$, $|P_i^*| \leqq 3$, and P_i^* cannot contain more than one item of type X_1, since each such item has a size exceeding $\frac{9}{5}$. Thus $w(P_i^*) \leqq \alpha_i - \frac{1}{10}$, and if $w(P_i) < w(P_i^*)$ we have

$$s(P_i) < \alpha_i + \tfrac{3}{10}$$

since $w(P_i) = s(P_i) - \frac{2}{5}$.

Since P_i is a fallback 2-bin, $s(u) + 2s(v) > \frac{7}{5}\alpha_i$. But $s(u) + s(v) + s(m) < \alpha_i + \frac{3}{10}$. Subtracting we get $s(v) > \frac{2}{5}\alpha_i + s(m) - \frac{3}{10} \geqq \frac{3}{2}$, since $\alpha_i \geqq 3$. $s(P_i) \geqq 2s(v) + 1 \geqq 4$ and, since $\alpha_i < \frac{7}{2}$, $4 < \alpha_i + \frac{3}{10} < \frac{19}{5}$, which is impossible.

Case 2B. $s(P_i) - s(P_i^*) > \frac{3}{5}$. Then $w(P_i) = s(P_i) - \frac{3}{5} > s(P_i^*) \geqq w(P_i^*)$. □

LEMMA 4.5. *If* $|P_i| \geqq 4$, *then* $w(P_i) \geqq w(P_i^*)$.

Proof. Lemmas 4.2 and 4.4 take care of the cases in which P_i may be a fallback 1- or 2-bin. Thus for any $a \in P_i$, $w(a) = s(a) - \frac{1}{10} \geqq \frac{9}{10} s(a)$. Hence $w(P_i) \geqq \frac{9}{10} s(P_i) \geqq (\frac{9}{10})(\frac{7}{5}\alpha_i - 1)$. If $w(P_i) < w(P_i^*)$, then

$$\tfrac{9}{10}(\tfrac{7}{5}\alpha_i - 1) < \alpha_i$$

and

$$\tfrac{13}{50}\alpha_i < \tfrac{9}{10}, \qquad \alpha_i < \tfrac{45}{13}.$$

Then $\frac{7}{5}\alpha_i < \frac{63}{13} < 5$ and $|P_i| = 4$. But then $w(P_i) = s(P_i) - \frac{4}{10} \geqq 4 - \frac{4}{10} > \frac{45}{13} > w(P_i^*)$. □

LEMMA 4.6. *If* $w(P_i) < w(P_i^*)$, *then* P_i *is a regular 3-bin,* $|P_i^*| = 2$, P_i^* *contains exactly one* X_1 *type item, and* $\alpha_i < \frac{13}{4}$.

Proof. Suppose $w(P_i) < w(P_i^*)$. From Lemmas 4.1–4.5 we know that P_i must be a regular 3-bin. Hence $w(P_i) = s(P_i) - \frac{3}{10} > \frac{7}{5}\alpha_i - 1 - \frac{3}{10}$ and $\alpha_i \geqq w(P_i^*) > w(P_i)$ implies $\alpha_i \leqq \frac{13}{4}$. Consequently $|P_i^*| \leqq 3$, and P_i^* cannot contain two items of type X_1 or one such item and two other items. If $|P_i^*| = 3$, then the fact that, by Lemma 3.5, $s(P_i) > s(P_i^*)$ implies $w(P_i) > w(P_i^*)$ since $w(P_i^*) = s(P_i^*) - \frac{3}{10}$. If $|P_i^*| = 2$ and P_i^* contains no X_1-type item, then $w(P_i^*) \leqq \alpha_i - \frac{1}{5}$ and

$$\tfrac{7}{5}\alpha_i - 1 - \tfrac{3}{10} < \alpha_i - \tfrac{1}{5}$$

implies $\alpha_i < \frac{11}{4}$. But then $s(P_i) \geqq 3$ implies $\frac{7}{5}\alpha_i - 1 \geqq 3$ and hence $\alpha_i \geqq \frac{20}{7}$, and so we have our final contradiction. □

LEMMA 4.7. *Let* $w(P_I) < w(P_I^*)$ *and* $P_I^* = \{x, z\}$, $x \in P_j$, $j < I$, x *an* X_1 *item,* z *not. Then there is at most one such bin* P_I *for which* $w(P_I) + w(P_j) < w(P_I^*) + w(P_j^*)$.

Proof. From Lemma 4.6 we know that if $w(P_I) < w(P_I^*)$, P_I is a regular 3-bin, $P_I^* = \{x, z\}$ where x is an item of type X_1 and $\alpha_I < \frac{13}{4}$. Thus $P_j = \{x\}$, $s(x) \leqq \alpha_I - 1 \leqq \frac{9}{4}$. Since $s(x) > \frac{7}{5}\alpha_j - 1$ we must have $\alpha_j < \frac{65}{28}$ and $|P_j^*| = 2$. Let $P_j^* = \{a, b\}$, $s(a) \geqq s(b)$. If a were of type X_1, $\alpha_j > \frac{9}{5} + 1 = \frac{14}{5} > \frac{65}{28}$. Hence, for each of a, b, and z, $w(*) \leqq s(*) - \frac{1}{10}$. If any of them satisfy $w(*) = s(*) - \frac{1}{5}$, then

$$w(P_I) + w(P_j) > (\tfrac{7}{5})(\alpha_I + \alpha_J) - 2 - \tfrac{3}{10},$$

$$w(P_I^*) + w(P_j^*) \leqq \alpha_I + \alpha_j - \tfrac{2}{5}.$$

If $w(P_I) + w(P_j) < w(P_I^*) + w(P_j^*)$, then

$$(\tfrac{2}{5})(\alpha_I + \alpha_j) < \tfrac{19}{10}$$

and

$$\alpha_I + \alpha_j < \tfrac{19}{4}.$$

Thus $\alpha_I < \frac{19}{4} - 2 = \frac{11}{4} < \frac{14}{5}$. But $\alpha_I \geqq \frac{9}{5} + 1 = \frac{14}{5}$, and we conclude that none of a, b, and z satisfies $w(*) = s(*) - \frac{1}{5}$ (hence all satisfy $w(*) = s(*) - \frac{1}{10}$).

Let P_k be the first regular 3-bin. Then unless one of a, b, z is placed before P_k, $P_k \cup P_j$ would dominate $P_I^* \cup P_j^*$ since P_k contains the three largest available items. Thus at least one of these three must precede the regular 3-bins. If $s(a) + s(b) + s(z) \geqq \frac{16}{5}$, then $\alpha_I + \alpha_j \geqq 5$ by Lemma 3.3 and $w(P_I) + w(P_j) \geqq \beta_I + \beta_j - 2 - \frac{3}{10} \geqq \alpha_I + \alpha_j - \frac{3}{10} \geqq w(P_I^*) + w(P_j^*)$. We see that the maximum size for the item preceding the regular 3-bins is $\frac{6}{5}$. If such an item were a fallback item, its weight would be $\frac{1}{5}$ less than its size. Therefore it is a regular item. The only remaining possibilities are that the item be in a regular 2-bin or a fallback 2-bin since if the largest item placed in a bin has size $\leqq \frac{6}{5}$, the bin cannot be a regular 1-bin, or a fallback 1-bin since the first item packed in a fallback 1-bin must be larger than half the bin size.

Suppose the bin, P_i, containing this item is a fallback 2-bin. Then since a third item of size $\leqq \frac{6}{5}$ would not fit, while one of size $\geqq 1$ does fit, $\frac{7}{5}\alpha_i - s(P_i) < \frac{1}{5}$ and $s(P_i) - s(P_i^*) > \frac{3}{5}$, causing each item in P_i to have weight $\frac{1}{5}$ less than its size, a contradiction.

The last possibility is for P_i to be regular 2-bin. If P_i^* contains no X_1-type item, the items in P_i have weight $\frac{1}{5}$ less than their size. Thus P_i^* must contain an X_1-type item $\alpha_i > \frac{9}{5} + 1 = \frac{14}{5}$ and $\beta_i \geqq \frac{98}{25}$. If both items in P_i are less than $\frac{6}{5}$ in size, $s(P_i)$ would be less than $\beta_i - 1$. Thus P_i can contain only one such item and hence there can be at most one such 2-bin, namely the last regular 2-bin preceding P_k, since all later bins have size at least $\frac{98}{25}$ and $3(\frac{6}{5}) < \frac{98}{25}$. □

The proof of the main result is now easy.

THEOREM 4.1. FFD $(\beta_1 = \frac{7}{5}\alpha_1, L)$ = succeed for any list L.

Proof. If the theorem fails we may choose L to be a minimal counterexample. Using the weight function w, and grouping by FFD bins,

$$w(L) = \sum_{i=1}^{M} w(P_I) + 1 - \frac{1}{10}$$

since the last item has size 1 and weight $1 - \frac{1}{10}$. Using the optimal bins,

$$w(L) = \sum_{i=1}^{M} w(P_i^*).$$

For the exceptional bin P_I mentioned in Lemma 4.7,

$$w(P_I) = s(P_I) - \tfrac{3}{10}, \quad \text{since } P_I \text{ is a regular 3-bin,}$$

and

$$w(P_I^*) \leqq s(P_I^*) - \tfrac{1}{10}, \quad \text{since } P_I^* \text{ contains exactly one } X_1 \text{ item.}$$

Since $s(P_I) \geqq s(P_I^*)$ by Lemma 3.5 we have

$$w(P_I) \geqq w(P_I^*) - \tfrac{1}{5}.$$

For the remaining bins, $w(\bigcup_{i \neq I} P_i) \geqq w(\bigcup_{i \neq I} P_i^*)$. Thus

$$w(L) = \sum_{i=1}^{M} w(P_i) + 1 - \tfrac{1}{10} \geqq \sum_{i=1}^{M} w(P_i^*) - \tfrac{1}{5} + 1 - \tfrac{1}{10} > w(L),$$

and we have a contradiction. □

5. Remarks. In the previous section we have shown that our version of MULTIFIT for uniform processors will produce a schedule whose length is at most $\frac{7}{5}$ times the optimum. In the arguments used, enough slack exists for the bound to be tightened. However, the difference is very slight and even a bound of 1.39 would entail significantly more work. The worst example we know of, Example 1, is a little more than 1.341 times optimal. (The interested reader may note that Example 1 can be modified to show that $R_0 \geqq$ any r such that $2r^3 + 4r^2 - 3r - 8 = 0$.) Thus we can only conclude that $1.341 < R_0 < 1.4$.

At this point, it may be appropriate to mention a few other implementation considerations. We assume our bins are sorted in nondecreasing order initially. In terms of worst-case performance this is preferable to the reverse ordering. As Example 2 shows, sorting the bins in nonincreasing order allows instances for which the FFD algorithm fails for expansion factors up to 1.5, while for our ordering, Theorem 4.1 insures that $R_0 \leqq 1.4$.

Example 1. FFD *packing with nondecreasing bin size.* $N = 12$, $M = 6$. Item sizes are 1.683, 1.683, 1.683, 1.299, 1.299, 1, 1, 1, 1, 1, 1, 1. Optimal bin sizes are 2, 2, 2, 2.683, 2.982, 2.982.

For R in [1, 1.341], FFD fails. See Figs. 1(a) and (b).

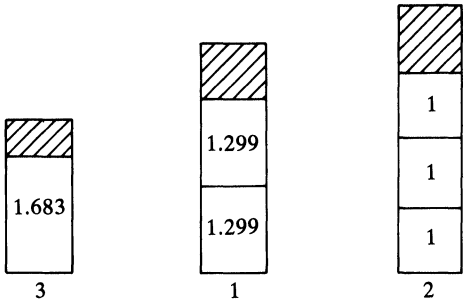


FIG. 1(a). *Attempted* FFD *packing with* R *in* [1, 1.341]. *Item of size* 1 *left over.*

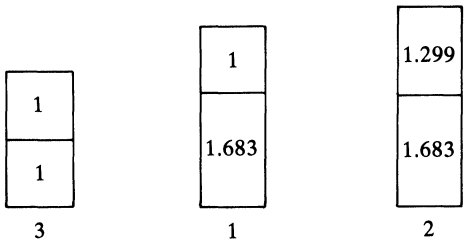


FIG. 1(b). *Optimal packing.*

Example 2. FFD *packing with nonincreasing binsize.* $N = M + 1$. Item sizes are $2, 2-\varepsilon, 2-2\varepsilon, 2-3\varepsilon, \cdots, 1+2\varepsilon, 1+\varepsilon, 1, 1$. Bin sizes are $2, 2, 2-\varepsilon, 2-2\varepsilon, \cdots, 1+2\varepsilon, 1+\varepsilon$.

FFD fails for R in

$$\left[1, \min\left\{\frac{3}{2}, \frac{3-\varepsilon}{2}, \frac{3-2\varepsilon}{2-\varepsilon}, \cdots, \frac{2+\varepsilon}{1+2\varepsilon}, \frac{2}{1+\varepsilon}\right\} = \frac{3-\varepsilon}{2} \text{ which approaches } \frac{3}{2} \text{ as } \varepsilon \to 0\right).$$

See Figs. 2(a) and (b).

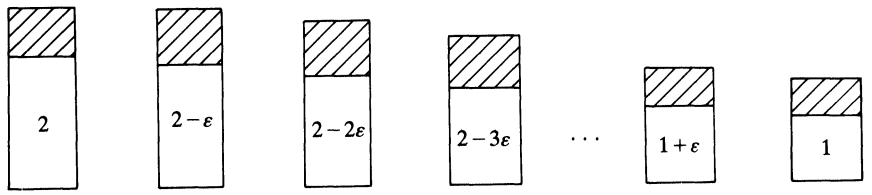


FIG. 2(a). *Attempted* FFD *packing with* R *in* [1, 1.5). *Item of size* 1 *left over.*

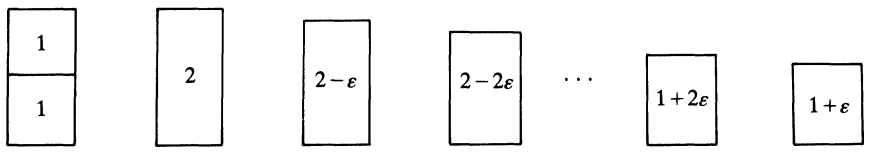


FIG. 2(b). *Optimal packing.*

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.

[Fr] D. K. FRIESEN, *Tighter bounds for the MULTIFIT processor scheduling algorithm*, this Journal, to appear.

[GIS] T. GONZALES, O. H. IBARRA AND S. SAHNI, *Bounds for LPT schedules on uniform processors*, this Journal, 6 (1977), p. 155–166.

[GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[Gr] R. L. GRAHAM, *Bounds on multiprocessor timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[HS] R. HOROWITZ AND S. SAHNI, *Exact and approximate algorithms for scheduling non-identical processors*, J. Assoc. Comput. Mach., 23 (1976), pp. 317–327.

[Jo] D. S. JOHNSON, *Fast algorithms for bin packing*, J. Comput. System Sci., 8 (1974), pp. 272–314.

[LL] J. W. S. LIU AND C. L. LIU, *Bounds on scheduling algorithms for heterogeneous computing systems*, Proceedings of the 1974 IFIP Congress, 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. 12, No. 1, February 1983

0097-5397/83/1201-0005 $01.25/0

MINIMUM *s-t* CUT OF A PLANAR UNDIRECTED NETWORK IN $O(n \log^2(n))$ TIME*

JOHN H. REIF†

Abstract. Let N be a planar undirected network with distinguished vertices s, t, a total of n vertices, and each edge labeled with a positive real (the edge's cost) from a set L. This paper presents an algorithm for computing a minimum (cost) s-t cut of N. For general L, this algorithm runs in time $O(n \log^2(n))$. For the case when L contains only integers $\leqq n^{O(1)}$, the algorithm runs in time $O(n \log(n) \log\log(n))$. Our algorithm also constructs a minimum s-t cut of a planar graph (i.e., for the case $L = \{1\}$) in time $O(n \log(n))$. Our algorithm can also be used to compute a minimum cut for a general undirected planar network.

The fastest previous algorithm for computing a minimum s-t cut of a planar undirected network (Itai and Shiloach [SIAM J. Comput., 8 (1979), pp. 135–150]) has time $O(n^2 \log(n))$; the s-t cut is a byproduct of the maximum flow computed by their algorithm. The best previous time bound for minimum s-t cut of a planar graph (Cheston, Probert and Saxton [report, Dept. Computer Science, Univ. Saskatchewan, 1977]) was $O(n^2)$.

Key words. planar, network, minimum s-t cut, graph algorithm

1. Introduction.

The importance of computing a minimum s-t cut of a network is illustrated by Ford and Fulkerson's [6], [7] theorem which states that the value of the minimum s-t flow of a network is precisely the minium s-t cut. The best known algorithm (Sleator [12] and Sleator and Tarjan [13]) for computing the maximum s-t flow or minimum s-t cut of a *sparse directed or undirected network* (with n vertices and $O(n)$ edges) has time[1] $O(n^2 \log(n))$. This paper is concerned with a *planar undirected network* N, which occurs in many practical applications.

Ford and Fulkerson [6], [7] have an elegant maximum s-t flow algorithm for the case N is (s, t)-*planar* (both s and t are on the same face) which when efficiently implemented by priority queues as described in Itai and Shiloach [9] has time $O(n \log(n))$. Moreover, $O(n)$ executions of their algorithm suffice to compute the maximum flow of a general *planar network* in total time $O(n^2 \log(n))$. Also, Cheston, Probert and Saxton [3] have an $O(n^2)$ algorithm for the minimum s-t cut of a planar graph and Shiloach [9] gives an $O(n \log(n))^2$ algorithm for the minimum cut of a planar graph.

Let $Q_L(n)$ be the asymptotic time complexity to maintain a priority queue of $O(n)$ elements with costs from a set L of nonnegative reals, and with $O(n)$ insertions and deletions. For the general case, $Q_L(n) = O(n \log(n))$ as described in Aho, Hopcroft and Ullman [1]. For the special case when L is a set of positive integers $\leqq n^{O(1)}$, Boas, Kaas and Zijlstra [2] show $Q_L(n) = O(n \log\log(n))$. It is obvious that if L is of constant cardinality then $Q_L(n) = O(n)$.

A key element of the Ford and Fulkerson [6], [7] algorithm for (s, t)-planar networks was an efficient reduction to finding a minimum cost path between two vertices in a sparse network. Dijkstra [4] gives an algorithm for a generalization of this problem (to find a minimum cost path from a fixed "source" vertex s to each other vertex). Dijkstra's algorithm may be implemented (see Aho, Hopcroft and

* Received by the editors February 27, 1981, and in revised form February 1, 1982. This work was supported in part by the National Science Foundation under grant NSF-MCS79-21024 and the Office of Naval Research under contract N00014-80-C-0647.

† Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138.

[1] We assume throughout this paper that our machine model is a unit cost criteria RAM (see Aho, Hopcroft and Ullman [1]).

Ullman [1]) in time $O(Q_L(n))$ for a sparse network with n vertices, and L is the set of nonnegative reals labeling the edges.

Our algorithm for computing the minimum s-t cut of a planar undirected network has time $O(Q_L(n)\log(n))$. This algorithm also utilizes an efficient reduction to minimum cost path problems. Our fundamental innovation is a "divide and conquer" approach for cuts on the plane.

The paper is organized as follows: The next section gives preliminary definitions of graphs, networks, minimum cuts, maximum flows, and duals of planar networks. Section 3 gives the Ford–Fulkerson algorithm for (s, t)-planar graphs. Section 4 describes briefly an efficient algorithm due to Itai and Shiloach [9] for finding a minimum cut intersecting a given face of the primal network. Our divide and conquer approach is described and proved in § 5. Section 6 presents our algorithm for minimum s-t cuts of planar networks. Finally, § 7 concludes the paper.

2. Preliminary definitions

2.1. Graphs. Let a *graph* $G=(V,E)$ consist of a *vertex set* V and a collection of *edges* E. Each edge $e\in E$ *connects* two vertices $u, v\in V$ (edge e is a *loop* if it connects identical vertices). We let $e=\{u, v\}$ denote edge e connects u and v. Edges e, e' are *multiple* if they have the same endpoints. Let a *path* be a sequence of edges $p=e_1,\cdots,e_k$ such that $e_i=\{v_{i-1}, v_i\}$ for $i=1,\cdots,k$ (we say p *traverses* vertices $v_0,\cdots,v_k$). Let p be a *cycle* if $v_0=v_k$ (cycles containing the same edges are considered identical). A path p' is a *subpath* of p if p' is a subsequence of p. Let G be a *standard graph* if G has neither multiple edges nor loops and is triconnected. Generally we let $n=|V|$ be the number of vertices of graph G. If G is planar, then by Euler's formula G contains at most $6n-12$ edges.

2.2. Networks. Let an *undirected network* $N=(G, c)$ consist of an undirected graph $G=(V,E)$ and a mapping c from E to the positive reals. For each edge $e\in v$, $c(e)$ is the *cost* of e. For any edge set $E'\subseteq E$, let $c(E')=\sum_{e\in E'} c(e)$. Let the *cost* of path $p=e_1,\cdots,e_k$ be $c(p)=\sum_{i=1}^{k} c(e_i)$. Let a path p from vertex u to vertex v be *minimum* if $c(p)\leqq c(p')$ for all paths p' from u to v. Let $N=(G, c, s, t)$ be a *standard network* if (G, c) is an undirected network, with $G=(V,E)$ a standard graph, and s, t are distinguished vertices of V (the *source*, *sink*, respectively). Note that triconnectivity can easily be achieved by adding $O(n)$ edges with cost 0.

2.3. Minimum cuts and maximum flows in networks. Let $N=(G, c, s, t)$ be a standard network with $G=(V,E)$. An edge set $X\subseteq E$ is an s-t *cut* if $(V, E-X)$ has no paths from s to t. Let s-t cut X be *minimum* if $c(X)\leqq c(X')$ for each s-t cut X'. See Fig. 1.

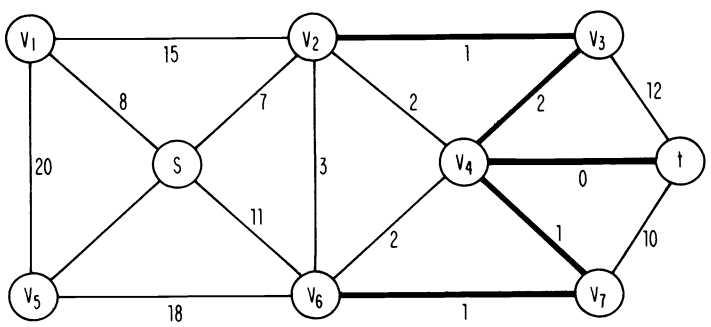


FIG. 1. *A network N with source s and sink t. The heavily drawn edges indicate a minimum s-t cut* $\{\{v_2, v_3\}, \{v_3, v_4\}, \{v_4, t\}, \{v_4, v_7\}, \{v_6, v_7\}\}$ *with cost* 5.

Let A be the set of directed edges $\{(u, v) | \{u, v\} \in E\}$. A function f mapping A to the nonnegative reals is a *flow* if

(i) For all $e \in A$, $f(e) \leqq c(e)$, and

(ii) For all $v \in V$, if $v \notin \{s, t\}$ then IN (f, v) = OUT (f, v), where

$$\text{IN}(f, v) = \sum_{(u,v)\in A} f(u, v) \quad \text{and} \quad \text{OUT}(f, v) = \sum_{(v,u)\in A} f(v, u).$$

The *value* of the flow f is OUT (f, s) − IN (f, t). The following motivates our work on minimum s-t cuts:

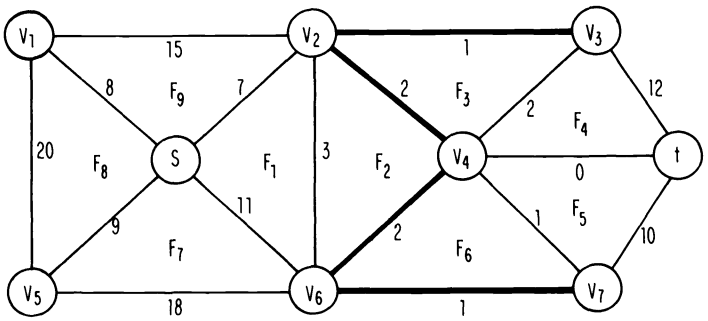
THEOREM 1 (Ford and Fulkerson [7]). *The maximum value of any flow is the cost of a minimum s-t cut.*

2.4. Planar networks and duals. Let $G = (V, E)$ be a planar standard graph, with a fixed embedding on the plane. G partitions the plane into connected regions. Each connected region is called a *face* and has a corresponding cycle of edges which it borders. For each edge $e \in E$, let $D(e)$ be the corresponding *dual edge* connecting the two faces bordering e. Let $D(G) = (\mathcal{F}, D(E))$ be the *dual graph* of G, with vertex set $\mathcal{F}$ = the faces of G, and with edge set $D(E) = \bigcup_{e\in E} D(e)$. Note that the dual graph is not necessarily standard (i.e., it may contain multiple edges and loops), but is planar. Let a cycle q of $D(G)$ be a *cut-cycle* if the region bounded by q contains exactly one of s or t. Note that a cycle is a cut-cycle independent of the way in which the dual graph is embedded in the plane, although a particular embedding may change which of s or t the cycle contains. See Figs. 1 and 2. The following proposition is trivial to derive:

PROPOSITION 1. *D induces a 1–1 correspondence between the s-t cuts of G and the cut-cycles of $D(G)$.*

Let $N = (G, c, s, t)$ be a *planar* standard network, with $G = (V, E)$ planar. Let the *dual network* $D(N) = (D(G), D(c))$ have edge costs $D(c)$, where the edge cost of each dual edge $D(e)$ is the cost of the original edge $e \in E$. (Generally we will use just c in place of $D(c)$ where no confusion will result.) See Fig. 3. For each face $F_i \in \mathcal{F}$, let a cut-cycle q in $D(N)$ be *F_i-minimum* if q contains F_i on (rather than inside) the cycle q and $c(q) \leqq c(q')$ for all cut-cycles q' containing F_i. The next proposition is easy but tedious to prove.

PROPOSITION 2. *A minimum s-t cut has the same cost as a minimum cost cut-cycle of $D(G)$.*



N

FIG. 2. *The same planar network N as in Fig. 1, with faces $F_1, \cdots, F_{10}$, and with a nonminimal s-t cut $X = \{\{v_2, v_3\}, \{v_2, v_4\}, \{v_4, v_6\}, \{v_6, v_7\}\}$ of cost 6, indicated by heavily drawn edges.*

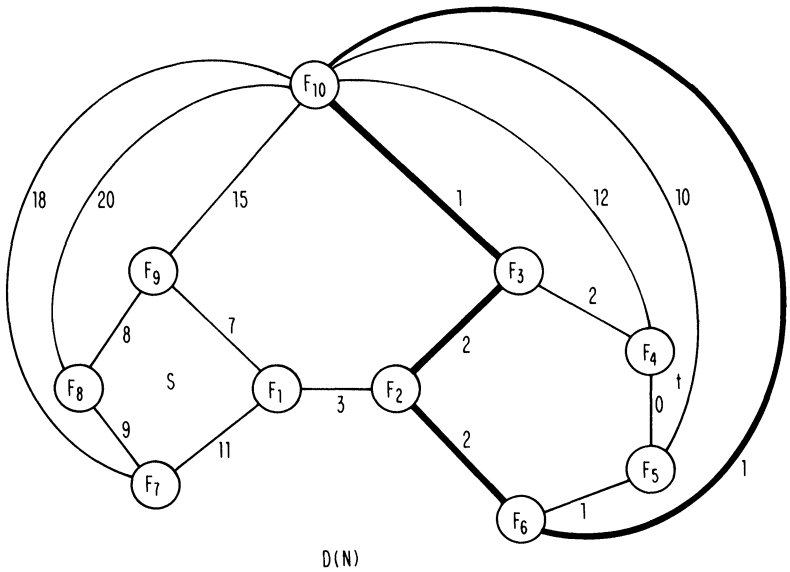


FIG. 3. *The dual network $D(N)$ derived from the planar network N of Figs. 1 and 2. The heavily drawn edges give an F_2-minimum cut cycle $D(X)=\{\{F_{10}, F_3\}, \{F_3, F_2\}, \{F_2, F_6\}, \{F_6, F_{10}\}\}$ which is the dual of the s-t cut X given in Fig. 2.*

3. Ford and Fulkerson's minimum *s-t* cut algorithm for (s, t)-planar networks. Let $N=(G, c, s, t)$ be a planar standard network. G (as well as N) is (s, t)-planar if there exists a face F_0 containing both s and t. Let planar network N' be derived from N by adding on edge e_0 connecting s and t with cost ∞. Let e_0 be embedded onto a line segment from s to t in F_0, which separates F_0 into two new faces F_1 and F_2. Ford and Fulkerson [6] have the following elegant characterization of the minimum s-t cut of (s, t)-planar network N.

THEOREM 2. *There is a 1–1 correspondence between the s-t cuts of N and the paths of $D(N')$ from F_2 to F_1 and avoiding $D(e_0)$. Furthermore, this correspondence preserves edge costs. Therefore, the minimum s-t cuts of N correspond to the minimum cost paths in $D(N')$ from F_2 to F_1.*

By use of Dijkstra's [4] shortest path algorithm, we have:

COROLLARY 2. *A minimum cut of (s, t)-planar network N with n vertices may be computed in time $O(Q_L)(n))$, where $L=$ range (c).*

Note that applications of this corollary include the $O(n \log (n))$ time minimum s-t cut algorithm of Itai and Shiloach [9] for (s, t)-planar undirected networks, and the $O(n)$ time minimum s-t cut algorithm of Cheston, Probert and Saxton [3] for (s, t)-planar graphs.

4. An efficient algorithm for *F*-minimum cut cycles. Let $N=(G, c, s, t)$ be a planar standard network, with $G=(V, E)$ and $L=$ range (c). Our algorithm for minimum s-t cuts will require efficient construction of an F-minimum cut-cycle for a given face F. For completeness, we very briefly describe here an algorithm for this, due to Itai and Shiloach [9].

Let $\mathcal{F}_s$ be the set of faces bordering s and let $\mathcal{F}_t$ be the faces bordering t. Let a $\mu(s, t)$ *path* be a minimum cost path in $D(N)$ from a face of $\mathcal{F}_s$ to a face of $\mathcal{F}_t$.

PROPOSITION 3 (Itai and Shiloach [9]). *Let μ be a $\mu(s, t)$ path traversing faces $F_1, \cdots, F_d$. Let $D(X_i)$ be a F_i-minimum cut-cycle of $D(N)$ for $i=1, \cdots, d$. Then X_{i_0} is a minimum s-t cut of N, where $c(X_{i_0})=\min\{c(X_i) | i=1, \cdots, d\}$.*

To compute a $\mu(s, t)$ path in time $O(Q_L(n))$, let M be the planar network derived from $D(N)$ by adding new vertices v_s, v_t and an edge connecting v_s to each face in $\mathcal{F}_s$ and an edge connecting each face in $\mathcal{F}_t$ to v_t. Let the cost of each of these edges be 1. Let p be a minimum cost path in M from v_s to v_t. Then p, less its first and last edges, is a $\mu(s, t)$ path. See Fig. 4.

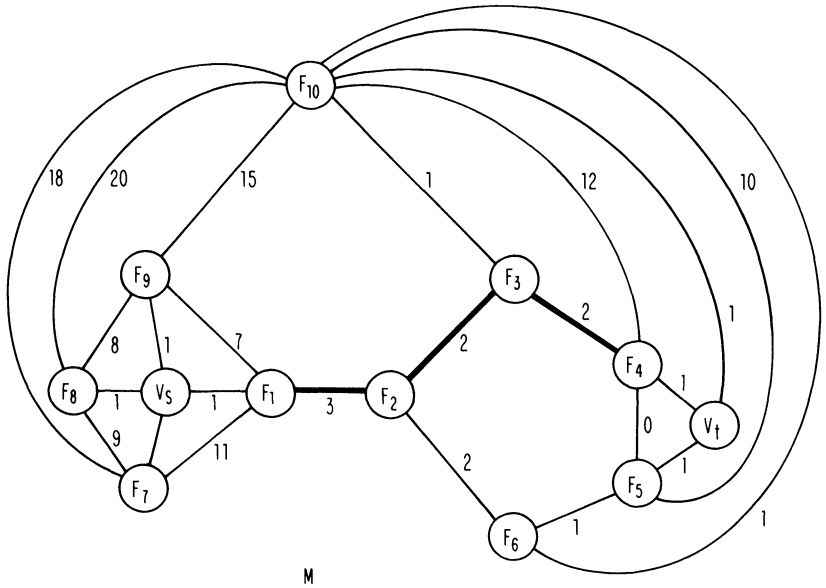


FIG. 4. *Network M derived from the dual network $D(N)$ given in Fig. 3. The heavily drawn edges are the $\mu(s, t)$-paths.*

Let μ be a $\mu(s, t)$ path in $D(N)$ traversing faces $F_1, \cdots, F_d$. By viewing μ as a horizontal line segment with s on the left and t on the right for each edge $D(e)$ of $D(N)$ which is not in $\mu(s, t)$ but is connected to a face F_i, $D(e)$ may be considered to be connected to F_i from *below* or *above* (or both). Let μ' be a copy of μ traversing new vertices $x_1, \cdots, x_d$. Let D' be the network derived from $D(N)$ by reconnecting to x_i each edge entering F_i from above. See Fig. 5. If p is a path of D', then a corresponding path $\hat{p}$ in $D(N)$ is constructed by replacing each edge and face appearing in μ' with the corresponding edge or face of μ. Clearly, $c(p) = c(\hat{p})$.

THEOREM 3 (Itai and Shiloach [9]). *If p is a minimum cost path connecting F_i and x_i in D', then $\hat{p}$ is an F_i-minimum cut-cycle of $D(N)$.*

By applying Corollary 2 to Theorem 3 we have:

COROLLARY 3. *This is an $O(Q_L(n))$ time algorithm to compute an F_i-minimum cut-cycle for any face F_i of a $\mu(s, t)$ path in $D(N)$.*

Note that for restricted L this may be more efficient than the $O(n \log n)$ upper bound given by Itai and Shiloach [9]; for example this gives an $O(n)$ time algorithm for an F_i-minimum cut-cycle of a planar graph.

5. A divide and conquer approach. Let μ be a $\mu(s, t)$ path of $D(N)$ traversing faces $F_1, \cdots, F_d$ as in § 4. Note that any s-t cut of planar network N must contain an edge bounding on a face in $\{F_1, \cdots, F_d\}$. The algorithm of Itai and Shiloach [9] for computing a minimum s-t cut of N is to construct an F_i-minimum cut-cycle $D(X_i)$ in $D(N)$ for each $i = 1, \cdots, d$. This may be done by $d = O(n)$ executions of the $O(Q_L(n))$ time algorithm of Corollary 3. Then by Proposition 3, X_{i_0} is a minimum s-t

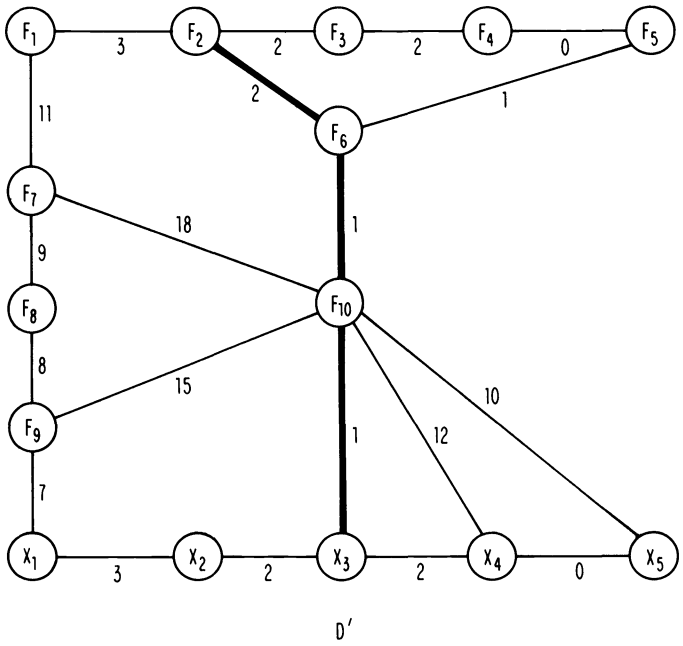


FIG. 5. *Network D' derived from dual network $D(N)$ of Fig. 3 using the $\mu(s, t)$-path of Fig. 4. The heavily drawn edges give the F_2-minimum cut-cycle $D(X)$ of Fig. 3.*

cut where $c(X_{1_0}) = \min\{c(X_1), \cdots, c(X_d)\}$. In the worst case, this requires $O(Q_L(n) \cdot n)$ total time. This section presents a divide and conquer approach which utilizes recursive executions of an F_i-minimum cut algorithm.

LEMMA 1. *Let F_i, F_j be distinct faces of μ, with $i < j$. Let p be any F_j-minimum cut-cycle of $D(N)$ such that the closed region R bounded by p contains s. Then there exists an F_i-minimum cut-cycle q contained entirely in R. (See Fig. 6.)*

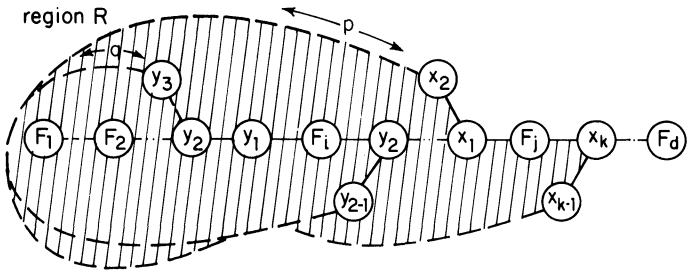


FIG. 6. *$F_1, F_2, \cdots, F_d$ is a $\mu(s, t)$ path in $D(N)$. $p = (F_j, x_1, x_2, \cdots, x_k)$ is a F_j-minimum cut-cycle enclosing region R. The F_i-minimum cut-cycle $q = (F_i, y_1, y_2, \cdots, y_l)$ is contained in R.*

Proof. Let q be any F_i-minimum cut-cycle. Let q' be the cut-cycle derived from q by repeatedly replacing subpaths of q connecting faces traversed by μ with the appropriate subpaths of μ (only apply replacements for which the resulting q' is a cut-cycle). Observe $c(q') \leqq c(q)$ (else we can show μ is not a $\mu(s, t)$ path). Let R' be the closed region bounded by q'. Suppose $R' \not\subset R$. Then there must be a subpath q_1 of q' connecting faces F^a, F^b of p such that q_1 only intersects R' at F^a and F^b. Let p_1 be the subpath of p connecting F^a and F^b in R'. We claim $c(p_1) \leqq c(q_1)$. Suppose $c(p_1) > c(q_1)$. By our construction of q', either q_1 avoids F_j, $F_j = F^a$ or $F_j = F^b$. In any

case, we may derive a cut-cycle p' from p by substituting q_1 for p_1. But this implies $c(p')<c(p)$, contradicting our assumption that p is an F_i-minimum cut-cycle. Now substitute p_1 for q_1 in q'. The resulting cut-cycle is no more costly than q', since $c(p_1)\leqq C(q_1)$. See Fig. 7. The lemma follows by repeated application of this process. □

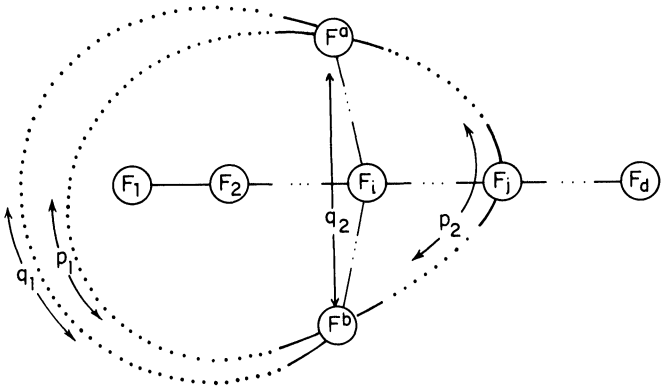


FIG. 7. *$F_1, F_2, \cdots, F_d$ is a $\mu(s,t)$-path, $p = p_1 \cdot p_2$ is a cut-cycle containing F_j. $q = q_1 \cdot q_2$ is a cut-cycle containing F_i. If $c(q_1)<c(p_1)$, then $p' = q_1 \cdot q_2$ is a cut-cycle containing F_i and with cost $c(p')<c(p)$.*

The above lemma implies a method for dividing the planar standard network N, given an s-t cut X. The network derived from N by deleting all edges of X can be partitioned into two networks N^s, N^t, where no vertex of N^s has a path to t, and no vertex of N^t has a path to s. Also, each edge $e \in X$ must have connections to a vertex of N^s and a vertex of N^t.

Let $N_0 = \text{DIVIDE}(N, X, s)$ be the standard planar network consisting of N^s,

(i) with a new vertex t_0 and

(ii) a new edge $\{u, t_0\}$ with cost $c(\{u, v\})$, for each edge $\{u, v\} \in X$ such that u is a vertex of N^s and v is a vertex of N^t;

(iii) finally (to insure N_0 is standard) merging multiple edges and setting the cost of each resulting edge to be the sum of the costs of the multiple edges from which it was derived. See Figs. 8 and 9.

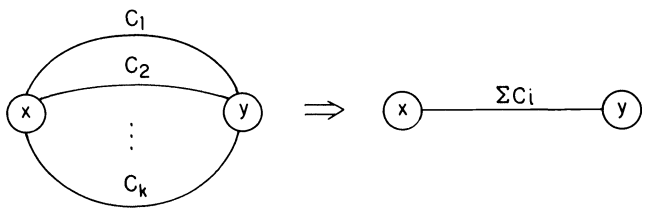


Fig. 8. *The merging into a single edge of multiple edges connected to vertex x and vertex y.*

Similarly, let $N_1 = \text{DIVIDE}(N, X, t)$ be the standard planar network consisting of N^t,

(i) with a new vertex s_1, and

(ii) for new edge $\{s_1, v\}$ with cost $c(\{u, v\})$, for each edge $\{u, v\} \in X$ such that u is a vertex of N^s and v is a vertex of N^t, and finally applying step (iii) above. See Fig. 9.

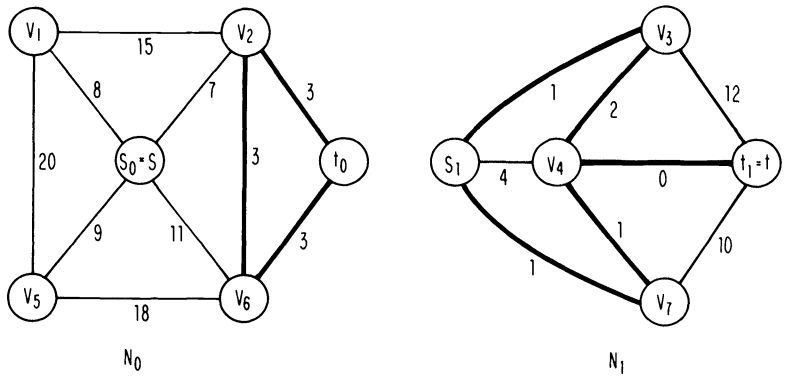


FIG. 9. *The networks* $N_0 = \text{DIVIDE}(N, X, s)$ *and* $N_1 = \text{DIVIDE}(N, X, t)$ *derived from the network* N *and* s-t *cut* X *given in Fig. 2.* N_0 *and* N_1 *will be further subdivided by the cuts* X_0, X_1 *respectively, indicated by heavily drawn edges.*

Let E be the set of edges of network N and let Y be a subset of the edges of $N_0 = \text{DIVIDE}(N, X, s)$ or of $N_1 = \text{DIVIDE}(N, X, t)$. Then let $E(Y)$ be the set of edges of E that were mapped into edges of Y when N_0 or N_1 was created. The next theorem follows immediately from the above Lemma 1 and Proposition 3.

THEOREM 4. *Let* X *be an* s-t *cut of a planar standard network* N *such that* $D(X)$ *is an* F-*minimum cut-cycle, for some face* F *in a* $\mu(s, t)$ *path of* $D(N)$. *Let* X_0 *be a minimum* s-t_0 *cut of* $N_0 = \text{DIVIDE}(N, X, s)$ *and let* X_1 *be a minimum* s_1-t *cut of* $N_1 = \text{DIVIDE}(N, X, t)$. *Then* $E(X_0)$ *or* $E(X_1)$ *is a minimum* s-t *cut of* N.

6. The minimum *s-t* cut algorithm for planar networks. Theorem 4 yields a very simple but efficient divide and conquer algorithm for computing minimum s-t cut of a planar standard network. We assume the Ford and Fulkerson [6] algorithm given in § 3:

(i) (s, t)-PLANAR-MIN-CUT(N) which computes a minimum s-t of (s, t)-planar standard network N in time $O(Q_L(n))$.

We also assume algorithms (given in § 4):

(ii) $\mu(s, t)$ PATH($D(N)$) computes a $\mu(s, t)$ path of $D(N)$ in time $O(Q_L(n))$.

(iii) F-MIN-CUT(N, F_i, μ) computes q, where $D(q)$ is an F_i-minimum cycle of N (for any F_i in $\mu(s, t)$ path μ), in time $O(Q_L(n))$.

RECURSIVE ALGORITHM PLANAR-MIN-CUT(N, μ).
input planar standard network $N = (G, c, s, t)$, where $G = (V, E)$, and $\mu(s, t)$ path μ.
begin
Let $F_1, \cdots, F_d$ be the faces traversed by μ.
if $d = 1$ **then return** (s, t)-PLANAR-MIN-CUT (N);
else begin
$X \leftarrow F$-MIN-CUT $(N, F_{\lfloor d/2 \rfloor}, \mu)$
$N_0 \leftarrow \text{DIVIDE}(N, X, s)$; $N_1 \leftarrow \text{DIVIDE}(N, X, t)$;
Let μ_0 and μ_1 be the subpaths of μ contained in N_0
and N_1, respectively
$X_1 \leftarrow$ PLANAR-MIN-CUT (N_1, μ_1); $X_0 \leftarrow$ PLANAR-MIN-CUT (N_0, μ_0)
if $c(E(X_0)) \leqq c(E(X_1))$ **then return** $E(X_0)$ **else return** $E(X_1)$;
end;
end

Associated with this recursive algorithm we define a *call tree* T whose root is N and whose descendants are the networks input to the algorithm on recursive calls. Let d be the number of faces traversed by μ, the $\mu(s, t)$ path of N. If $d = 1$ then root N has no children. Otherwise, N has left child N_0 and right child N_1, as computed in the algorithm, and so on.

For any $\omega \in \{0, 1\}^*$ inductively let $N_\omega = (G_\omega, c_\omega, s_\omega, t_\omega)$ be the planar standard network and let μ_ω be the $\mu(s_\omega, t_\omega)$ path in N_ω defined by some recursive calls to PLANAR-MIN-CUT. Suppose PLANAR-MIN-CUT (N_ω, μ_ω) is called. If μ_ω contains only one face, then let $N_{\omega 0}$ and $N_{\omega 1}$ be empty networks, and let $\mu_{\omega 0}$ and $\mu_{\omega 1}$ be empty paths. Else let X_ω be the set s_ω-t_ω cut of N_ω computed by the call to F-MIN-CUT$(\cdot)$, let $N_{\omega 0}$, $N_{\omega 1}$ be the planar standard networks constructed by the calls to DIVIDE, and let $\mu_{\omega 0}$, $\mu_{\omega 1}$ be the subsets of μ contained in $N_{\omega 0}$, $N_{\omega 1}$. Then it is easy to verify that $\mu_{\omega 0}$ is a $\mu(s_{\omega 0}, t_{\omega 0})$ path in $N_{\omega 0}$ and $\mu_{\omega 1}$ is a $\mu(s_{\omega 1}, t_{\omega 1})$ path in $N_{\omega 1}$, and the length of $\mu_{\omega 0}$ and the length of $\mu_{\omega 1}$ are each $\leqq \lceil \frac{1}{2} d_\omega \rceil$, where d_ω is the length of μ_ω. Hence there can be no more than $\lceil \log(d) \rceil$ mutually recursive calls, so the call tree T has depth at most $\lceil \log(d) \rceil \leqq \lceil \log(n) \rceil$, where n is the number of nodes in N.

Let m be the number of edges of N and let m_ω be the number of edges of N_ω. The following theorem provides an upper bound of $2m + 2^r$ on the number of edges of networks of depth r in the call tree T.

THEOREM 5. *For each* $r \geqq 0$, $\sum_{\omega \in \{0,1\}^r} m_\omega \leqq 2m + 2^r$.

Proof. Note that by definition of DIVIDE, the edges of $N_{\omega 0}$ or $N_{\omega 1}$ are derived from disjoint sets of edges of N_ω. Fix an edge e of N. Let e_ω be the edge (if it exists) of N_ω derived from a set of edges of N containing e. Let edge e *contribute* to N_ω if $e \neq \{s_\omega, t_\omega\}$ and let e *fully contribute* to N_ω if e_ω contains neither s_ω nor t_ω. For each $r \geqq 0$, let $B_r(e) = \{e_\omega | e_\omega \neq \{s_\omega, t_\omega\}$ and $\omega \in \{0, 1\}^r\}$. Thus $|B_r(e)|$ is the number of networks of depth r in T to which edge e contributes.

Let the strings of $\{0, 1\}^*$ be ordered lexicographically. We require a technical lemma.

LEMMA 2. $|B_r(e)| \leqq 2$, *and furthermore if* $B_r(e) = \{e_\omega, e_z\}$ *for* $\omega < z$, $z \in \{0, 1\}^r$, *then edge* e_ω *is connected to* t_ω *and edge* e_z *is connected to* s_z.

This lemma states that e contributes to at most two networks of depth r in T, and e fully contributes to no two distinct networks of depth r. For example, consider edge $e = \{v_2, v_3\}$ of network N given in Fig. 2. Edge e fully contributes to N. In Fig. 9, edge e contributes to N_0 by edge $e_0 = \{v_2, t_0\}$ and also contributes to N_1 by edge $e_1 = \{s_1, v_3\}$. Furthermore, in Fig. 10 edge e contributes to N_{00} by edge $e_{00} = \{v_2, t_{00}\}$ and in Fig. 11 edge e contributes to N_{11} by edge $e_{11} = \{s_{11}, v_3\}$ but e contributes to neither N_{01} nor N_{10}.

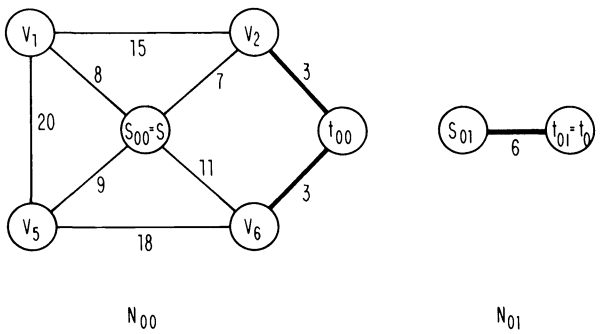


FIG. 10. *Networks* $N_{00} = \text{DIVIDE}(N_0, X_0, s_0)$ *and* $N_{01} = \text{DIVIDE}(N_0, X_0, t_0)$ *derived from network* N_0 *with* s-t_0 *cut* X_0 *of Fig.* 9.

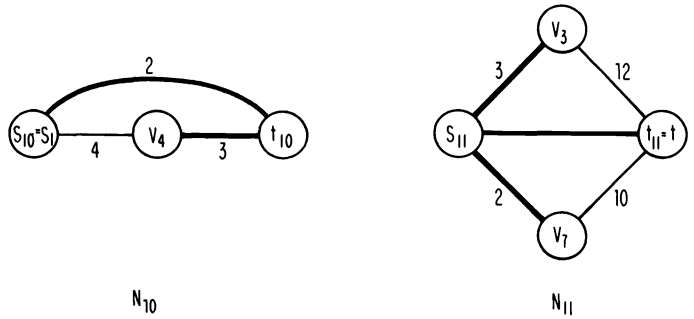


FIG. 11. *Networks* $N_{10} = \text{DIVIDE}(N_1, X_1, s_1)$ *and* $N_{11} = \text{DIVIDE}(N_1, X_1, t_1)$ *derived from network* N_1 *with* s_1*-*t *cut* X_1 *of Fig.* 9.

Proof of Lemma 2 *by induction.* Suppose for some fixed r_0, this lemma holds for all $r \leqq r_0$. If $B_{r_0}(e) = \varnothing$ then clearly $B_{r_0+1}(e) = \varnothing$. Suppose $1 \leqq |B_{r_0}(e)| \leqq 2$ and consider any $e_\omega \in B_{r_0}(e)$. If $e_\omega \notin X_\omega$ then by definition of DIVIDE, either $e_\omega = e_{\omega 0}$ appears in $N_{\omega 0}$ or $e_\omega = e_{\omega 1}$ appears in $N_{\omega 1}$, but not both. On the other hand, if $e_\omega \in X_\omega$, then $e_{\omega 0}$ appears in $N_{\omega 0}$ connected to $t_{\omega 0}$ and also $e_{\omega 1}$ appears in $N_{\omega 1}$ connected to $s_{\omega 1}$. In either case, if $|B_{r_0}(e)| = 1$, then $|B_{r_0+1}(e)| \leqq 2$. Otherwise suppose $|B_{r_0}(e)| = 2$ so there exists some $e_z \in B_{r_0}(e)$ with $\omega < z$. By the induction hypothesis, e_ω is connected to t_ω and e_z is connected to s_z. Thus for $j = 0, 1$ edge $e_{\omega j}$ (if it exists) is connected to $t_{\omega j}$ and edge e_{zj} (if it exists) is connected to s_{zj}. Hence if $e_\omega \in X_\omega$ then $e_{z1} = \{s_{z1}, t_{z1}\}$. In each case, $|B_{r_0+1}(e)| \leqq 2$. □

To complete the proof of Theorem 5, observe that $|\{\{s_\omega, t_\omega\} | \omega \in \{0, 1\}^r\}| = 2^r$. Hence

$$\sum_{\omega \in \{0,1\}^r} m_\omega \leqq \left(\sum_{e \in E} |B_r(e)| \right) + |\{\{s_\omega, t_\omega\} | \omega \in \{0, 1\}^r\}| \leqq 2m + 2^r$$

by Lemma 2. □

THEOREM 6. *Given a planar standard network* $N = (G, c, s, t)$ *with* $L = \text{range}(c)$, *and* μ *is a* $\mu(s, t)$ *path of* N *then* PLANAR-MIN-CUT (N, μ) *computes a minimum* s*-*t *cut of* N *in time* $O(Q_L(n) \log(n))$.

Proof. The total time cost is

$$\sum_{\substack{\omega \in \{0,1\}^r \\ 0 \leqq r \leqq \lceil \log(n) \rceil}} O(Q_L(m_\omega)) = \sum_{0 \leqq r \leqq \lceil \log(n) \rceil} O(Q_L(2m + 2^r)) \quad \text{by Theorem 5,}$$

$$= O(Q_L(n) \log(n)) \quad \text{since } 2m + 2^{\log(n)} = O(n). \qquad \square$$

By known upper bounds on the cost of maintaining queues (as discussed in the Introduction), we also have:

COROLLARY 4. *A minimum* s*-*t *cut of* N *is computed in time* $O(n \log^2(n))$ *for general* L (*i.e., a set of positive reals*), *in time* $O(n \log(n) \log\log(n))$ *for the case where* L *is a set of positive integers bounded by a polynomial in* n *and in time* $O(n \log(n))$ *for the case where* N *is a graph with identically weighted edges.*

7. Conclusion. We have presented a divide and conquer method for computing a minimum s-t cut of a planar undirected network which improves on the running time of the algorithm of Itai and Shiloach [9] by a factor of $n/\log n$. An additional attractive feature of this algorithm is its *simplicity*, as compared to other algorithms for computing minimum s-t cuts for sparse networks (Galil and Naamad [8], Shiloach [10] and Sleator and Tarjan [13]).

[1] A. AHO, J. HOPCROFT AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
[2] P. VAN EMDE BOAS, R. KAAS AND E. ZIJLSTRA, *Design and implementation of an efficient priority queue*, Math. Systems Theory, 10 (1977), pp. 99–127.
[3] G. CHESTON, R. PROBERT AND C. SAXTON, *Fast algorithms for determination of connectivity sets for planar graphs*, Dept. Computer Science, Univ. Saskatchewan, 1977.
[4] E. DIJKSTRA, *A note on two problems in connections with graphs*, Numer. Math., 1 (1959), pp. 269–271.
[5] S. EVEN AND R. TARJAN, *Network flow and testing graph connectivity*, this Journal, 4 (1975), pp. 507–518.
[6] C. FORD AND D. FULKERSON, *Maximal flow through a network*, Canad. J. Math., 8 (1956), pp. 399–404.
[7] ———, *Flows in Networks*, Princeton Univ. Press, Princeton, NJ, 1962.
[8] Z. GALIL AND A. NAAMAD, *Network flow and generalized path compression*, in Proc. of Symposium on Theory of Computing, Atlanta, Georgia, 1979.
[9] A. ITAI AND Y. SHILOACH, *Maximum flow in planar networks*, this Journal, 8 (1979), pp. 135–150.
[10] Y. SHILOACH, *An $O(nI \cdot \log^2 I)$ maximum-flow algorithm*, Computer Science Dept., Stanford Univ., Stanford, CA, 1978.
[11] ———, *A multi-terminal minimum cut algorithm for planar graphs*, this Journal, 9 (1980), pp. 214–219.
[12] D. SLEATOR, *An $O(nm \log n)$ algorithm for maximum network flow*, Ph.D. dissertation, Stanford Univ., Stanford, CA, 1980.
[13] D. SLEATOR AND R. TARJAN, *A data structure for dynamic trees*, 13th Annual ACM Symposium on Theory of Computing, 1981, pp. 114–122.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0006 $01.25/0

A TECHNIQUE FOR ESTABLISHING COMPLETENESS RESULTS IN THEOREM PROVING WITH EQUALITY*

GERALD E. PETERSON†

Abstract. It is proved that an automatic theorem proving system consisting of resolution, paramodulation, factoring, equality reversal, simplification, and subsumption removal is complete in first-order logic with equality. When restricted to equality units, the system is similar to the Knuth–Bendix procedure for deriving consequences from equalities. However, our proofs of completeness are restricted to the case in which the ordering on words (terms or atoms) that is required in this type of process is order-isomorphic to the positive integers. The completeness of resolution and paramodulation without the functionally reflexive axioms is a simple corollary of our result. The methods used are based upon the familiar ideas associated with semantic trees, and should be helpful in showing that other theorem proving systems with equality are complete.

Key words. theorem proving, equality, first-order logic, semantic tree, simplification, completeness, resolution, paramodulation, simplification ordering

1. Introduction. About fifteen years ago (see [3] for references) some quite efficient and useful general-purpose theorem proving systems based on resolution in first-order logic without equality were created and used to prove some theorems which had not previously been proved automatically. Unfortunately, these systems were not able to prove anything very complicated, and additions were sought which would make them more powerful. Since it is awkward to express ideas involving equality in a logical system without equality, various ways of incorporating equality into the system were tried [9], [26], [30], [34], [35].

For example, Robinson and Wos [32] introduced paramodulation in 1969 and proved that if the functionally reflexive axioms (axioms such as $f(x, y) = f(x, y)$) were added to the set of clauses, then resolution and paramodulation constituted a complete set of inference rules. Since that time, some effort [1], [2], [17], [18] has been expended in considering the so-called "functionally reflexive problem," that is, the problem of proving that resolution and paramodulation are complete without the functionally reflexive axioms. In fact, Brand [1] provided a solution to the functionally reflexive problem in 1975. An independent solution is contained in this paper as a corollary to Theorem 8.

Unrestricted paramodulation is a very weak inference rule because it produces mountains of irrelevant clauses which rapidly clog the search space. In 1970 Knuth and Bendix [15] (see [10] for a more complete treatment of the mathematics), working independently of Robinson and Wos, created a very effective procedure for deriving useful consequences from equality units. Their process used paramodulation, but since it also used simplification and subsumption removal, most of the derived equalities were discarded and the search space remained small.

The main defects with the Knuth–Bendix procedure were that each equality had to be construed as a reduction, so equalities such as the commutative law were excluded, and that the process works only on equality units, so axioms such as $x \neq 0 \Rightarrow xx^{-1} = 1$ in field theory were excluded. The first defect has been at least partially removed by using specialized unification algorithms such as associative-

* Received by the editors March 17, 1980, and in final revised form February 15, 1982.
† Department of Mathematical Sciences, University of Missouri–St. Louis, St. Louis, Missouri 63121.

commutative unification [22], [27], [31], [36]. There has also been progress [17], [21] in removing the second defect, but to the author's knowledge, no one has developed a refutation complete set of inference rules for all of first-order logic with equality which reduces to the Knuth–Bendix procedure when restricted to equality units.

The inference system which is developed in this paper is refutation complete, and it does reduce to a procedure which is nearly identical to the Knuth–Bendix process when restricted to equality units, but completeness results are obtained only when the underlying ordering of terms and atoms is order-isomorphic to the positive integers. This is a condition which Knuth and Bendix did not impose on their ordering and, indeed, many of the complete sets of reductions contained in their examples used versions of their ordering which were not order-isomorphic to the positive integers. However, our goal is somewhat different from that of Knuth and Bendix. They were searching for complete sets of reductions; we are searching for an efficient inference system which is complete in the logical sense. It does not matter to us (at least not at present) whether or nor any complete sets of reductions are obtained in the course of a proof, but only that useful consequences be derived and trivial ones eliminated. This goal seems likely to be achieved even with the isomorphism condition on the order. It may be possible to obtain completeness theorems without the isomorphism condition, but the mathematics to do so might involve transfinite trees and would be the subject of a separate paper.

The main result of this paper is that an automatic inference system consisting of resolution, paramodulation, factoring and equality reversal is complete in first-order logic with equality. Useful restrictions may be imposed which do not destroy the completeness. These include: no paramodulation into variables, deletion of subsumed or simplified clauses, and resolution only of simplified clauses.

In rough outline, these results were obtained as follows. A key idea is the embedding of equality atoms in a simplification ordering of atoms and terms in a way such that each equality occurs before terms or atoms which it may reduce. This is described in § 3, and it is shown how to create one such ordering by modifying that of Knuth and Bendix. In § 4 we show how to build E-interpretations inductively on an ordered set B of atoms. This process can then be used to create E-semantic trees, that is, trees whose branches correspond to E-interpretations. If S is an E-unsatisfiable set of clauses, then each branch of an E-semantic tree for S can be cut off at some point and the remaining partial interpretation will still falsify some ground instance of a clause in S. It is shown how a properly chosen resolvent or paramodulant can be added to S so that the cut-off tree for the new S is in some sense smaller than the cut-off tree for the original S. If a sufficient number of resolvents and/or paramodulants are added, the tree will shrink to its root and thus imply that the empty clause has been generated. These ideas are considered in § 6. A difficulty with this approach is that initially the resolutions and paramodulations are known to be available on the ground level and a "lifting lemma" is required to ensure that corresponding inferences are available at the general level. Unfortunately, the paramodulation lifting lemma does not always work. In order to ensure that the lifting lemma will work when paramodulating from $C_1\phi$ into $C_2\theta$, it is necessary to know that the replaced portion of $C_2\theta$ consists of a subtree of $C_2\theta$ that begins in C_2. We provide this assurance by showing that it is possible to use only substitutions whose terms are not available for paramodulations of the type necessary for the completeness theorem; that is, the terms of the substitutions are "completely reduced." These ideas are considered at the end of § 4 and in § 6 at the point where λ_N is defined. In § 7 we discuss deletion strategies, and in § 8 a hand example is cranked out.

The paper is meant to be theoretical. However, an implementation of a prover based on these ideas will be created and experiments will be conducted to better assess its usefulness. Some of the proofs are presented in excruciating detail. This is felt to be necessary because intuition is of little, if any, value in these matters, and the terrain is loaded with pitfalls.

2. Terminology. Basically we use [3] for terminology. Some differences and key ideas are presented here. Let T be a labeled tree. The set of nodes of T will be called the *domain* of T, dom T. If $n \in \text{dom } T$ and the label at n is l, then we write $Tn = l$ or $T(n) = l$. If $n, m \in \text{dom } T$, then n is an *ancestor* of m, $n \geqslant m$, if n is on the branch of T leading from m to the root, including the case when $n = m$. We write $n \perp m$ and say n is *independent* of n if n is not an ancestor of m and m is not an ancestor of n. The subtree of T which begins at node n is denoted by T/n. If u is another tree, then $T[n \leftarrow u]$ is the tree formed when T/n is replaced by u. When no confusion is likely, this will be abbreviated to $T[u]$. Similarly, if $n_1 \perp n_2$ and u_1 and u_2 are trees, then $T[n_1 \leftarrow u_1, n_2 \leftarrow u_2]$ is $(T[n_1 \leftarrow u_1])[n_2 \leftarrow u_2]$ and will be shortened to $T[u_1, u_2]$. When $T[u_1]$ and $T[u_2]$ are used in the same discussion, it is to be understood that there is $n \in \text{dom } T$ such that these are $T[n \leftarrow u_1]$ and $T[n \leftarrow u_2]$, respectively. See [34] for a rigorous discussion of tree terminology.

We consider terms, clauses, etc., to be labeled trees. If A is a term or a clause and θ is a substitution, then $A\theta$ is the result of applying θ to A; that is, each variable of A is replaced by the term specified in θ. Let the *restricted domain* of A, rdom A, be the set of all $n \in \text{dom } A$ such that An is not a variable.

We work in first-order logic with equality and use $=$ as the equality symbol both in the metalanguage and in the first-order language. To avoid confusion, equality atoms in the first-order language will usually be enclosed in parentheses.

3. Simplification orderings. We will be concerned with a first-order logical system with equality. Define a *word* to be either a term or an atom and an *equality word*, or simply an *equality*, to be an atom whose predicate symbol is $=$. Let $\mathcal{T}$ denote the set of all terms and $\mathcal{W}$ the set of all words.

We assume that the set of all non equality words has been endowed with a partial order, $<$, which satisfies the following properties.

W1. The full set of non equality words is well-founded [23, p. 205], [25, p. 183], [4, p. 20].

W2. The subset of ground non equality words is order-isomorphic [4, p. 22] to the set of positive integers.

W3. For all non equality words w, v and every substitution θ, if $w < v$, then $w\theta < v\theta$.

W4. For each non equality word w and each term t, if t is a subterm of w, then $t \leqq w$. In particular, if t is a strict subterm of w, then $t < w$.

W5. For every non equality word w, every $n \in \text{dom } w$, and all terms t, s, if $t < s$, then $w[n \leftarrow t] < w[n \leftarrow s]$.

Note that W1 follows from W2 and W3. We include W1 only for emphasis.

Orderings on sets of terms which satisfy W4 and W5 (with "non equality word" replaced by "term") were called *simplification orderings* by Dershowitz [5], [6]. Plaisted [28], [29], however, uses the denotation *simplification ordering* for an ordering on terms which is total on ground terms and satisfies the analogues of W1, W3 and W5. In either case we may consider the order $<$ as a special kind of simplification order which has been extended to apply to all non equality words.

The prototype of an order which satisfies W1–W5 is the order of Knuth and Bendix [15] in which every operator is given positive weight. Knuth and Bendix define their order only for terms, but it is easily extended to apply to atoms as well. For convenience we repeat their definition here. Let O be the set of operators, that is, function (including constant) or non equality predicate symbols. Assume O is finite and linearly ordered. Let λ be a mapping from O into the positive integers which assigns to each operator a weight. Extend λ to apply to non equality words as follows. Let w be a non equality word. If w is a variable, then $\lambda(w) = \lambda_0$ where λ_0 is the minimum weight of a constant. If $w = f(t_1, \cdots, t_k)$, then λ is defined by structural induction as $\lambda(w) = \lambda(f) + \lambda(t_1) + \cdots + \lambda(t_k)$. If w is a word and s a symbol, let $n(s, w)$ denote the number of occurrences of s in w. If w and v are words, then $w > v$ if and only if

(1) $\lambda(w) > \lambda(v)$ and $n(x, w) \geqq n(x, v)$ for every variable x; or

(2) $\lambda(w) = \lambda(v)$ and $n(x, w) = n(x, v)$ for every variable x and with $w = f(t_1, \cdots, t_k)$, $v = g(s_1, \cdots, s_l)$ either

(2a) $f > g$ in the order on O; or

(2b) $f = g$ and $t_1 = s_1, \cdots, t_{j-1} = s_{j-1}, t_j > s_j$ for some j, $1 \leqq j \leqq k$.

The definition in [15] is slightly more complicated than this since Knuth and Bendix allow some of their operators to have zero weight.

That this order satisfies W1–W5 is quite easy to see. Knuth and Bendix show that it is a well-order on ground words and that it satisfies W1 and W3. Properties W4 and W5 are easily established and are left to the reader. Property W2 will follow if we show that every ground word has only finitely many predecessors. Any ground word w has some weight, $\lambda(w)$, and since there are only finitely many operator symbols, each with positive weight, there are only finitely many ground words of any given weight. Thus there are only finitely many ground words with weight less than or equal to $\lambda(w)$. But, if $v < w$, then $\lambda(v) \leqq \lambda(w)$ by definition, so there are only finitely many ground words less than w.

The author is unaware of any other easily calculable ordering which satisfies all the properties W1–W5. Other methods of defining such an order would be welcome.

In what follows it will be necessary to have the equality $(s = t)$ precede words which can be simplified by $(s = t)$. It is therefore necessary to imbed equalities in the order in a special way.

DEFINITION. Suppose w is a non equality word and a, b, s, t are terms. Then,

EO1. $w < (s = t)$ means $w < s$ or $w < t$;

EO2. $(s = t) < w$ means $s \leqq w$ and $t \leqq w$;

EO3. $(s = t) < (a = b)$ means

EO3a. $s \leqq a$ and $t \leqq b$ with one inequality strict, or

EO3b. $s \leqq b$ and $t \leqq a$ with one inequality strict, or

EO3c. $s = b$ and $t = a$ and $t < s$, or

EO3d. $s < a$ and $t < a$, or

EO3e. $s < b$ and $t < b$.

THEOREM 1. *The order $<$ on words which has just been defined satisfies the following properties.*

O1. *The set $\mathcal{W}$ is well-founded.*

O2. *The subset of ground words is order-isomorphic to the positive integers.*

O3. *For all $w, v \in \mathcal{W}$ and every substitution θ, if $w < v$, then $w\theta < v\theta$.*

O4. *If $t \in \mathcal{T}$, $w \in \mathcal{W}$, t is a subterm of w and w is not an equality, then $t \leqq w$.*

O5. *If $w \in \mathcal{W}$, $n \in \operatorname{dom} w$, $t, s \in \mathcal{T}$, and $t < s$, then $w[n \leftarrow t] < w[n \leftarrow s]$.*

O6. *If $t, s \in \mathcal{T}$ and $t < s$, then $(s = t)$ and $(t = s)$ are adjacent and $(t = s) > (s = t)$.*

O7. *Suppose* $t, s, a, b \in \mathcal{T}$, $w \in \mathcal{W}$, $t \leqq s$, *and* s *is a subterm of* w. *If* w *is not an equality, or if* $w = (a = b)$ *and* s *is a strict subterm of* a *or* b, *or if* $w = (a = s)$ *or* $(s = a)$ *and* $t < a$, *then* $(t = s) < w$.

Proof (excerpts). Much of the proof of this theorem consists of tedious case-by-case analyses. Therefore, we just give an outline and some highlights. Parts 1 and 2 of the proof establish that $<$ is indeed an order, parts 3, 4, and 5 suffice to give O2, and the relationship of the remainder of the proof to the theorem statement is self-explanatory.

Part 1. $<$ *is irreflexive.* Suppose $w < w$ for some word w. Then w must be an equality, say $(s = t)$. From $(s = t) < (s = t)$ it follows using EO3 that $s \leqq s$ and $t \leqq t$ with one inequality strict, or $s = t$ and $t < s$, or $s < s$ and $t < s$, or $s < t$ and $t < t$. But none of these is possible.

Part 2. $<$ *is transitive.* We prove two representative cases and leave the rest to the reader.

Case 1. Suppose $w < (s = t)$ and $(s = t) < (a = b)$ where w is not an equality and $(s = t) < (a = b)$ via EO3b. Then from EO1 and EO3b we have $w < s$ or $w < t$, and $s \leqq b$ and $t \leqq a$. It follows that $w < b$ or $w < a$. Thus $w < (a = b)$.

Case 2. Suppose $(s = t) < (a = b)$ and $(a = b) < (c = d)$ where the first inequality is via EO3d and the second comes via EO3a. Then $s < a$ and $t < a$ and $a \leqq c$ and $b \leqq d$. Thus $s < c$ and $t < c$. By EO3d, $(s = t) < (c = d)$.

Part 3. $<$ *is a total order on ground words.* Suppose w and v are ground words. If neither is an equality, then from W2 it follows that $w < v$, $w = v$ or $w > v$. Suppose one of w and v is an equality, say $w = (s = t)$. If $s \leqq v$ and $t \leqq v$, then $w < v$ by EO2. Otherwise, either $v < s$ or $v < t$ and then $v < w$ by EO1. Finally, suppose both w and v are equalities, $w = (s = t)$ and $v = (a = b)$. Then there are many cases, but each is covered by EO3, giving $w < v$ or $w = v$ or $w > v$.

Part 4. *The ground words have a minimum element.* Suppose this were not so and let $w_1 > w_2 > w_3 \cdots$ be an infinite descending sequence of ground words. By W2, only finitely many of these can be non equalities and we may, therefore, suppose that for every i, $w_i = (s_i = t_i)$. By EO3, one of the following conditions holds for each i.

(1) $s_i > s_{i+1}$ and $t_i > t_{i+1}$,
(2) $s_i > s_{i+1}$ and $t_i = t_{i+1}$,
(3) $s_i = s_{i+1}$ and $t_i > t_{i+1}$,
(4) $s_i > t_{i+1}$ and $t_i > s_{i+1}$,
(5) $s_i > t_{i+1}$ and $t_i = s_{i+1}$,
(6) $s_i = t_{i+1}$ and $t_i > s_{i+1}$,
(7) $s_i = t_{i+1}$ and $t_i = s_{i+1}$ and $t_{i+1} < s_{i+1}$,
(8) $s_i > s_{i+1}$ and $s_i > t_{i+1}$,
(9) $t_i > s_{i+1}$ and $t_i > t_{i+1}$.

Condition (7) cannot hold for two successive inequalities in the sequence because if $(s_{i-1} = t_{i-1}) > (s_i = t_i) > (s_{i+1} = t_{i+1})$ and both inequalities are via (7), it follows that $s_i > t_i$ and $s_i < t_i$. Therefore, there are infinitely many inequalities in the chain which obtain for reasons other than (7). We construct two sequences of terms, A and B, as follows. Begin with $A = s_1$ and $B = t_1$. For i running from 2 to ∞, add to the chains according to the following rules. If condition (1) holds, apply rule (1), else if condition 2 holds, apply rule (2), etc.

(1) $A \leftarrow A, s_{i+1}$ and $B \leftarrow B, t_{i+1}$,
(2) $A \leftarrow A, s_{i+1}$,
(3) $B \leftarrow B, t_{i+1}$,
(4) $A \leftarrow B, s_{i+1}$ and $B \leftarrow A, t_{i+1}$,

(5) $B \leftarrow A$, t_{i+1},
(6) $A \leftarrow B$, s_{i+1},
(7) $A \leftarrow B$ and $B \leftarrow A$,
(8) $A \leftarrow A$, s_{i+1} and $B \leftarrow A$, t_{i+1},
(9) $A \leftarrow B$, s_{i+1} and $B \leftarrow B$, t_{i+1}.

Note that at the ith stage, A ends in s_i and B ends in t_i, and that each of A and B is a descending sequence. Let LA and LB be the length of A and the length of B, respectively, each changing as i increases. Each of conditions (1), (4), (8) and (9) increases min (LA, LB) and no condition decreases min (LA, LB). Thus, if any combination of inequalities satisfying (1), (4), (8) or (9) occurred infinitely often, we would have LA and LB both increasing to infinity and A and B would both be infinite descending sequences of ground terms. Since this is impossible, we must have some combination of (2), (3), (5) and (6) occurring infinitely often. But each of (2), (3), (5) and (6) increases the length of one of the two chains and leave the length of the other one alone. Therefore, both of these chains cannot remain finite in length. Again this is impossible, and we must conclude at this point that our original hypothesis of an infinite descending sequence of ground words was in error.

Part 5. *Every ground word has only finitely many predecessors.* Let w be a ground word. If w is not an equality, then by W2, w can have only finitely many non equality predecessors and by EO2, w can have only finitely many equality predecessors. If w is an equality, then by EO1 and W2, w can have only finitely many non equality predecessors. Let $w = (a = b)$. If $(s = t) < w$, then by EO3, both s and t are less than max (a, b). Thus there are at most finitely many equalities less than w.

This completes the proof of O2.

Part 6. O3. Suppose $w < v$ and θ is a substitution. A simple case analysis will show that $w\theta < v\theta$. For example, if $w = (s = t)$, then we have $(s = t) < v$ and by EO2, $s \leqq v$ and $t \leqq v$, then by W3, $s\theta \leqq v\theta$ and $t\theta \leqq v\theta$, finally by EO2, $(s\theta = t\theta) < v\theta$ which is the same as $w\theta < v\theta$.

Property O1 follows from O2 and O3.

Part 7. O4. This is W4.

Part 8. O5. This follows from W5 and EO3a.

Part 9. O6. That $(t = s) > (s = t)$ follows from EO3c. That $(s = t)$ and $(t = s)$ are adjacent requires showing that $(s = t) < w < (t = s)$ leads to a contradiction for any word w. This is not difficult, but requires a tedious case-by-case analysis and is left to the reader.

Part 10. O7. Suppose $t \leqq s$ and s is a subterm of w. If w is not an equality, then $s \leqq w$ by O4 and $(t = s) < w$ by EO2. If $w = (a = b)$ and s is a strict subterm of a or b, then either $t \leqq s < a$ or $t \leqq s < b$. By EO3d or EO3e, $(s = t) < w$. If $w = (a = s)$ or $(s = a)$ and $t < a$, then $(t = s) < w$ by EO3a or EO3b. □

In what follows, we use only that $<$ is a partial order on $\mathcal{W}$ which satisfies O1 to O7.

4. Interpretations. Let $\mathcal{G}$ be the set of ground words and $\mathcal{B}$ the set of ground atoms. An *interpretation on a set* $\mathcal{B}' \subseteq \mathcal{B}$ is a mapping $I : \mathcal{B}' \to \{\mathbf{T}, \mathbf{F}\}$. An *interpretation* is an interpretation on $\mathcal{B}$. An *E-interpretation* on $\mathcal{B}'$ is an interpretation I on $\mathcal{B}'$ which satisfies

E1. $I(s = s) = \mathbf{T}$ if $(s = s) \in \mathcal{B}'$,
E2. $I(s = t) = I(t = s)$ if $(s = t), (t = s) \in \mathcal{B}'$, and
E3. if $(s = t)$, $B[s]$ and $B[t]$ are in $\mathcal{B}'$ and $I(s = t) = \mathbf{T}$, then $I(B[s]) = I(B[t])$.

An *E-interpretation* is an E-interpretation on $\mathcal{B}$.

Note that E2 follows from E1 and E3 if $\mathcal{B}' = \mathcal{B}$ and that the transitive law: if $(s = t)$, $(t = a)$, $(s = a) \in \mathcal{B}'$, $I(s = t) = \mathbf{T}$, and $I(t = a) = \mathbf{T}$, then $I(s = a) = \mathbf{T}$: follows from E3.

Since $\mathcal{G}$ is order-isomorphic to the positive integers and $\mathcal{B} \subset \mathcal{G}$, we may write $\mathcal{B} = \{B_1, B_2, \cdots\}$, where $B_i < B_j$ if and only if $i < j$. For each $k \geqq 1$, define $\mathcal{B}_k = \{B_1, B_2, \cdots, B_k\}$. A *left segment* of $\mathcal{B}$ is either $\mathcal{B}$ itself or one of the sets $\mathcal{B}_k$. Note that I is an E-interpretation on $\mathcal{B}$ if and only if I is an E-interpretation on $\mathcal{B}_k$ for all k.

Our immediate goal is to describe how an E-interpretation on $\mathcal{B}_{k-1}$ can be extended to an E-interpretation on $\mathcal{B}_k$.

Suppose I is an interpretation on a left segment $\mathcal{B}'$ of $\mathcal{B}$. Let $\mathcal{P}(\mathcal{G})$ be the power set of $\mathcal{G}$, that is, the set of all subsets of $\mathcal{G}$. We define a function $f_I : \mathcal{G} \to \mathcal{P}(\mathcal{G})$ by induction as follows. If w is the smallest element in $\mathcal{G}$, then $f_I(w)$ is the empty set, $\varnothing$. If f_I has been defined for every element of $\mathcal{G}$ which is less than w, then $f_I(w)$ is defined as the set of all $v \in \mathcal{G}$ such that either

F1. $w = (t = s)$, $t < s$, $f_I(s = t) = \varnothing$, and $v = (s = t)$, or

F2. there is a subterm s of w, $w = w[s]$, and a term t such that $t < s$, $(s = t) < w$, $(s = t) \in B'$, $I(s = t) = \mathbf{T}$, $f_I(s = t) = \varnothing$, and $v = w[t]$.

We now define a partial order, $\to(I)$, or simply $\to$ if I is understood, on $\mathcal{G}$ by the statement: $w \to (I)\ v$ if and only if $v \in f_I(w)$. It is clear that $w > v$ if $w \to v$ and, therefore, $\to$ is well-founded. We will say that w *reduces* (I) to v if $w \to (I)\ v$, and that w is irreducible (I) if $f_I(w) = \varnothing$. If $w \to v$ and $v \in f_I(w)$ by reason of F2, then we write $w \to v$ *using* $(s = t)$.

When constructing an E-interpretation, the truth values which are assigned the various atoms cannot be chosen independently. The purpose of the reduction $\to(I)$ is to provide an explicit relationship between an atom and other atoms below it in the $<$ order which all must be assigned the same truth value if I is to be an E-interpretation. Condition F1 relates to the fact that $(s = t)$ and $(t = s)$ must have the same truth value, and F2 relates to the fact that replacing one side of a true equality by the other, in any term, must not alter the truth value of that term.

An arbitrary partial order, R, is *confluent* [11] if for every x and y, if there exists z such that zR^*x and zR^*y, then there exists w such that xR^*w and yR^*w, where R^* is the reflexive, transitive closure of R. The following example shows that $\to$ is not necessarily confluent.

Example. Suppose $\mathcal{B}'$ contains the set of atoms shown in the left column of Table 1, $A < B$ if A is above B in the table, and the interpretation I is as shown in the

TABLE 1

$\mathcal{B}'$	I
$^*b = a$	T
$a = b$	F
$^*g(a) = a$	T
$g(a) = b$	F

second column. We are supposing that $a < b$. We have marked with an asterisk those equalities $(s = t)$ that are irreducible and satisfy $I(s = t) = T$. That is, they are the equalities that may be used in reducing other words. Note that

$$(g(a) = b) \to (a = b) \to (b = a),$$

$$(g(a) = b) \to (g(a) = a),$$

$(b = a)$ is irreducible, and $(g(a) = a)$ is irreducible. Thus, $\rightarrow$ is not confluent, for with $x = (b = a)$, $y = (g(a) = a)$, there is a $z = (g(a) = b)$, but no w.

The usefulness of $\rightarrow(I)$ stems from the following two theorems. The first of these shows that $\rightarrow(I)$ does, in some cases, have a property similar to confluence.

THEOREM 2. *Suppose I is an E-interpretation on $\mathscr{B}_{k-1}$. If $B_k \rightarrow C$ and $B_k \rightarrow D$, then $I(C) = I(D)$.*

Proof. The only way for $f_I(B_k)$ to contain both C and D (if $C \neq D$) is for $B_k \rightarrow C$ using $(s_1 = t_1)$ and $B_k \rightarrow D$ using $(s_2 = t_2)$. Let $B_k/n_1 = s_1$ and $B_k/n_2 = s_2$. The proof will be by cases depending on the relation of n_1 and n_2 in dom B_k.

Case 1. Suppose $n_1 \perp n_2$. If we write $B_k[u, v] = B_k[n_1 \leftarrow u, n_2 \leftarrow v]$, then we have by hypothesis

$$I(C) = I(B_k[t_1, s_2]) = I(B_k[t_1, t_2]) = I(B_k[s_1, t_2]) = I(D).$$

Case 2. Suppose n_1 and n_2 are not independent. Then one must be an ancestor of the other and we assume $n_1 \geqslant n_2$ for definiteness. Then s_2 is a subterm of s_1. If s_2 were a strict subterm of s_1, then by O7, $(s_2 = t_2) < (s_1 = t_1)$ and it follows that $(s_1 = t_1)$ is irreducible (I). Thus $s_2 = s_1$. Write $s = s_2 = s_1$ and $B_k = B_k[s]$. Since $t_1 < s \leqq B_k$ and $t_2 < s \leqq B_k$, it follows from EO2 that $(t_1 = t_2) < B_k$. Since $I(s = t_1) = \mathbf{T}$ and $I(s = t_2) = \mathbf{T}$, and I is an E-interpretation on $\mathscr{B}_{k-1}$, we have $I(t_1 = t_2) = \mathbf{T}$. Therefore

$$I(C) = I(B_k[t_1]) = I(B_k[t_2]) = I(D)$$

since C, $B_k[t_1]$, $B_k[t_2]$ and D are all in $\mathscr{B}_{k-1}$. □

THEOREM 3. *Suppose I is an interpretation on $\mathscr{B}_k$ which is an E-interpretation on $\mathscr{B}_{k-1}$. Then I is an E-interpretation on $\mathscr{B}_k$ if and only if*

(1) *B_k is reducible (I) and for all C such that $B_k \rightarrow C$, $I(B_k) = I(C)$, or*

(2) *B_k is irreducible, of the form $(t = t)$ and $I(B_k) = \mathbf{T}$, or*

(3) *B_k is irreducible and not of the form $(t = t)$.*

Proof. If neither (1), (2) nor (3) holds, then it is obvious that I is not an E-interpretation on $\mathscr{B}_k$. Suppose that (1), (2) or (3) holds. We must show that I is an E-interpretation on $\mathscr{B}_k$, i.e., that E1, E2 and E3 hold on $\mathscr{B}_k$. Since I is known to be an E-interpretation on $\mathscr{B}_{k-1}$, it will suffice to prove the following:

P01. $I(B_k) = \mathbf{T}$ if $B_k = (t = t)$ for some term t.

P02. If $B_k = (t = s)$ and $t < s$, then $I(B_k) = I(s = t)$.

P03. If $B_k = (s = t)$, $A[s] < B_k$, $A[t] < B_k$ and $I(B_k) = \mathbf{T}$, then $I(A[s]) = I(A[t])$.

P04. If $B_k = B_k[s]$, $(s = t) < B_k$, $B_k[t] < B_k$ and $I(s = t) = \mathbf{T}$, then $I(B_k) = I(B_k[t])$.

P05. If $B_k = B_k[t]$, $(s = t) < B_k$, $B_k[s] < B_k$ and $I(s = t) = \mathbf{T}$, then $I(B_k) = I(B_k[s])$.

P01. If $B_k = (t = t)$ is irreducible, then neither (1) nor (3) holds, so we must have (2) and $I(B_k) = \mathbf{T}$. If $B_k = (t = t)$ reduces using $(s_1 = t_1)$ then s_1 must be a subterm of t and we have

$$\begin{aligned} I(B_k) &= I(t[s_1] = t[s_1]) \\ &= I(t[t_1] = t[s_1]) && \text{by (1)} \\ &= I(t[t_1] = t[t_1]) && \text{by hypothesis} \\ &= \mathbf{T} && \text{by hypothesis.} \end{aligned}$$

P02. Suppose $B_k = (t = s)$ and $t < s$. If $(s = t)$ is irreducible, then $(t = s) \rightarrow (s = t)$ by $F1$ and $I(B_k) = I(s = t)$ by (1). If $(s = t)$ reduces to $(s' = t')$, then $(t = s)$ reduces to

$(t' = s')$. Therefore

$$I(B_k) = I(t' = s') \quad \text{by (1)}$$

$$= I(s' = t') \quad \text{by hypothesis}$$

$$= I(s = t) \quad \text{by hypothesis.}$$

P03. Suppose $B_k = (s = t)$, $A[s] < B_k$, $A[t] < B_k$ and $I(B_k) = \mathbf{T}$. If $s < t$, then by P02, $I(s = t) = I(t = s) = \mathbf{T}$. By hypothesis, $I(A[s]) = I(A[t])$. If $s = t$ the result is obvious. Suppose $s > t$. By O7, $A[s] = (s = a)$ or $(a = s)$ and $t > a$. For definiteness, assume $A[s] = (s = a)$; the other case will be similar. If B_k is irreducible, then $I(A[s]) = I(A[t]) = \mathbf{F}$. For, if otherwise, say $I(A[s]) = \mathbf{T}$, then $I(A[s]) = I(s = a) = I(s = a') = \mathbf{T}$, where a' is irreducible and $a \to^* a'$. This implies that B_k is reducible ($\to \leftarrow$). Suppose B_k reduces using $(s_1 = t_1)$. If s_1 is a subterm of s, then $\mathbf{T} = I(s[s_1] = t) = I(s[t_1] = t)$ by (1). So by hypothesis

$$I(A[s]) = I(s[s_1] = a)$$

$$= I(s[t_1] = a)$$

$$= I(t = a)$$

$$= I(A[t]).$$

If s_1 is a subterm of t, then similarly $\mathbf{T} = I(s = t[s_1]) = I(s = t[t_1])$ and

$$I(A[s]) = I(s = a)$$

$$= I(t[t_1] = a)$$

$$= I(t[s_1] = a)$$

$$= I(A[t]).$$

P04. Suppose $B_k = B_k[s]$, $(s = t) < B_k$, $B_k[t] < B_k$ and $I(s = t) = \mathbf{T}$. It follows from $B_k[t] < B_k[s]$ that $t < s$.

Suppose first that s is irreducible. Let $t \to^* t'$ where t' is irreducible. Then by hypothesis, $\mathbf{T} = I(s = t) = I(s = t') = I(t = t')$. Thus

$$I(B_k[s]) = I(B_k[t']) \quad \text{by (1)}$$

$$= I(B_k[t]) \quad \text{by hypothesis.}$$

Suppose now that s is reducible to s' using $(s_1 = t_1)$. Write $s = s[s_1]$. Then $\mathbf{T} = I(s = t) = I(s[t_1] = t)$ and

$$I(B_k[s]) = I(B_k[s[s_1]]) = I(B_k[s[t_1]]) = I(B_k[t]).$$

P05. This becomes the same as P04 with s and t interchanged if we note that $s < t$ and therefore $I(s = t) = I(t = s) = \mathbf{T}$. □

Theorem 3 will be used to construct semantic trees whose branches correspond to E-interpretations. Completeness results obtained using semantic trees require a so-called lifting lemma [3, p. 84] to make them applicable to clause sets containing variables. The lifting lemma for the paramodulation inference rule works only if the paramodulation was done on the ground level to a term which can be "seen" at the general level; that is, the paramodulation was not done inside a term which came from a substitution. It is necessary, therefore, to deal only with "fully simplified" substitutions. The purpose of the following discussion is to formalize these ideas.

Let I be an E-interpretation on a left segment $\mathscr{B}'$ of $\mathscr{B}$. Let $\theta = \{x_1 \leftarrow t_1, \cdots, x_k \leftarrow t_k\}$ be a ground substitution. We write $\theta \rightarrow (I)\ \theta'$ if θ' is identical to θ except that one t_i has been replaced by t_i' and $t_i \rightarrow t_i'$. We say that θ is *irreducible* if every term t_i of θ is irreducible.

Let $\mathscr{C}(\mathscr{B}')$ be the set of clauses that can be formed from the atoms in $\mathscr{B}'$. If I is an interpretation on $\mathscr{B}'$ then I can be extended to $\mathscr{C}(\mathscr{B}')$ in the usual manner by defining $I(\sim A)$ as $\sim I(A)$ for $A \in \mathscr{B}'$ and $I(L_1 \vee \cdots \vee L_k)$ as $I(L_1) \vee \cdots \vee I(L_k)$ for literals $L_1, \cdots, L_k$ whose atoms are in $\mathscr{B}'$.

THEOREM 4. *Suppose θ is a ground substitution and C is a clause such that $C\theta$ is in $\mathscr{C}(\mathscr{B}')$. If $\theta \rightarrow^* \theta'$, then $I(C\theta) = I(C\theta')$.*

Proof. It will suffice to prove that if $\theta \rightarrow \theta'$, then $I(C\theta) = I(C\theta')$. Suppose $\theta \rightarrow \theta'$. Then a term t_i of θ has been replaced by t_i' in order to form θ' and $t_i \rightarrow t_i'$. Since t_i occurs somewhere in $C\theta$ (if it doesn't, the result is obvious), it follows that $t_i < A$ for some $A \in \mathscr{B}'$ and therefore $(t_i = t_i') \in \mathscr{B}'$. Now $(t_i = t_i') \rightarrow (t_i' = t_i')$. Since I is an E-interpretation on $\mathscr{B}'$ and $(t_i = t_i') \in \mathscr{B}'$, it follows that $I(t_i = t_i') = \mathbf{T}$. But $C\theta'$ differs from $C\theta$ only in that 1 or more occurrences of t_i have been replaced by t_i'. It follows that $I(C\theta) = I(C\theta')$. □

5. Inference rules. Let S be a set of clauses. In the next section we will deal with proof procedures which are composed of some or all of the following rules of inference. In those rules involving two clauses, we assume that their variables have been standardized apart.

Factoring. If $L_1, \cdots, L_k$ are literals of a clause C which are unifiable with mgu σ, then a *factor* of C is the clause $C' = C\sigma - (L_2\sigma \vee \cdots \vee L_k\sigma)$. The *factoring rule* states that one may add to S a factor of a clause $C \in S$.

Resolution. If $C_1 = L_1 \vee C_1'$ and $C_2 = L_2 \vee C_2'$ are clauses such that L_1 and $\tilde{}L_2$ are unifiable with mgu σ, then a resolvent of C_1 and C_2 is the clause $C = C_1'\sigma \vee C_2'\sigma$. The *resolution rule* states that one may add to S a resolvent of clauses $C_1, C_2 \in S$.

Paramodulation. If C_1 and C_2 are clauses such that $C_1 = (s = t) \vee C_1'$ and C_2 has a subterm s' at node n which is unifiable with s with mgu σ, then a *paramodulant* C of C_1 into node n of C_2 is the clause $C = (C_2[n \leftarrow t] \vee C_1')\sigma$. The *paramodulation rule* states that one may add to S a paramodulant of clauses $C_1, C_2 \in S$.

Equality reversal. If $C = \pm(s = t) \vee C'$ is in S, then the *equality reversal* rule states that one may add to S the clause $\pm(t = s) \vee C'$.

Simplification. If $C_1 = (s = t)$, a clause C_2 contains a subterm which is an instance $s\sigma$ of s, and $s\sigma > t\sigma$, then the clause $C = C_2[t\sigma]$ is a *simplification* of C_2 using C_1. The *simplification rule* states that one may *replace* in S a clause which has been simplified, by its simplification.

Subsumption. If C_1 and C_2 are clauses such that $C_1\sigma \subseteq C_2$ for some substitution σ, then C_2 is said to be *subsumed* by C_1. The *subsumption rule* states that one may delete from S any clause which is subsumed by another clause in S.

Examples. Suppose S is the set of clauses

C1. $P(x) \vee f(g(x, a)) = x$,

C2. $\sim P(g(a, x)) \vee Q(f(x))$,

C3. $P(x) \vee P(f(y))$,

C4. $f(x) = x$,

C5. $Q(x)$.

Then $P(f(y))$ is a factor of C3,

$$f(g(g(a, x), a)) = g(a, x) \vee Q(f(x))$$

is a resolvent of C1 and C2,

$$\sim P(g(a, g(x, a))) \vee Q(x) \vee P(x)$$

is a paramodulant of C1 into C2, $P(x) \vee P(y)$ is a simplification of C3, and C5 subsumes C2.

The following fundamental result is due to G. Robinson and L. Wos [32].

THEOREM 5 (Paramodulation lifting lemma). *If C' is a paramodulant of $C_1\theta$ into node n of $C_2\theta$ and $n \in \operatorname{dom} C_2$, then there is a paramodulant C of C_1 into node n of C_2 such that C' is an instance of C.*

Proof. Let $C_1 = (s = t) \vee C_1'$; then $C_1\theta = (s\theta = t\theta) \vee C_1'\theta$. Let ϕ be the mgu of $s\theta$ and $C_2\theta/n$, so that

$$C' = (C_2\theta[n \leftarrow t\theta] \vee C_1'\theta)\phi.$$

Now $C_2\theta/n = (C_2/n)\theta$, and $s\theta\phi = (C_2\theta/n)\phi$. Thus $(C_2/n)\theta\phi = s\theta\phi$ and it follows that C_2/n and s are unifiable. Let σ be their mgu. Then there is a substitution ψ such that $\theta\phi = \sigma\psi$ and there is a paramodulant C of C_1 into node n of C_2 given by

$$C = (C_2[n \leftarrow t] \vee C_1')\sigma.$$

Furthermore,

$$\begin{aligned} C\psi &= (C_2[n \leftarrow t] \vee C_1')\sigma\psi \\ &= (C_2[n \leftarrow t] \vee C_1')\theta\phi \\ &= (C_2\theta[n \leftarrow t\theta] \vee C_1'\theta)\phi \\ &= C'. \end{aligned}$$

□

6. Semantic trees. The usefulness of semantic trees in obtaining completeness theorems for sets of clauses without equality has been considered in [16].

A *truth-value tree* is a binary tree such that each node other than the root is labeled with **T** or **F**. If τ is a truth-value tree and N is a node of τ, then $\tau(N)$ denotes the label at N, i.e., τ is considered to be a function from the nodes to the labels. A *numbered truth-value tree* is a finite truth-value tree such that each leaf node is labeled with a nonnegative integer in addition to its truth value. If N is a leaf node we write $\tau'(N)$ to denote the associated number. We partially order numbered truth-value trees by writing $\tau_1 < \tau_2$ if and only if

TO1. $\operatorname{dom} \tau_1 \subset \operatorname{dom} \tau_2$, or

TO2. $\operatorname{dom} \tau_1 = \operatorname{dom} \tau_2$ and $\tau_1'(N) \leqq \tau_2'(N)$ for every leaf node N with the inequality strict for at least one N.

It is clear that this partial order is well-founded.

Let N be a node in a truth-value tree at level k. Then $I_N : \mathcal{B}_k \to \{\mathbf{T}, \mathbf{F}\}$ is the interpretation on $\mathcal{B}_k$ defined by $I_N(B_j) = \tau(M)$ where M is the ancestor of N at level j. Similarly, if b is a branch of a truth-value tree, then I_b is the interpretation (possibly

on a finite left segment of $\mathscr{B}$ if b is finite) defined by $I_b(B_j) = \tau(M)$ where M is the node of b at level j.

An *E-semantic tree* is a truth-value tree τ defined inductively as follows.

T1. The root of τ is unlabeled and at level 0.

T2. A node N of τ at level $k-1$ has one or two children according to the following rules. (Compare with Theorem 3.)

T2a. If B_k is reducible (I_N), say $B_k \to C$, then N has one child labeled $I_N(C)$.

T2b. If B_k is irreducible (I_N) and of the form $(t = t)$, then N has one child labeled **T**.

T2c. If B_k is irreducible (I_N) and not of the form $(t = t)$, then N has two children, the left one is labeled **T** and the right one **F**.

Theorem 3 tells us that every I_N in an E-semantic tree is an E-interpretation on $\mathscr{B}_k$ where k is the level of N. Furthermore, the set of all interpretations I_b on the branches of an E-semantic tree is identical to the set of all E-interpretations on $\mathscr{B}$.

If S is a set of clauses, then $\tau(S)$ will denote an E-semantic tree over the Herbrand base $\mathscr{B}$ of S.

A node N of $\tau(S)$ is a *failure node* if I_N falsifies some ground instance C_N of a clause in S, but if M is an ancestor of N, then I_M does not falsify any ground instance of a clause in S.

LEMMA 1. *If N is a failure node at level k then*

(1) *if $I_N(B_k) = \mathbf{F}$ then B_k is a literal of C_N,*

(2) *if $I_N(B_k) = \mathbf{T}$ then $\sim B_k$ is a literal of C_N.*

Proof. Since $I_N(C_N) = \mathbf{F}$, it follows that if A is an atom of C_N, then $A \leqq B_k$. If every atom of C_N were $< B_k$, then we could find a strict ancestor M of N for which $I_M(C_N) = \mathbf{F}$ and N would not be a failure node. Thus B_k occurs as an atom in C_N. If $I_N(B_k) = \mathbf{F}$, then B_k occurs as a literal in C_N; otherwise, $\sim B_k$ occurs as a literal in C_N. □

An E-model for a set S is an E-interpretation I defined on the Herbrand base of S such that $I(C\theta) = \mathbf{T}$ for every ground instance $C\theta$ of a clause C in S. If S has no E-model, then every branch of $\tau(S)$ will have a failure node.

We will next describe how to assign a nonnegative integer to each failure node N which somehow measures the "reducibility" of the clauses which are falsified by I_N. For each failure node N of $\tau(S)$, let S_N be the set of ground instances of clauses in S that are falsified by I_N. The definition of failure node shows that S_N is not empty. If B_j is an atom in one of the clauses of S_N, define

$$\lambda_N(B_j) = \begin{cases} 0 & \text{if } B_j \text{ is irreducible } (I_N), \\ j & \text{otherwise.} \end{cases}$$

If L is a literal in one of the clauses of S_N and A is the atom of L, define $\lambda_N(L) = \lambda_N(A)$. Finally, if $C = L_1 \vee L_2 \vee \cdots \vee L_k$ is a clause in S_N, define $\lambda_N(C) = \sum_{i=1}^{k} \lambda_N(L_i)$. This completes the definition of a function λ_N which maps the elements of C to nonnegative integers and is such that $\lambda_N(C) = 0$ if and only if C is irreducible. For each failure node N, let C_N be a clause in S_N at which a minimum of λ_N occurs. That is, $\lambda_N(C_N) \leqq \lambda_N(C)$ for every $C \in S_N$. We assign to node N the nonnegative integer $\lambda_N(C_N)$. Now C_N is a ground instance of some clause D_N in S, that is, $C_N = D_N\theta$. This substitution *θ may and will be taken to be irreducible* (I_N), because if it were not, say $\theta \to^+ \theta^*$ where θ^* is irreducible (I_N), we may select $C_N = D_N\theta^*$ since by Theorem 4, $I_N(D_N\theta) = I_N(D_N\theta^*) = \mathbf{F}$ so $D_N\theta^* \in S_N$ and $\lambda_N(D_N\theta^*) \leqq \lambda_N(D_N\theta)$.

If S has no E-model, then the *closed semantic tree for S*, $\tau_*(S)$, will be the numbered truth-value tree consisting of that portion of $\tau(S)$ which includes all nodes

that are at or above the failure nodes and has each failure node N labeled with $\tau'_*(N) = \lambda_N(C_N)$. An example is shown in Fig. 1.

An *R failure node* is a failure node N such that $\lambda_N(C_N) = 0$, that is, C_N is irreducible (I_N). A *P failure node* is a failure node which is not an R failure node. The bottom two leaf nodes of Fig. 1 are P failure nodes and the top one is an R failure node.

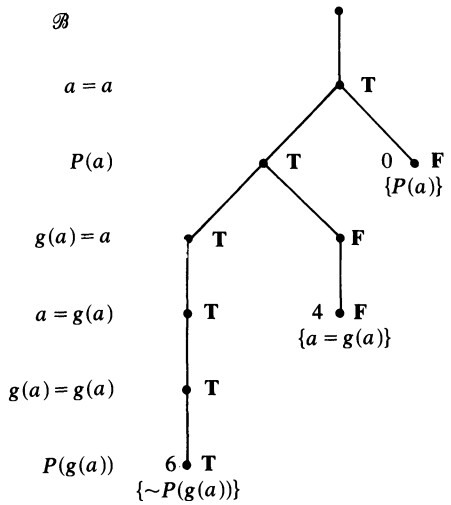


FIG. 1. *The closed semantic tree $\tau_*(S)$ for the set $S = \{x = g(x), P(a), \sim P(g(a))\}$. The elements of $\mathcal{B}$ are shown in order in the left-hand column. The truth-value of each node is shown on its right and the numeric value $\tau'_*(N)$ is shown on the left of each failure node. Although not part of $\tau_*(S)$, the clause C_N is shown in braces under each failure node N.*

A *reduction node* is a non leaf node N of $\tau_*(S)$ at level k such that B_k is an equality, $\tau_*(N) = \mathbf{T}$, and N has a brother. From this it follows that B_k is irreducible (I_M) where M is the father of N, and B_k is not of the form $(t = t)$. The only reduction node in Fig. 1 is the node on the third level which is labeled **T**.

A *resolution inference node* is a node N of $\tau_*(S)$ such that all the children of N are R failure nodes. There are no resolution inference nodes in Fig. 1.

A *paramodulation inference node* is a P failure node N such that every reduction node ancestor of N has a brother which is an R failure node. In Fig. 1 the node with number label 4 is a paramodulation inference node.

LEMMA 2. *If S does not contain the empty clause and has no E-model, then $\tau_*(S)$ has either a resolution or a paramodulation inference node.*

Proof. Suppose $\tau_*(S)$ has no resolution inference nodes. Let's start at the root of $\tau_*(S)$ and take a walk down $\tau_*(S)$. As we walk we will obey the following rules.

A1. We will never walk on an R failure node.

A2. We will never walk on a reduction node unless its brother is an R failure node.

This walk must end at a paramodulation inference node. For, if we are not at a failure node we can always keep walking since $\tau_*(S)$ has no resolution inference nodes. Therefore, when we stop we will be at a P failure node and it will be a paramodulation inference node because of condition A2. □

Let $S^a = S \cup \{x = x\}$. Recall that numbered truth-value trees were endowed with a partial order at the beginning of this section.

THEOREM 6. *If S has no E-model and $\tau_*(S)$ has a resolution inference node, then there is a resolvent C of clauses or factors of clauses in S^a such that $\tau_*(S \cup C) < \tau_*(S)$.*

Proof. We just give an outline since this result is well known in the context of ordinary semantic trees [see 3].

Let N be a resolution inference node in $\tau_*(S)$. If N has two children, let these children be nodes L and M. Then there are ground instances C_L and C_M of clauses of S such that $I_L(C_L) = \mathbf{F}$ and $I_M(C_M) = \mathbf{F}$. If N is at level $k-1$, then L and M are at level k. Thus $I_L(B_k) = \mathbf{T}$ and $I_M(B_k) = \mathbf{F}$. It follows from Lemma 1 that $\sim B_k$ occurs in C_L and B_k occurs in C_M. Thus C_L and C_M or factors of C_L and C_M will resolve to produce a clause C' which does not contain B_k and therefore $I_N(C') = \mathbf{F}$. This resolution can be lifted to the general level to obtain a clause C of which C' is a ground instance. In $\tau_*(S \cup C)$ node N or one of its ancestors is a failure node. Thus $\tau_*(S \cup C) < \tau_*(S)$.

If N has one child, say node M, and N is at level $k-1$, then B_k is of the form $(t = t)$. There is a ground instance C_M of a clause of S such that $I_M(C_M) = \mathbf{F}$. It follows that C_M contains $(t \neq t)$. Thus C_M or a factor of C_M resolves with $(t = t)$ to produce a clause C' which does not contain $(t \neq t)$. This resolution can be lifted to a general level resolution with $(x = x)$, and we proceed as above. □

In the proof of the following theorem we will use Var (θ) to denote the set of variables $\{x_1, \cdots, x_k\}$ appearing on the left of elements in a substitution $\theta = \{x_1 \leftarrow t_1, \cdots, x_k \leftarrow t_k\}$.

THEOREM 7. *If $\tau_*(S)$ has a paramodulation inference node, then there is a paramodulant C of clauses or factors of clauses in S, or a clause C obtained by reversing an equality (possibly contained in an inequality) in some clause of S, such that $\tau_*(S \cup C) < \tau_*(S)$.*

Proof. Let N be a paramodulation inference node of $\tau_*(S)$ and let C_2 be a clause in S such that $C_2\theta = C_N$ for some irreducible (I_N) substitution θ. Since N is a P failure node, $C_2\theta$ is reducible (I_N). Let E be a clause such that $C_2\theta \rightarrow E$.

Case 1. Suppose $C_2\theta \rightarrow E$ by reason of F1. Then C_2 contains an equality $(t = s)$ such that $t\theta < s\theta$ and E is identical to $C_2\theta$ except that $(t\theta = s\theta)$ has been reversed. Let C be the same as C_2 except that the equality $(t = s)$ occurs as $(s = t)$ in C. Then $E = C\theta$, and $I_N(C\theta) = I_N(C_2\theta) = \mathbf{F}$. Also $\lambda_N(C\theta) < \lambda_N(C_2\theta)$. It follows that $\tau_*(S \cup C) < \tau_*(S)$.

Case 2. (Refer to Fig. 2) Suppose $C_2\theta \rightarrow E$ by reason of F2. Then there is an atom A of $C_2\theta$ and a ground equality $(s_1 = t_1)$ such that s_1 is a subterm of A, $t_1 < s_1$, $(s_1 = t_1) < A$, $I_N(s_1 = t_1) = T$, $(s_1 = t_1)$ is irreducible (I_N) and $E = C_2\theta[n \leftarrow t_1]$ where $n \in \text{dom } C_2\theta$ is such that $C_2\theta/n = s_1$. Since θ is irreducible (I_N), $n \in \text{rdom } C_2$. Since $(s_1 = t_1) < A$, $I_N(s_1 = t_1) = \mathbf{T}$, and $(s_1 = t_1)$ is irreducible (I_N), it follows that N has a reduction node ancestor M at level k, say, and $B_k = (s_1 = t_1)$. Since N is a paramodulation inference node, the brother K of M is an R failure node. Let $C_1 \in S$ be such that $C_1\psi = C_K$ for some irreducible (I_K) substitution ψ. Now $I_K(s_1 = t_1) = \mathbf{F}$ and it follows from Lemma 1 that $(s_1 = t_1)$ is a literal of C_K.

If $(s_1 = t_1)$ occurs twice or more in C_k, then there will be literals $(s^1 = t^1), \cdots, (s^k = t^k)$ in C_1 such that $(s_1 = t_1) = (s^i\psi = t^i\psi)$ for $1 \leqq i \leqq k$. Thus $(s^1 = t^1), \cdots, (s^k = t^k)$ are unifiable. Let σ be their mgu and let ϕ be such that $\psi = \sigma\phi$. Then

$$C_1^* = C_1\sigma - \{(s^2\sigma = t^2\sigma) \vee \cdots \vee (s^k\sigma = t^k\sigma)\}$$

is a factor of C_1 such that $C_1^*\phi$ is identical to $C_1\psi$ except that $C_1^*\phi$ has only one occurrence of $(s_1 = t_1)$. Now ϕ may be chosen such that Var $(\phi) \cap$ Var (σ) is empty (to see this, note that the unification algorithm [3, p. 77] implies that none of the variables in Var (σ) occur in any of the terms of σ) and it follows from the definition

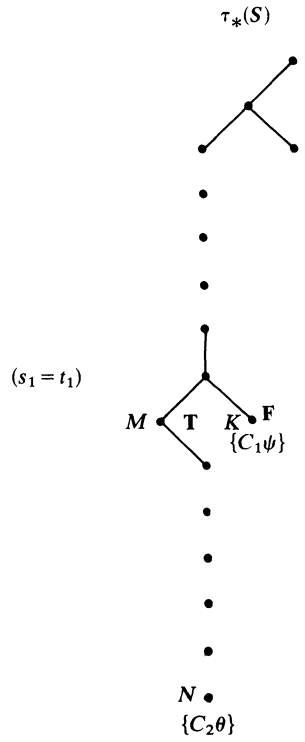


FIG. 2. *The situation in the proof of Theorem* 7.

of composition of substitutions [3, p. 76] that all the terms of ϕ are also terms of ψ. Thus ϕ is irreducible (I_k). We may assume, therefore, that $(s_1 = t_1)$ occurs only once in $C_1\psi$.

Thus we have $C_1 = (s = t) \vee C_1'$ where $(s_1 = t_1) = (s\psi = t\psi)$ and $I_K(C_1'\psi) = I_M(C_1'\psi) = I_N(C_1'\psi) = \mathbf{F}$ since all the atoms in $C_1'\psi$ are less than $(s_1 = t_1)$, and I_M, I_N and I_K have identical values on such atoms. Furthermore, $C_1'\psi$ is irreducible (I_M, I_N or I_K) since K is an R failure node.

Let C' be the ground paramodulant

$$C' = C_2\theta[n \leftarrow t\psi] \vee C_1'\psi$$

of $C_1\psi$ into node n of $C_2\theta$. If C_1 and C_2 have had their variables standardized apart, and we assume they have, then we may think of C' as a paramodulant of $C_1(\theta \cup \psi)$ into node n of $C_2(\theta \cup \psi)$. By the paramodulation lifting lemma, there is a paramodulant C of C_1 into node n of C_2 such that C' is a ground instance, say $C\alpha$, of C. Since $C_1'\psi$ is irreducible (I_N), $\lambda_N(C') = \lambda_N(C_2\theta[n \leftarrow t\psi]) < \lambda_N(C_2\theta) = \lambda_N(C_N)$. It follows that $\tau_*(S \cup C) < \tau_*(S)$. □

Note that in Theorems 6 and 7 all resolutions and factoring were done with irreducible clauses and clause $C_1\psi$ which was used to paramodulate into $C_2\theta$ was also irreducible. Furthermore, equality reversal was only performed on an otherwise irreducible equality.

THEOREM 8. *If S has no E-model then S has a refutation by factoring, equality reversal, resolution and paramodulation.*

Proof. This follows from Lemma 2 and Theorems 6 and 7. □

This theorem implies that resolution and paramodulation (including factoring) is complete without the functionally reflexive axioms.

7. Unnecessary clauses. Suppose S has no E-model. A clause $C \in S$ is *unnecessary* if for every failure node N of $\tau_*(S)$, there is a clause in S_N with minimum λ_N value which is an instance of a clause in $S-\{C\}$; that is, C_N may be chosen in $S-\{C\}$. It is clear that an unnecessary clause may be deleted from S without affecting $\tau_*(S)$. We show here that subsumed clauses and most clauses which can be simplified are unnecessary.

THEOREM 9. *If C_1 is subsumed by C_2, then C_1 is unnecessary.*

Proof. Let $C_2\sigma \subseteq C_1$. Let N be a failure node of $\tau_*(S)$ and suppose $C_N = C_1\theta$ for some θ. *Then* $I_N(C_1\theta) = \mathbf{F}$. Since $C_2\sigma\theta \subseteq C_1\theta$, it follows that $I_N(C_2\sigma\theta) = \mathbf{F}$ and $\lambda_N(C_2\sigma\theta) \leqq \lambda_N(C_1\theta)$. Thus C_N may be chosen as $C_2\sigma\theta$, and C_1 is unnecessary.

THEOREM 10. *Suppose C_1, C_2 and $(s = t)$ are clauses in S such that for some substitution σ, C_1 contains an atom A which has $s\sigma$ as a subterm, that is, $C_1 = C_1[s\sigma]$. Suppose $C_2 = C_1[t\sigma]$ and $s\sigma > t\sigma$. Suppose A is not of one of the forms $(\sigma\sigma = p)$ or $(p = s\sigma)$ where p is a term such that $p\phi < t\sigma\phi$ for some ground instances, $p\phi$ and $t\sigma\phi$, of p and $t\sigma$. Then C_1 is unnecessary.*

Proof. Suppose there is a failure node N of $\tau_*(S)$ such that $C_N = C_1\theta$. Then $C_1\theta = C_1\theta[s\sigma\theta]$, $C_2\theta = C_1\theta[t\sigma\theta]$ and $s\sigma\theta > t\sigma\theta$. Thus $\lambda_N(C_2\theta) \leqq \lambda_N(C_N)$.

If $(s\sigma\theta = t\sigma\theta) \geqq A\theta$, then by O7, $A\theta$ must be of one of the forms $(s\sigma\theta = q)$ or $(q = s\sigma\theta)$ where $q \leqq t\sigma\theta$. It follows that A must be of one of the forms $(s\sigma = p)$ or $(p = s\sigma)$ where $p\theta = q \leqq t\sigma\theta$. But this was ruled out in the hypothesis. Therefore, $(s\sigma\theta = t\sigma\theta) < A\theta$.

If $I_N(s\sigma\theta = t\sigma\theta)$ were false, then since $(s\sigma\theta = t\sigma\theta)$ is a ground instance of the clause $(s = t) \in S$ and $\lambda_N(s\sigma\theta = t\sigma\theta) < \lambda_N(A\theta) \leqq \lambda_N(C_1\theta)$, it would follow that $C_1\theta$ is not a clause of minimum λ_N value in S_N. Thus $I_N(s\sigma\theta = t\sigma\theta) = \mathbf{T}$.

Suppose N is at level k in $\tau_*(S)$. Then I_N is an E-interpretation on $\mathcal{B}_k$. Thus $I_N(C_2\theta) = I_N(C_1\theta) = \mathbf{F}$. It follows that C_N may be reselected as $C_2\theta$ and, therefore, C_1 is unnecessary. □

As a practical matter, the restriction on the form of A in Theorem 8 is probably not needed. To see this, consider first the case in which $C_1 = C_1' \vee (s\sigma = p)$. Then $C_2 = C_1' \vee (t\sigma = p)$ and a paramodulation of C_2 into $(s = t)$ yields C_1. Thus, if C_1 is discarded it can be recovered and is unnecessary in that sense. Consider also the case in which $C_1 = C_1' \vee (s\sigma \neq p)$. Then $C_2 = C_1' \vee (t\sigma \neq p)$. Now let L be some literal which can resolve with $(s\sigma \neq p)$. Then $L\theta = (s\sigma\theta = p\theta)$ for some substitution θ. Paramodulation of $(s = t)$ into $L\theta$ yields $(t\sigma\theta = p\theta)$ which resolves with $(t\sigma \neq p)$. Thus no resolutions are lost if C_1 is discarded and it again appears to be unnecessary. For these reasons, implementors would very likely retain completeness if they ignored the restrictions on the form of A in Theorem 8.

8. Practical considerations and examples. Putting these ideas together and considering carefully the proof of Theorem 5, we find that a complete theorem prover for first-order predicate calculus with equality could consist of resolution, paramodulation, factoring, equality reversal, simplification, and subsumption removal with the following restrictions.

P1. Simplification and subsumption are performed first, since they do not increase the size of S.

P2. No paramodulation into variables.

P3. All paramodulations replace s by t where $t \not\succ s$.

To be more precise (but ignoring the restrictions of Theorem 8), suppose $RP(S)$

is the set of clauses obtained from S by resolution, paramodulation, equality reversal and factoring with subsumed clauses and clauses which have been simplified, deleted. Suppose further that in forming $RP(S)$ restrictions P1–P3 are obeyed. Then $RP^n(S)$ contains the empty clause for some n, where $RP^n(S) = RP(RP^{n-1}(S))$. This follows from our theorems, since it is clear that $\tau_*(RP(S)) < \tau_*(S)$.

If this procedure was applied to a set S consisting entirely of equality units, then it would be very similar to the process that Knuth and Bendix [15] used for producing new reductions from old since their process consisted of paramodulations into nonvariables, simplifications, and deletion of clauses subsumed by $(x = x)$.

As a hand example, let us prove that a two-element group is commutative. We write $s \to t$ if the equality $(s = t)$ is such that $s > t$. Our set of operators is $O = \{e, a, b, c, d, {}^{-1}, \cdot\}$ ordered as listed. Let $\lambda(f) = 1$ for every $f \in O$ and use the Knuth–Bendix order described in § 3. Clauses 2 through 11 below are the complete set of reductions found for groups by Knuth and Bendix except that clause 11 has been reversed in order to conform to our ordering. (Thus reductions 2–11 will not be a complete set of reductions in our order.) Clause 12 states that the group consists of two elements and clause 13 that it is not commutative. We use the abbreviations P, S, and R for paramodulation, simplification and resolution, respectively. The proof has been abbreviated somewhat by omitting certain rather obvious steps.

1. $x = x$
2. $x \cdot e \to x$
3. $e \cdot x \to x$
4. $x \cdot x^{-1} \to e$
5. $x^{-1} \cdot x \to e$
6. $e^{-1} \to e$
7. $x^{-1-1} \to x$
8. $(x \cdot y) \cdot z \to x \cdot (y \cdot z)$
9. $x \cdot (x^{-1} \cdot y) \to y$
10. $x^{-1} \cdot (x \cdot y) \to y$
11. $y^{-1} \cdot x^{-1} \to (x \cdot y)^{-1}$
12. $x = a \vee x = b$
13. $c \cdot d \neq d \cdot c$
14. $a \to e \vee e^{-1} \to b$ P of 12.1 into 6
15. $a \to e \vee b \to e$ S of 14

    delete 14
16. $x = e \vee x = b \vee b \to e$ P of 15.1 into 12
17. $e \cdot d \neq d \cdot e \vee c \to b \vee b \to e$ P of 16.1 into 13
18. $d \neq d \vee c \to b \vee b \to e$ S of 17

    delete 17
19. $c \to b \vee b \to e$ R of 18.1 and 1

    delete 18 Subsumed by 19
20. $b \cdot d \neq d \cdot b \vee b \to e$ P of 19.1 into 13
21. $b \cdot e \neq e \cdot b \vee d \to b \vee b \to e$ P of 16.1 into 20
22. $b \neq b \vee d \to b \vee b \to e$ S of 21

    delete 21
23. $d \to b \vee b \to e$ R of 22.1 and 1

    delete 22 Subsumed by 23
24. $b \cdot b \neq b \cdot b \vee b \to e$ P of 23.1 into 20
25. $b \to e$ R of 24.1 and 1

    delete 24, 23, 20, 19, 16 and 15 Subsumed by 25.

26. $x = a \vee x \to e$ S of 12
    delete 12
27. $e \cdot d \neq d \cdot e \vee c \to a$ P of 26.2 into 13
28. $d \neq d \vee c \to a$ S of 27
    delete 27
29. $c \to a$ R of 28 and 1
    delete 28 Subsumed by 29
30. $a \cdot d \neq d \cdot a$ S of 13
    delete 13
31. $a \cdot e \neq e \cdot a \vee d \to a$ P of 26.2 into 30
32. $d \to a$ S of 31 followed by R with 1
    delete 31 Subsumed by 32
33. $a \cdot a \neq a \cdot a$ S of 30
34. $\square$ R of 33 and 1

In this example, note that the many deletions made possible by subsumption and simplification kept the total number of clauses down to a reasonable level. This is in contrast to previous provers for first-order logic with equality whose main difficulty was that space would quickly be exhausted. It is therefore hoped that a prover based on the ideas of this paper will be better than previous complete ones.

Further research into automatic theorem proving in first-order logic with equality could proceed in many directions. One could try to remove the restriction that the ordering of terms must be order-isomorphic to the positive integers. This could involve both extending König's lemma [24, p. 69] to well-ordered sets with limit ordinals and somehow dealing with trees constructed from such sets. Many strategies for improving the power of resolution provers for logic without equality have been developed [3]. Which of these are useful in provers which use equality? See [24] for a description of some work in this direction. Special unification methods such as associative-commutative unification might be incorporated and completeness results sought [8], [12], [13], [14], [19], [22], [27], [36]. Finally, experience has shown that in implementations of the Knuth–Bendix procedure, the majority of execution time is spent in simplification. One could consider the possibility of "compiling" the simplifiers and producing code that will speed the simplification process.

REFERENCES

[1] D. BRAND, *Proving theorems with the modification method*, this Journal, 4 (1975), pp. 412–430.
[2] T. C. BROWN, JR., *A structured design-method for specialized proof procedures*, Ph.D. Dissertation, California Institute of Technology, Pasadena, CA, 1975.
[3] C. L. CHANG AND R. C. T. LEE, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.
[4] P. M. COHN, *Universal Algebra*, Harper & Row, New York, 1965.
[5] N. DERSHOWITZ, *A note on simplification orderings*, Tech. Rep. R-79-968, Dept. of Computer Science, Univ. Illinois at Urbana–Champaign, Urbana, IL, April, 1979.
[6] ———, *Orderings for term-rewriting systems*, Tech. Rep. R-79-987, Dept. of Computer Science, Univ. Illinois at Urbana–Champaign, Urbana, IL, August, 1979.
[7] N. DERSHOWITZ AND Z. MANNA, *Proving termination with multiset orderings*, Comm. ACM, 22 (1979), pp. 465–476.
[8] M. J. FAY, *First-order unification in an equational theory*, Presented at the Fourth Workshop on Automated Deduction, Austin, TX, Feb. 1–3, 1979.
[9] J. R. GUARD, F. C. OGLESBY, J. H. BENNETT AND L. G. SETTLE, *Semi-automated mathematics*, J. Assoc. Comput. Mach., 16 (1969), pp. 49–62.

[10] G. HUET, *A complete proof of correctness of the Knuth–Bendix completion algorithm*, INRIA and SRI International, 1980.
[11] ———, *Confluent reductions: Abstract properties and applications to term rewriting systems*, J. Assoc. Comput. Mach., 27 (1980), pp. 797–821.
[12] G. HUET AND D. C. OPPEN, *Equations and rewrite rules: A survey*, Technical Report CSL-11, SRI International, Menlo Park, CA, January, 1980.
[13] J. HULLOT, *A catalogue of canonical term rewriting systems*, Tech. Rep. CSL-113, SRI International, Menlo Park, CA, April, 1980.
[14] ———, *Canonical forms and unification*, Tech. Rep. CSL-114, SRI International, Menlo Park, CA, April, 1980.
[15] D. E. KNUTH AND P. B. BENDIX, *Simple word problems in universal algebras*, in Computational Problems in Abstract Algebras, J. Leech, ed., Pergamon Press, New York, 1970, pp. 263–297.
[16] R. KOWALSKI AND P. J. HAYES, *Semantic trees in automatic theorem-proving*, in Machine Intelligence 4, B. Meltzer and D. Michie, eds., American Elsevier, New York, 1969, pp. 87–101.
[17] D. S. LANKFORD, *Canonical algebraic simplification in computational logic*, Memo ATP-25, Dept. Mathematics, Univ. Texas at Austin, Austin, TX, May, 1975.
[18] ———, *Canonical inference*, Technical Report ATP-25, Dept. Mathematics, Univ. Texas at Austin, Austin, TX, Dec., 1975.
[19] ———, *Mechanical theorem proving in field theory*, Technical Report MTP-2, Dept. Mathematics, Louisiana Tech Univ., Ruston, LA, January, 1979.
[20] ———, *On proving term rewriting systems are Noetherian*, Memo MTP-3, Dept. of Mathematics, Louisiana Tech Univ., Ruston, LA, May, 1979.
[21] D. S. LANKFORD AND A. M. BALLANTYNE, *The refutation completeness of blocked permutative narrowing and resolution*, Presented at the Fourth Workshop on Automated Deduction, Austin, TX, Feb. 1–3, 1979.
[22] ———, *Decision procedures for simple equational theories with commutative-associative axioms: Complete sets of commutative-associative reductions*, Tech. Rep., Dept. Mathematics, Univ. Texas at Austin, Austin, TX, August, 1977.
[23] L. S. LEVY, *Discrete Structures of Computer Science*, John Wiley, New York, 1980.
[24] D. W. LOVELAND, *Automated Theorem Proving: A Logical Basis*, North-Holland, Amsterdam, 1978.
[25] Z. MANNA, *Mathematical Theory of Computation*, McGraw-Hill, New York, 1974.
[26] A. J. NEVINS, *A human oriented logic for automatic theorem-proving*, J. Assoc. Comput. Mach., 21 (1974), pp. 606–621.
[27] G. E. PETERSON AND M. E. STICKEL, *Complete sets of reductions for some equational theories*, J. Assoc. Comput. Mach., 22 (1981), pp. 233–264.
[28] D. A. PLAISTED, *Well-founded ordering for proving termination of systems of rewrite rules*, Tech. Rep. R-78-932, Dept. Computer Science, Univ. of Illinois at Urbana–Champaign, Urbana, IL, July, 1978.
[29] ———, *A recursively defined ordering for proving termination of term rewriting systems*, Tech. Rep. R-78-943, Dept. Computer Science, Univ. of Illinois at Urbana–Champaign, Sept., 1978.
[30] G. D. PLOTKIN, *Building-in equational theories*, in Machine Intelligence 7, B. Meltzer and D. Michie, eds., John Wiley, New York, 1972, pp. 73–89.
[31] P. RAULEFS, J. SICKMANN, P. SZABO AND E. UNVERICHT, *A short survey on the state of the art in matching and unification problems*, SIGSAM Bulletin, 13 (1979), pp. 14–20.
[32] G. A. ROBINSON AND L. WOS, *Paramodulation and theorem proving in first order theories with equality*, in Machine Intelligence 4, B. Meltzer and D. Michie, eds., American Elsevier, New York, 1969, pp. 135–150.
[33] B. K. ROSEN, *Tree-manipulating systems and Church–Rosser theorems*, J. Assoc. Comput. Mach., 20 (1973), pp. 160–187.
[34] J. R. SLAGLE, *Automatic theorem proving with built-in theories including equality, partial ordering, and sets*, J. Assoc. Comput. Mach., 19 (1972), pp. 120–135.
[35] ———, *Automated theorem-proving for theories with simplifiers, commutativity, and associativity*, J. Assoc. Comput. Mach., 21 (1974), pp. 622–642.
[36] M. E. STICKEL, *A unification algorithm for associative-commutative functions*, J. Assoc. Comput. Mach., 28 (1981), pp. 423–434.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0007 $01.25/0

CHARACTERIZATION OF DIVISION ALGEBRAS OF MINIMAL RANK AND THE STRUCTURE OF THEIR ALGORITHM VARIETIES*

HANS F. DE GROOTE†

Abstract. The purpose of this paper is to show that every finite dimensional division algebra of minimal rank, i.e., of minimal complexity with respect to the noncommutative model of computation, is a finite simple field extension. Moreover, we investigate the structure of the variety of optimal algorithms for the computation of the multiplication in such fields modulo the isotropy group of the problem.

Key words. algebras, minimal rank, division algebras, simple field extensions, algorithm varieties

Introduction. Let $\mathcal{A}$ be a finite dimensional algebra over a field k. We will assume always that $\mathcal{A}$ is associative and has a unit element. Let $\mathcal{A}^*$ be the dual of the k-vectorspace $\mathcal{A}$. A (commutative) algorithm of length R for the multiplication in $\mathcal{A}$ is a tuple $(U_1, V_1, w_1, \cdots, U_R, V_R, w_R)$, where $U_\rho, V_\rho \in \mathcal{A}^* \times \mathcal{A}^*$ and $w_\rho \in \mathcal{A}$ $(\rho = 1, \cdots, R)$ such that for all $x, y \in \mathcal{A}$

$$xy = \sum_{\rho=1}^{R} U_\rho(x, y) V_\rho(x, y) w_\rho$$

holds. The minimal R possible in such a representation is called the complexity $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$. $\mathcal{L}(\mathcal{A})$ is the number of essential multiplications needed for the computation of xy from x and y.

In this paper we use a somewhat coarser but more feasible computational model: we shall consider only those algorithms for which the U_ρ's and V_ρ's have the special form

$$U_\rho = (u_\rho, 0), \qquad V_\rho = (0, v_\rho).$$

Then

$$xy = \sum_{\rho=1}^{R} u_\rho(x) v_\rho(y) w_\rho$$

for all $x, y \in \mathcal{A}$; this representation of the multiplication in $\mathcal{A}$ is equivalent to a representation of the tensor $t_{\mathcal{A}} \in \mathcal{A}^* \otimes \mathcal{A}^* \otimes \mathcal{A}$ of $\mathcal{A}$ as

$$t_{\mathcal{A}} = \sum_{\rho=1}^{R} u_\rho \otimes v_\rho \otimes w_\rho.$$

The minimal R possible in such a representation is called the rank of the tensor $t_{\mathcal{A}}$ or the *rank of* $\mathcal{A}$ for short, denoted by $\mathrm{rk}(\mathcal{A})$. This point of view goes back to V. Strassen, and the reader who is not familiar with these notions should compare [11] for further details. Obviously we have $\mathrm{rk}(\mathcal{A}) \geqq \mathcal{L}(\mathcal{A})$, and it is not difficult to prove that $\mathrm{rk}(\mathcal{A}) \leqq 2\mathcal{L}(\mathcal{A})$.

Recently, A. Alder and V. Strassen proved a general lower bound for the complexity of algebras ([1]):

$$\mathcal{L}(\mathcal{A}) \geqq 2 \dim \mathcal{A} - \#\,\mathrm{mspec}(\mathcal{A}),$$

where $\mathrm{mspec}(\mathcal{A})$ is the set of two-sided maximal ideals of $\mathcal{A}$ ($\mathcal{A}$ itself is not considered as an ideal). This lower bound contains almost all known lower bounds for the

* Received by the editors July 8, 1981, and in revised form March 25, 1982.
† Fachbereich Mathematik, Johann Wolfgang Goethe-Universität, Frankfurt am Main, Germany.

complexity of concrete algebras. Of course $2 \dim \mathscr{A} - \#\text{mspec}(\mathscr{A})$ is also a lower bound for rk $(\mathscr{A})$.

Now it is quite natural to ask for which algebras this lower bound is exact, i.e., which are the algebras $\mathscr{A}$ that satisfy

$$\text{rk}(\mathscr{A}) = 2 \dim \mathscr{A} - \#\text{mspec}(\mathscr{A}).$$

An algebra with this property is said to have minimal rank.

The problem of classifying algebras of minimal rank has two important aspects. The first is a mathematical one: is it possible to describe such algebras by means of their internal structure? If the answer is positive (at least for some special classes of algebras), then this has applications in complexity theory: we will obtain new lower bounds for the rank of algebras whose structure differs from that which is determined by the minimal rank condition. This is the second aspect of our problem.

This paper is a starting point for the program described above. We investigate the structure of finite dimensional division algebras of minimal rank and show that these are the finite simple field extensions.[1]

Another question, intimately related to the rank problem, is that of determining the structure of the variety of optimal algorithms for a given bilinear computational problem. We will do this for the algorithm variety of finite simple field extensions K/k (provided the base field k is large enough), and in particular we will determine the orbit space of this variety under the action of the isotropy group of K/k in the sense of [4]. Furthermore, we shall compare this with the orbit space of optimal algorithms for polynomial multiplication.

We will now fix some of the mathematical terminology used in this paper.

An element a of an algebra is called a unit if a has a multiplicative inverse. If $\mathscr{A}$ is an algebra, $a \in \mathscr{A}$, then $L_a(R_a)$ denotes the operator of left (right) multiplication with a.

Let V be a vectorspace over a field k. If $x \in V$ we will write $[x]$ for the subspace of V generated by x. End (V) denotes the k-algebra of all endomorphisms, $Gl(V)$ the group of all automorphisms of the vectorspace V. If $\varphi: V \to W$ is a linear mapping, then $\varphi^*: W^* \to V^*$ denotes the adjoint of φ.

If k and K are fields, then K/k means that K is an extension of k; $[K:k]$ denotes the degree of the extension K/k, i.e., the dimension of the k-vectorspace K. Ω denotes the set of primitive elements of a simple field extension K/k. γ_n denotes the permutation group of n elements and $\#M$ the cardinality of the set M, and $\square$ marks the end of a proof. In general the terminology is that of [4].

1. Division algebras with minimal rank.

1.1. Preliminaries. Let $\Phi: U \times V \to W$ be a bilinear mapping between finite dimensional vectorspaces over a field k, $t \in U^* \otimes V^* \otimes W$ the tensor corresponding to Φ. In [5] we introduced the notion "layer of t". In order to make this notion more flexible we would like to generalize it to a coordinate-free one and show up its relation to the left and right multiplication in case Φ is the multiplication in a finite dimensional algebra.

[1] If $\mathscr{A} = \mathscr{A}_1 \oplus \cdots \oplus \mathscr{A}_r$ is a direct sum of division algebras, then $\mathscr{A}$ has minimal rank if and only if all the $\mathscr{A}_\rho$'s have [7]. However, one can derive more generally from a preliminary version of [1] that if $\mathscr{A}$ is semisimple and $\mathscr{A} = \mathscr{A}_1 \oplus \cdots \oplus \mathscr{A}_r$ is a decomposition of $\mathscr{A}$ into simple algebras, then $\mathscr{A}$ has minimal rank if and only if all the $\mathscr{A}_\rho$'s have.

Let

$$t = \sum_{\rho=1}^{R} u_\rho \otimes v_\rho \otimes w_\rho \tag{1.1.1}$$

be a representation of t as sum of rank-one tensors $u_\rho \otimes v_\rho \otimes w_\rho \in U^* \otimes V^* \otimes W$.

For $x \in U$, $y \in V$, $\chi \in W^*$ we define the layers of t with respect to x, y, and χ respectively as

$$t_x := \sum_{\rho=1}^{R} u_\rho(x) v_\rho \otimes w_\rho, \quad t^y := \sum_{\rho=1}^{R} v_\rho(y) u_\rho \otimes w_\rho, \quad t_\chi := \sum_{\rho=1}^{R} \chi(w_\rho) u_\rho \otimes v_\rho.$$

Note that $t_x \in V^* \otimes W$, $t^y \in U^* \otimes W$, and $t_\chi \in U^* \otimes V^*$ can be interpreted as linear mappings

$$t_x : V \to W, \quad t^y : U \to W, \quad t_\chi : U \to V^*.$$

Note further that the definition of these layers is independent of the choice of the representation (1.1.1) of t because for all $(x, y) \in U \times V$ we have

$$t_x(y) = \Phi(x, y), \qquad t^y(x) = \Phi(x, y),$$

$$t_\chi(x)(y) = \chi(\Phi(x, y)).$$

This also shows that in case Φ is the multiplication in a finite dimensional algebra $\mathcal{A}$, then

$$t_x = L_x \quad \text{and} \quad t^y = R_y,$$

where L_x, R_y denote left and right multiplication in $\mathcal{A}$ respectively. In what follows we also need a relation between t_χ and the left multiplication in $\mathcal{A}$. This is a somewhat subtle question, and we will answer it for the special case of a simple algebra $\mathcal{A}$ with unit.

PROPOSITION 1.1. *Let $\mathcal{A}$ be a finite dimensional simple k-algebra with unit. Then there is a vectorspace isomorphism $S : \mathcal{A} \to \mathcal{A}^*$ such that*

(i) $$L_x^* = S R_x S^{-1} \quad \textit{for all } x \in \mathcal{A},$$

(ii) $$t_\chi = S L_{S^{-1}\chi} \quad \textit{for all } \chi \in \mathcal{A}^*.$$

Proof. Let δ be any vectorspace isomorphism $\delta : \mathcal{A} \to \mathcal{A}^*$. Consider the subspaces

$$\mathcal{R} := \{R_x | x \in \mathcal{A}\}$$

and

$$\mathcal{L}_\delta := \{\delta^{-1} L_x^* \delta | x \in \mathcal{A}\}$$

and End $(\mathcal{A})$. These are subalgebras of End $(\mathcal{A})$, and since $x \mapsto R_x$ and $x \mapsto \delta^{-1} L_x^* \delta$ $(x \in \mathcal{A})$ are antiautomorphisms from $\mathcal{A}$ onto $\mathcal{R}$ and $\mathcal{L}_\delta$ respectively, $\mathcal{R}$ and $\mathcal{L}_\delta$ are simple subalgebras of End $(\mathcal{A})$. $\varphi : R_x \mapsto \delta^{-1} L_x^* \delta$ is an algebra isomorphism from $\mathcal{R}$ onto $\mathcal{L}_\delta$ which leaves the center $k \cdot id$ of End $(\mathcal{A})$ elementwise fixed. Hence the classical Skolem–Noether theorem [8] applies: there is a $T \in Gl(\mathcal{A})$ such that $\varphi(R_x) = T R_x T^{-1}$ for all $x \in \mathcal{A}$, i.e.,

$$\delta^{-1} L_x^* \delta = T R_x T^{-1} \quad \text{for all } x \in \mathcal{A}.$$

Hence

$$\forall_{x \in \mathcal{A}} \quad L_x^* = (\delta T) R_x (\delta T)^{-1},$$

$S := \delta T$ is a vectorspace isomorphism from $\mathcal{A}$ onto $\mathcal{A}^*$ and (i) holds. To prove (ii), consider $\chi \in \mathcal{A}^*$ and $x, y \in \mathcal{A}$. Then

$$t_\chi(x)(y) = \chi(xy) = (L_x^*\chi)(y) = (SR_xS^{-1}\chi)(y) = (S((S^{-1}\chi)\cdot x))(y),$$

hence $t_\chi(x) = SL_{S^{-1}\chi}(x)$ for all x. □

Remarks 1.2. (1) S is not unique: if S fulfills $L_x^* = SR_xS^{-1}$ for all $x \in \mathcal{A}$ and $a \in \mathcal{A}$ is a *unit*, then also $\tilde{S} := SL_a$ gives the desired link between L_x^* and R_x. Conversely, if S_1, S_2 are isomorphisms $\mathcal{A} \to \mathcal{A}^*$ such that Proposition 1.1 (i) holds, then $S_2 = S_1L_{S_1^{-1}S_21}$, where 1 denotes the unit element of $\mathcal{A}$. We will use this possibility of changing S later on.

(2) Using Wedderburn's structure theorem [8], it is not difficult to show that Proposition 1.1 also holds for semisimple k-algebras. Proposition 1.1 however, cannot be generalized to nonsemisimple algebras, as the example $\mathcal{A} := k[X^2, X^3]/(X^6, X^7)$ shows.

(3) If $\mathcal{A}$ is a division algebra, i.e., if L_x is a vector space isomorphism whenever $x \neq 0$, then S can be defined by $S(x) := \chi \circ L_x$, where χ is a fixed but arbitrary element of $\mathcal{A}^*\backslash\{0\}$.

1.2. Characterization of division algebras with minimal rank. Let $\mathcal{A}$ be a finite dimensional k-algebra with unit element 1 and let s be the number of maximal (two-sided) ideals of $\mathcal{A}$. As usual the algebra $\mathcal{A}$ itself is not regarded as a maximal ideal.

In [1] A. Alder and V. Strassen showed that $2\dim\mathcal{A} - s$ is a lower bound for the (commutative) complexity of the multiplication in $\mathcal{A}$, hence also a lower bound for the rank of $\mathcal{A}$, i.e., the rank of the tensor of $\mathcal{A}$.

If $\mathcal{A}$ is a division algebra (also called a "skew field" sometimes) this lower bound is simply $2\dim\mathcal{A} - 1$, for (0) is the only maximal ideal of $\mathcal{A}$. This is a well-known lower bound and its proof is in fact an almost trivial exercise.

The only known examples of division algebras for which $2\dim\mathcal{A} - 1$ is the actual rank are the simple algebraic field extensions of k, provided k has enough elements [13]. Therefore we are faced with the question whether this class of examples is exhaustive or not. We will show here that a division algebra $\mathcal{A}$ with minimal rank $2\dim\mathcal{A} - 1$ is generated (as an algebra) by a single element, i.e., is a simple field extension of k.

In what follows, $\mathcal{A}$ is a unitary n-dimensional division algebra with minimal rank, t the tensor of its multiplication.

LEMMA 1.3. *If* $t = \sum_{\rho=1}^{2n-1} u_\rho \otimes v_\rho \otimes w_\rho \in \mathcal{A}^* \otimes \mathcal{A}^* \otimes \mathcal{A}$ *is an optimal decomposition of* t, *then any subset of* $\{u_1, \cdots, u_{2n-1}\}$, $\{v_1, \cdots, v_{2n-1}\}$ *or* $\{w_1, \cdots, w_{2n-1}\}$ *with* n *elements is linearly independent.*

Proof. Let $\{w_{\rho_1}, \cdots, w_{\rho_n}\} \subset \{w_1, \cdots, w_{2n-1}\}$ be an n-element subset. Without loss of generality we may assume that $\rho_i = i$ for $i = 1, \cdots, n$. Consider a non-zero $x \in \mathcal{A}$ which is orthogonal to $\{u_{n+1}, \cdots, u_{2n-1}\}$, i.e., it satisfies the equations $u_\rho(x) = 0$ for $\rho = n+1, \cdots, 2n-1$. Then the layer $t_x = L_x$ of t with respect to x is simply

$$L_x = \sum_{\rho=1}^{n} u_\rho(x) v_\rho \otimes w_\rho.$$

Because $\mathcal{A}$ is a division algebra, x is a unit and therefore $L_x \in Gl(\mathcal{A})$. This implies that $\{v_1, \cdots, v_n\}$ and $\{w_1, \cdots, w_n\}$ are linearly independent subsets of $\mathcal{A}^*$ and $\mathcal{A}$ respectively.

If $\{u_{\rho_1}, \cdots, u_{\rho_n}\}$ is an n-element subset of $\{u_1, \cdots, u_{2n-1}\}$, consider a nonzero element y orthogonal to $\{v_1, \cdots, v_{2n-1}\}\backslash\{v_{\rho_1}, \cdots, v_{\rho_n}\}$ and argue with R_y in the same way as above. □

THEOREM 1.4. *A division algebra $\mathscr{A}$ over k has rank* $2\dim\mathscr{A}-1$ *if and only if $\mathscr{A}$ is a simple field extension of k and* $\#k \geqq 2\dim\mathscr{A}-2$.

Proof. The idea of the proof is to construct a primitive element for $\mathscr{A}$. According to Proposition 1.1 we choose a vectorspace isomorphism $S:\mathscr{A}\to\mathscr{A}^*$ such that $t_\chi = SL_{S^{-1}\chi}$ for all $\chi\in\mathscr{A}^*$. Consider an optimal decomposition of t into tensors of rank one:

$$t=\sum_{\rho=1}^{2n-1} u_\rho\otimes v_\rho\otimes w_\rho. \tag{1.4.1}$$

In order to simplify our discussion we shall use the fact that we can obtain further optimal decompositions of t by means of the transformation

$$u_\rho\otimes v_\rho\otimes w_\rho\mapsto A^*u_\rho\otimes B^*v_\rho\otimes Cw_\rho \qquad (\rho=1,\cdots,2n-1)$$

where

$$A=L_aR_b,\quad B=L_{b^{-1}}R_c,\quad C=L_{a^{-1}}R_{c^{-1}}$$

and a, b, c are units of $\mathscr{A}$ [4, Thm. 3.1]. Hence we may assume without loss of generality that $v_1=S1$ in decomposition (1) of t. From Lemma (1.3) we know that $\{w_n,\cdots,w_{2n-1}\}$ is linearly independent, i.e., is a basis of $\mathscr{A}$. Let $\{\chi_1,\cdots,\chi_n\}\subset\mathscr{A}^*$ be the dual basis:

$$\chi_i(w_{n-1+j})=\delta_{ij}\quad \text{for } i,j=1,\cdots,n.$$

Then we obtain for all $i=1,\cdots,n$:

$$SL_{S^{-1}\chi_i}=t_{\chi_i}=\sum_{\rho=1}^{n-1} u_\rho\otimes v_\rho\chi_i(w_\rho)+u_{n-1+i}\otimes v_{n-1+i}. \tag{1.4.2}$$

Now choose $y_1\perp\{u_1,\cdots,u_{n-1}\}$, $y_1\neq 0$, and $y_2\perp\{u_2,\cdots,u_{n-1}\}$ such that $u_1(y_2)=1$. In particular this implies that $\{y_1,y_2\}$ is linearly independent, i.e., $y_1^{-1}y_2\notin k$. We conclude from (2) that for all $i=1,\cdots,n$

$$S((S^{-1}\chi_i)\cdot y_1)=u_{n-1+i}(y_1)v_{n-1+i} \tag{1.4.3}$$

and

$$S((S^{-1}\chi_i)\cdot y_2)=\chi_i(w_1)S1+u_{n-1+i}(y_2)v_{n-1+i} \tag{1.4.4}$$

holds. Now (3) implies

$$S^{-1}\chi_i=u_{n-1+i}(y_1)(S^{-1}v_{n-1+i})y_1^{-1}$$

for $i=1,\cdots,n$. Substituting this in (4) we can solve for $S^{-1}v_{n-1+i}$. It follows that

$$S^{-1}v_{n-1+i}=\chi_i(w_1)[u_{n-1+i}(y_1)(y_1^{-1}y_2)-u_{n-1+i}(y_2)]^{-1} \tag{1.4.5}$$

for $i=1,\cdots,n$. Observe that, for all i, $u_{n-1+i}(y_1)(y_1^{-1}y_2)-u_{n-1+i}(y_2)$, and hence $S^{-1}v_{n-1+i}$, is contained in the subfield $k(y_1^{-1}y_2)$ of $\mathscr{A}$ generated by $y_1^{-1}y_2$. On the other hand,

$$\mathscr{A}=\operatorname{lin}_k\{S^{-1}v_{n-1+i}\,|\,1\leqq i\leqq n\}$$

by Lemma 1.3, hence

$$\mathscr{A}=k(y_1^{-1}y_2),$$

and $y_1^{-1}y_2$ is a primitive element of the field extension $\mathscr{A}/k$.

The assertion $\#k\geqq 2n-2$ will follow from the classification of optimal algorithms for t.

Conversely, it is well known that multiplication in a simple field extension of degree n over k has rank $2n-1$ provided that $\#k \geqq 2n-2$ (cf. [3], [12]). □

Remark 1.5. Our proof uses Proposition 1.1, hence implicitly the Skolem–Noether theorem. It is possible, however, to give a quite analogous proof without using Proposition 1.1: without loss of generality assume $w_1 = 1$; let $\{x_1, \cdots, x_n\} \subset \mathcal{A}$ be the basis dual to $\{u_n, \cdots, u_{2n-1}\}$. Then

$$L_{x_i} = \sum_{\rho=1}^{n-1} u_\rho(x_i) v_\rho \otimes w_\rho + v_{n-1+i} \otimes w_{n-1+i}.$$

Consider $z_1 \perp \{v_1, \cdots, v_{n-1}\}$, $z_1 \neq 0$, and $z_2 \perp \{v_2, \cdots, v_{n-1}\}$ such that $v_1(z_2) = 1$. Now it should be clear how to proceed, and we leave this as an exercise to the reader. The reason for the proof given for Theorem 1.4 is that it gives us information on the structure of $v_n, \cdots, v_{2n-1}$ (cf. (1.4.5)). We will need this information later on for the classification of optimal algorithms for t.

COROLLARY 1.6. (1) *If $\mathcal{A}$ is a noncommutative division algebra over k, then its rank is at least* $2 \dim \mathcal{A}$.

(2) *Let k be an infinite field, K a finite algebraic extension of k. The extension K/k is simple if and only if K has rank* $2[K:k]-1$.

Note that the assumption $\#k = \infty$ is quite natural for the equivalence stated in (2), for if k is a finite field, each finite algebraic extension K over k is a finite field. The multiplicative group of a finite field is cyclic, hence the extension K/k is simple.

2. Classification of optimal algorithms for the multiplication in simple field extensions.

2.1. Parametrization of optimal algorithms for simple field extensions. In this subsection we will derive a parametrization of the variety of optimal algorithms for the multiplication in a simple extension K/k of complexity $2n-1$ where $n := [K:k]$ is the degree of the field extension.

The result is a minor modification of that in [13]. The proof, however, is much less computational than that of S. Winograd, thus showing the power of coordinate-free methods in algebraic complexity theory.

In what follows we will denote by t the tensor of multiplication in K.

DEFINITION 2.1. A product $u \otimes v$ is called *symmetric* if $[u] = [v]$.

The next lemma shows that optimal algorithms for K must have strong symmetries:

LEMMA 2.2. *Let $t = \sum_{\rho=1}^{2n-1} u_\rho \otimes v_\rho \otimes w_\rho$ be an optimal decomposition of t. If one of the products $u_\rho \otimes v_\rho$ is symmetric then all of them are.*

Proof. Let $u_1 \otimes v_1$ be symmetric. Without loss of generality we may assume that $u_1 = v_1$ by scaling (cf. [4]). Consider $(n-1)$-element sets $\{u_{\rho_1}, \cdots, u_{\rho_{n-1}}\}$, $\{v_{\rho_1}, \cdots, v_{\rho_{n-1}}\}$ that do not contain u_1. If we can show that the products $u_{\rho_i} \otimes v_{\rho_i}$ $(i = 1, \cdots, n-1)$ are symmetric, we are done. For notational convenience we will assume that the sets above are $\{u_2, \cdots, u_n\}$ and $\{v_2, \cdots, v_n\}$ respectively. According to Lemma 1.3 there are (unique) elements $x, y \in K$,

$$x \perp \{u_{n+1}, \cdots, u_{2n-1}\}, \qquad y \perp \{v_{n+1}, \cdots, v_{2n-1}\},$$

such that $u_1(x) = v_1(y) = 1$. As K is commutative, we have

$$m_z := L_z = R_z \quad \text{for all } z \in K.$$

Therefore

$$m_x = \sum_{\rho=1}^{n} u_\rho(x) v_\rho \otimes w_\rho,$$

and on the other hand

$$m_y = \sum_{\rho=1}^{n} v_\rho(y)u_\rho \otimes w_\rho.$$

Now

$$m_{x-y} = m_x - m_y = \sum_{\rho=1}^{n} [u_\rho(x)v_\rho - v_\rho(y)u_\rho] \otimes w_\rho$$

$$= \sum_{\rho=2}^{n} [u_\rho(x)v_\rho - v_\rho(y)u_\rho] \otimes w_\rho,$$

and it follows that the multiplication operator m_{x-y} is singular. This is possible only if $m_{x-y} = 0$, i.e. $x = y$, and from the linear independence of $\{w_2, \cdots, w_n\}$ we then conclude that

$$u_\rho(x)v_\rho = v_\rho(x)u_\rho \quad \text{for } \rho = 2, \cdots, n.$$

Thus the products $u_2 \otimes v_2, \cdots, u_n \otimes v_n$ are symmetric. □

Now let $S: K \to K^*$ be a k-vectorspace isomorphism with the properties (i), (ii) of Proposition 1.1. Let $t = \sum_{\rho=1}^{2n-1} u_\rho \otimes v_\rho \otimes w_\rho$ be an optimal decomposition of t. By means of the isotropy group we may assume that

$$u_1 = v_1 = S1. \tag{2.1.1}$$

Then Lemma 2.2 shows that we may scale the other products in such a way that

$$u_\rho = v_\rho$$

holds for all $\rho = 1, \cdots, 2n-1$. From the proof of Theorem 1.4 we see that there are a primitive element $\omega \in K$ and elements $\alpha_{n-1+\nu}, \beta_{n-1+\nu} \in k$ such that

$$\forall_{\nu=1,\cdots,n} \quad v_{n-1+\nu} = S(1/(\alpha_{n-1+\nu}\omega - \beta_{n-1+\nu})).$$

As $\{v_1, v_{n-1+\nu}\}$ $(\nu = 1, \cdots, n)$ must be linearly independent, it follows that $\alpha_{n-1+\nu} \neq 0$ for all ν, hence we may scale the $v_{n-1+\nu}$ such that

$$\forall_{\nu=1,\cdots,n} \quad v_{n-1+\nu} = S(1/(\omega - \beta_{n-1+\nu})) \tag{2.1.2}$$

with suitable $\beta_{n-1+\nu} \in k$. It remains to determine $v_2, \cdots, v_{n-1}$. For this we shall make a special choice for the isomorphism $S: K \to K^*$. Observe first that our primitive element ω does not depend on S.

Now choose $\chi \in K^*$ such that

$$\chi \perp \{1, \omega, \cdots, \omega^{n-2}\}.$$

There is exactly one $a \in K$ with $\chi = S(a)$. Now $S' := Sm_a$ has the property

$$S'1 = Sa \perp \{1, \omega, \cdots, \omega^{n-2}\}.$$

Finally observe that the switch from S to S' can be achieved by applying $m_a^* \otimes m_a^* \otimes m_{a^{-2}} \in \Gamma^0$ to our algorithm:

$$m_a^* Sx = Sm_a S^{-1} Sx = S'x.$$

Hence we may assume that S in (2) has the property

$$S1 \perp \{1, \omega, \cdots, \omega^{n-2}\}.$$

Now let $\{\chi_1, \cdots, \chi_n\} \subset K^*$ be the basis dual to $\{w_1, \cdots, w_n\} \subset K$. Then for all $\nu = 1, \cdots, n$ we have

$$Sm_{S^{-1}\chi_\nu} = \sum_{\rho=n+1}^{2n-1} v_\rho \otimes v_\rho \chi_\nu(w_\rho) + v_\nu \otimes v_\nu.$$

According to the special choice of S,

$$y_1 := \prod_{\nu=2}^{n} (\omega - \beta_{n-1+\nu})$$

is orthogonal to $\{v_{n+1}, \cdots, v_{2n-1}\}$ and

$$y_2 := \prod_{\nu=3}^{n} (\omega - \beta_{n-1+\nu})$$

is orthogonal to $\{v_{n+2}, \cdots, v_{2n-1}\}$.

Now the very same reasoning as in the proof of Theorem 1.4 shows that

$$\bigwedge_{\nu=1}^{n} \quad S^{-1}v_\nu = v_{n+1}(y_2)\chi_\nu(w_{n+1}) \cdot \frac{S^{-1}v_{n+1}}{v_\nu(y_1)y_1^{-1}y_2 - v_\nu(y_2)}.$$

As $S^{-1}v_{n+1} = 1/(\omega - \beta_{n+1})$ and $y_1^{-1}y_2 = 1/(\omega - \beta_{n+1})$, we see that

$$v_\nu = S(1/(\tilde{\alpha}_\nu \omega - \tilde{\beta}_\nu)) \qquad (\nu = 1, \cdots, n),$$

and by suitable scaling we may assume that

$$v_\nu = S(1/(\omega - \beta_\nu)) \qquad (\nu = 2, \cdots, n).$$

The products of our algorithms are now reduced to $v_\rho \otimes v_\rho$ $(\rho = 1, \cdots, 2n-1)$ where

$$\begin{aligned} &v_1 = S1, \\ &v_\rho = S(1/(\omega - \beta_\rho)) \qquad (\rho = 2, \cdots, 2n-1). \end{aligned} \tag{2.1.3}$$

Since any n-element subset of $\{v_1, \cdots, v_{2n-1}\}$ must be linearly independent, it follows that $\beta_\rho \neq \beta_\sigma$ for $\rho \neq \sigma$. Hence $\#k \geqq 2n-2$.

Conversely, if any primitive element $\omega \in K$ and any $2n-2$ elements $\beta_2, \cdots, \beta_{2n-1}$ k are given, the products

$$v_1 \otimes v_1 := S1 \otimes S1, \quad v_\rho \otimes v_\rho := S(1/(\omega - \beta_\rho)) \otimes S(1/(\omega - \beta_\rho)) \qquad (\rho = 2, \cdots, 2n-1)$$

define an optimal algorithm for t. Indeed, observe that as before we may assume that

$$S1 \perp \{1, \omega, \cdots, \omega^{n-2}\}.$$

But then for all $\gamma \in \{1, \cdots, n-1\}$ and all $\rho \in \{2, \cdots, 2n-1\}$ we have

$$S\frac{1}{\omega - \beta_\rho}(\omega^\gamma) = S1\left(\frac{\omega^\gamma}{\omega - \beta_\rho}\right) = S1\left(\omega^{\gamma-1} + \frac{\beta_\rho \omega^{\gamma-1}}{\omega - \beta_\rho}\right) = \beta_\rho S1\left(\frac{\omega^{\gamma-1}}{\omega - \beta_\rho}\right),$$

hence

$$S\frac{1}{\omega - \beta_\rho}(\omega^\gamma) = S\frac{1}{\omega - \beta_\rho}(1)\beta_\rho^\gamma$$

for all $\gamma = 0, \cdots, n-1$.

Therefore we have for all $x = \sum_{\gamma=0}^{n-1} x_\gamma \omega^\gamma \in K$

$$S\frac{1}{\omega-\beta_\rho}(x) = S\frac{1}{\omega-\beta_\rho}(1)\left(\sum_{\gamma=0}^{n-1} x_\gamma \beta_\rho^\gamma\right),$$

and we conclude that the products $v_\rho \otimes v_\rho$ determine one of the well-known interpolation algorithms (with one interpolation point at infinity due to the choice of $S1 \otimes S1$) for the multiplication in K.

Thus we have the following result (cf. [13]):

THEOREM 2.3. *Let k be a field, K/k a simple finite extension of k. Let $S: K \to K^*$ be a k-vectorspace isomorphism such that $m_x^* = Sm_xS^{-1}$ for all $x \in K$. Let $:= [K:k]$ be the degree of the extension K/k. If $\#k \geqq 2n-2$ then any optimal algorithm for the multiplication in K is determined up to equivalence modulo the isotropy group $\Gamma^0_{K/k}$ by products $v_\rho \otimes v_\rho$ $(\rho = 1, \cdots, 2n-1)$ of the form*

$$v_1 = S1, \qquad v_\rho = S(1/(\omega-\beta_\rho)) \qquad (\rho \geqq 2)$$

where $\omega \in K$ is a primitive element and $\beta_2, \cdots, \beta_{2n-1} \in k$ are pairwise different. Conversely, any set of products of the above form, i.e., any choice of a primitive element $\omega \in K$ and of pairwise different $\beta_2, \cdots, \beta_{2n-1} \in k$, determines an optimal algorithm for the multiplication in K.

This description of the algorithm variety, however, is not yet satisfactory, for some of the algorithms mentioned above are equivalent modulo the isotropy group of K/k. There remains the task of determining the *orbit space* of the algorithm variety modulo the isotropy group.

2.2. The orbit space of the algorithm variety of a finite simple field extension modulo its isotropy group. We recall first how the (small) isotropy group $\Gamma^0_{\mathcal{A}}$ of a finite dimensional algebra $\mathcal{A}$ over a field k looks.

THEOREM 2.4 [4, Thm. 3.1]. *Let $\mathcal{A}$ be a finite dimensional algebra over the field k and $A^* \otimes B^* \otimes C$ an automorphism of the vector space $\mathcal{A}^* \otimes \mathcal{A}^* \otimes \mathcal{A}$. Then $A^* \otimes B^* \otimes C \in \Gamma^0_{\mathcal{A}}$ if and only if there are units $a, b \in \mathcal{A}$ and an automorphism φ of the algebra $\mathcal{A}$ such that*

$$A = L_a \circ \varphi, \quad B = R_b \circ \varphi, \quad C = \varphi^{-1} \circ L_{a^{-1}} \circ R_{b^{-1}}.$$

In our case of a finite simple field extension K/k, we have to consider

$$A = m_a \circ \varphi, \quad B = m_b \circ \varphi, \quad C = \varphi^{-1} \circ m_{(ab)^{-1}},$$

where $a, b \in K \setminus \{0\}$ and $\varphi \in G$, the group of automorphisms of K over k.

Our first task is to determine the adjoint A^* of $A = m_a \circ \varphi \in Gl(\mathcal{A})$. Now $A^* = \varphi^* \circ m_a^* = \varphi^* \circ S \circ m_a \circ S^{-1}$ with a suitable isomorphism $S: K \to K^*$, it suffices to compute $\varphi^*: K^* \to K^*$ from $\varphi \in G$.

LEMMA 2.5. *Let K/k be a finite field extension, G the automorphism group of K over k. Then there is a k-vectorspace isomorphism $S: K \to K^*$ such that*

(i) $\qquad m_x^* = Sm_xS^{-1}$ *for all $x \in K$ and*

(ii) $\qquad \varphi^* = S\varphi^{-1}S^{-1}$ *for all $\varphi \in G$.*

Proof. (a) We will discuss first the special case where K/k is a Galois extension, i.e. normal and separable. In this case the trace of K over k fulfills our requirements: Let $G := \mathrm{Gal}(K/k)$ be the Galois group of the extension. The trace of an element

$x \in K$ is defined as

$$\mathrm{tr}\,(x) := \sum_{\sigma \in G} \sigma(x).$$

Recall that $\mathrm{tr}\,(x)$ is always an element of k. Now define $S: K \to K^*$ by

$$Sx(y) := \mathrm{tr}\,(xy).$$

S is a k-vectorspace isomorphism and for all $x, y, z \in K$ we have

$$\begin{aligned} m_x^* Sy(z) &= Sy(xz) = \mathrm{tr}\,(y(xz)) \\ &= \mathrm{tr}\,((xy)z) = Sxy(z) = Sm_x y(z), \end{aligned}$$

hence

$$m_x^* = Sm_x S^{-1}.$$

Let $\varphi \in G$. Then for all $x, y \in K$ we have

$$\begin{aligned} \varphi^* Sx(y) &= Sx(\varphi(y)) = \mathrm{tr}\,(x\varphi(y)) \\ &= \sum_{\sigma \in G} \sigma(x\varphi(y)) = \sum_{\sigma \in G} \sigma\varphi(\varphi^{-1}(x)y) \\ &= \sum_{\tau \in G} \tau(\varphi^{-1}(x)y) = S\varphi^{-1}(x)(y), \end{aligned}$$

hence

$$\varphi^* = S\varphi^{-1}S^{-1}.$$

(b) In this part of the proof we would like to discuss conditions for the existence of a k-vectorspace isomorphism $S: K \to K^*$ with the properties (i), (ii) where K is any finite extension of k, and G the group of automorphisms of K over k. For $\varphi \in G$ let $I_\varphi := \{\varphi(x) - x \mid x \in K\}$. I_φ is a k-vectorspace and, as φ leaves k elementwise fixed, of dimension $\leqq n-1$. ($n := [K:k]$.) Let $S: K \to K^*$ be a k-vectorspace isomorphism with property (i). Then an easy calculation, using the fact that $Sx(y) = S1(xy)(x, y \in K)$, shows that S also satisfies property (ii) if and only if $S1 \perp \{x\varphi(y) - \varphi^{-1}(x)y \mid x, y \in K\}$. Now $x\varphi(y) - \varphi^{-1}(x)y = (\varphi - id)(\varphi^{-1}(x)y)(x, y \in K, \varphi \in G)$ and $\{\varphi^{-1}(x)y \mid x, y \in K\} = K$ imply that S satisfies (ii) if and only if $S1 \perp I_\varphi$ for all $\varphi \in G$.

According to the fact that if $S: K \to K^*$ has the property (i) then also $S \circ m_a$ ($a \in K \setminus \{0\}$) has this property, we get

(*) There is a k-vectorspace isomorphism $S: K \to K^*$ satisfying (i) and (ii) if and only if there is a $\chi \in K^* \setminus \{0\}$ such that $\chi(I_\varphi) = 0$ for all $\varphi \in G$.

(c) Now let $F := \{x \in K \mid \varphi(x) = x \text{ for all } \varphi \in G\}$ be the fixed field of G. Then a theorem of E. Artin asserts that the extension K/F is Galois with Galois group $\mathrm{Gal}\,(K/F) = G$ (see [10]). Observe that the spaces I_φ are also vectorspaces over the field F. Hence by (a) and (*) there is a nonzero $\chi \in \mathrm{Hom}\,(K, F)$ such that $\chi(I_\varphi) = 0$ for all $\varphi \in G$. Choose any $\xi \in \mathrm{Hom}\,(F, k)$ such that $\xi \circ \chi \neq 0$. Then $\xi \circ \chi \in K^* \setminus \{0\}$ and

$$\xi \circ \chi(I_\varphi) = 0$$

for all $\varphi \in G$. According to (*), the lemma is proved. □

In order to find the distinct equivalence classes of the variety of optimal algorithms for K we have, according to Theorem 2.3, to study transformations

$$v_\rho \otimes v_\rho \mapsto A^* v_\rho \otimes A^* v_\rho$$

of the symmetric products $v_\rho \otimes v_\rho$ described in Theorem 2.3, where $A = m_a \circ \varphi$, $a \in K\backslash\{0\}$, $\varphi \in G$. From Lemma 2.5 we conclude that

$$A^* = \varphi^* m_a^* = S\varphi^{-1} m_a S^{-1},$$

hence

$$A^* S x = S\varphi^{-1} m_a x$$

for all $x \in K$. Therefore our problem is to investigate the action of transformations

$$x \mapsto \varphi(ax) = \varphi(a)\varphi(x)$$

on tuples $(1/(\alpha_1\omega + \beta_1), \cdots, 1/(\alpha_{2n-1}\omega + \beta_{2n-1})) \in K^{2n-1}$, where $\omega \in \Omega$ and the points $(\alpha_\rho : \beta_\rho)$ $(\rho = 1, \cdots, 2n-1)$ of the projective line $\mathbb{P}_1$ over k are pairwise distinct. In other words, given any two such tuples $(1/(\alpha_1\omega + \beta_1), \cdots, 1/(\alpha_{2n-1}\omega + \beta_{2n-1}))$ and $(1/(\alpha'_1\omega' + \beta'_1), \cdots, 1/(\alpha'_{2n-1}\omega' + \beta'_{2n-1}))$, we shall look for conditions that there are $a \in K\backslash\{0\}$ and $\varphi \in G$ such that

$$\bigwedge_{\rho=1}^{2n-1} \frac{1}{\alpha'_\rho\omega' + \beta'_\rho} = \frac{\varphi(a)}{\alpha_\rho\varphi(\omega) + \beta_\rho}.$$

Take $\rho = 1$. Then

$$\varphi(a) = \frac{\alpha_1\varphi(\omega) + \beta_1}{\alpha'_1\omega' + \beta'_1}.$$

Substituting this for $\rho \geqq 2$, we obtain

$$\frac{\alpha'_1\omega' + \beta'_1}{\alpha'_\rho\omega' + \beta'_\rho} = \frac{\alpha_1\varphi(\omega) + \beta_1}{\alpha_\rho\varphi(\omega) + \beta_\rho} = \varphi\left(\frac{\alpha_1\omega + \beta_1}{\alpha_\rho\omega + \beta_\rho}\right),$$

because the automorphism φ leaves k elementwise fixed. Notice that

$$H_\rho : z \mapsto \frac{\alpha_1 z + \beta_1}{\alpha_\rho z + \beta_\rho}$$

and

$$H'_\rho : z \mapsto \frac{\alpha'_1 z + \beta'_1}{\alpha'_\rho z + \beta'_\rho}$$

are *homographies*, for $(\alpha : \beta) \neq (\gamma : \delta)$ is equivalent to the nonsingularity of the $(2, 2)$-matrix $\left(\begin{smallmatrix}\alpha & \beta\\ \gamma & \delta\end{smallmatrix}\right)$. On the other hand, each nonsingular $(2, 2)$-matrix $\left(\begin{smallmatrix}\alpha & \beta\\ \gamma & \delta\end{smallmatrix}\right)$ defines a homography via $z \mapsto (\alpha z + \beta)/(\gamma z + \delta)$ and two matrices $M_1, M_2 \in Gl(k^2)$ determine the same homography if and only if M_2 is a scalar multiple of M_1.

We will write $\left[\begin{smallmatrix}\alpha & \beta\\ \gamma & \delta\end{smallmatrix}\right]$ for the homography defined by $\left(\begin{smallmatrix}\alpha & \beta\\ \gamma & \delta\end{smallmatrix}\right)$. In this way the group $\mathscr{H}$ of homographies of K over k is isomorphic to the group $PGl(2)$ of projectivities of $\mathbb{P}_1$ over k.

Now we can write

$$H'_\rho\omega' = \varphi H_\rho\omega = H_\rho\varphi\omega$$

for $\rho = 2, \cdots, 2n-1$, and

$$\omega' = (H'_2)^{-1} H_2\varphi\omega.$$

Substituting this for $\rho \geqq 3$ we get

$$H'_\rho (H'_2)^{-1} H_2\varphi\omega = H_\rho\varphi\omega. \tag{2.2.1}$$

If $n>2$, the group $\mathcal{H}$ of homographies over k acts fixpoint-free on Ω; hence

$$H'_\rho = H_\rho H_2^{-1} H'_2$$

for all $\rho = 2, \cdots, 2n-1$. A simple calculation shows that

$$H := H_2^{-1}H'_2 = \begin{bmatrix} \beta_2\alpha'_1 - \beta_1\alpha'_2 & \beta_2\beta'_1 - \beta_1\beta'_2 \\ -\alpha_2\alpha'_1 + \alpha_1\alpha'_2 & -\alpha_2\beta'_1 + \alpha_1\beta'_2 \end{bmatrix}$$

$$=: \begin{bmatrix} \varepsilon_1 & \varepsilon_2 \\ \varepsilon_3 & \varepsilon_4 \end{bmatrix}$$

and

$$(\alpha'_\rho : \beta'_\rho) = (\alpha_\rho : \beta_\rho)H,$$

i.e., $\alpha'_\rho = \varepsilon_1\alpha_\rho + \varepsilon_3\beta_\rho$, $\beta'_\rho = \varepsilon_2\alpha_\rho + \varepsilon_4\beta_\rho$ for $\rho = 1, 2, \cdots, 2n-1$.

Therefore, using the shorthand notation $(\omega, (\alpha_1:\beta_1), \cdots, (\alpha_{2n-1}:\beta_{2n-1})) \in \Omega \times \mathbb{P}_1^{2n-1}$ for the algorithm determined by $(1/(\alpha_1\omega + \beta_1), \cdots, 1/(\alpha_{2n-1}\omega + \beta_{2n-1}))$, we have proved:

PROPOSITION 2.6. *If $n>2$, then an algorithm $(\omega', (\alpha'_1:\beta'_1), \cdots, (\alpha'_{2n-1}:\beta'_{2n-1}))$ is contained in the orbit of $(\omega, (\alpha_1:\beta_1), \cdots, (\alpha_{2n-1}:\beta_{2n-1}))$ under the isotropy group of K if and only if there are $H \in \mathcal{H}$ and $\varphi \in G$ such that*

$$\omega' = H^{-1}\varphi\omega \quad \text{and} \quad (\alpha'_\rho : \beta'_\rho) = (\alpha_\rho : \beta_\rho)H$$

for all $\rho = 1, \cdots, 2n-1$.

We will use now the fact that $PGl(2)$ acts sharply three-fold transitive on $\mathbb{P}_1$. Therefore it is possible to transform the factors of the first three products of all algorithms to $(0:1)$, $(1:1)$ and $(1:0)$ respectively, and we conclude that for any two equivalent algorithms of this form the corresponding H must be the identity.

Hence we have proved that if $n>2$, the orbit space of the algorithm variety modulo the isotropy group is given by

$$\Omega/G \times (((0:1), (1:1), (1:0)) \times \mathbb{P}_1^{2n-4}) \backslash \Delta_{2n-1},$$

where

$$\Delta_m := \left\{ (p_1, \cdots, p_m) \in \mathbb{P}_1^m \,\middle|\, \exists_{i \neq j}\ p_i = p_j \right\}$$

is the weak diagonal of $\mathbb{P}_1^m$.

It remains to discuss the case of a quadratic extension K/k. Here all $\omega \in K \backslash k$ are primitive elements and the situation differs from the case $[K:k]>2$ because each homography has fixed points in $\Omega = K \backslash k$: Let $\omega \in \Omega$ and let $X^2 + pX + q$ be the irreducible polynomial of ω. If $H = \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$, then $H\omega = (a\omega + b)/(c\omega + d) = \omega$ if and only if $c\omega^2 + (d-a)\omega - b = 0$. If $c = 0$, then, since $\omega \notin k$, $b = 0$ and $a = d$, i.e., H is the identity. Otherwise

$$\omega^2 + \frac{d-a}{c}\omega - \frac{b}{c} = 0,$$

i.e. $b = -qc$ and $d = a + pc$. Therefore

$$\text{Fix}(\omega) := \left\{ \begin{bmatrix} a & -qc \\ c & a+pc \end{bmatrix} \Big/ a, c \in k, a \neq 0 \text{ or } c \neq 0 \right\}$$

is the subgroup of $\mathcal{H}$ that leaves ω fixed. Observe that Fix (ω) = Fix $(\varphi(\omega))$ for $\varphi \in G$, because ω and $\varphi(\omega)$ have the same irreducible polynomial. Going back to (2.2.1), we see that we can only conclude that there is a $F_\omega \in$ Fix (ω) such that

$$H_3' H^{-1} = H_3 F_\omega,$$

i.e.

$$H_3' = H_3 F_\omega H. \tag{2.2.2}$$

But an easy calculation shows that for every choice of $(\alpha_3' : \beta_3')$ there exists $F_\omega \in$ Fix (ω) such that $(\alpha_3' : \beta_3') = (\alpha_3 : \beta_3) F_\omega H$ holds. Hence the orbit space is simply Ω/G.

It is worth remarking that the orbit space in the case $k = \mathbb{R}$, $K = \mathbb{C}$ looks like

$$\mathfrak{H}_+ = \{z \in \mathbb{C} \mid \operatorname{Im} z > 0\},$$

the upper half of the complex plane.

Summarizing our results, we get the following.

THEOREM 2.7. *The orbit space of the algorithm variety of simple field extensions K/k of degree n modulo the isotropy group $\Gamma_{K/k}$ is*

$$\Omega/G \times (((0:1), (1:1), (1:0)) \times \mathbb{P}_1^{2n-4}) \setminus \Delta_{2n-1},$$

where $\Omega \subseteq K$ is the set of primitive elements of K, and Δ_{2n-1} is the weak diagonal of the $(2n-1)$-fold product of the projective line $\mathbb{P}_1$ over k.

If we consider the action of the extended isotropy group (cf. [4, p. 16]), the situation becomes somewhat more complicated. Here we have to take into account the effect of permutations of the products, and the equivalence class of an algorithm modulo the permutation group is just the *set* of products occurring in the algorithm.

In our case the extended isotropy group is $\Gamma^0_{K/k} \gamma_{2n-1}$. Let $\mathfrak{A}_{K/k}$ be the algorithm variety of the field extension K/k. The actions of $\Gamma^0_{K/K}$ and γ_{2n-1} commute on $\mathfrak{A}_{K/k}$, hence γ_{2n-1} operates on the quotient $\mathfrak{A}_{K/k}/\Gamma^0_{K/k}$. Let a Γ^0-orbit be represented by $(\tilde{\omega}, P_1, \cdots, P_{2n-1})$, where $\tilde{\omega} \in \Omega/G$ and the tuple $(P_1, \cdots, P_{2n-1}) \in \mathbb{P}_1^{2n-1} \setminus \Delta_{2n-1}$ is such that $P_1 = (1:0)$, $P_2 = (1:1)$, $P_3 = (0:1)$. We denote the set of such tuples by $\mathcal{P}^{2n-1}_{0,1,\infty}$. If $\mathcal{P} = (P_1, \cdots, P_{2n-1}) \in \mathcal{P}^{2n-1}_{0,1,\infty}$ and $\pi \in \gamma_{2n-1}$, write $\pi^{-1}(\mathcal{P})$ for $(P_{\pi(1)}, \cdots, P_{\pi(2n-1)})$. Observe that for every $P \in \mathcal{P}^{2n-1}_{0,1,\infty}$ and $\pi \in \gamma_{2n-1}$ there is a unique $H \in \mathcal{H}$ such that $H\pi(\mathcal{P}) \in \mathcal{P}^{2n-1}_{0,1,\infty}$. In this way γ_{2n-1} operates on $\mathcal{P}^{2n-1}_{0,1,\infty}$, and we write T_π for $H\pi$.

Now it can happen that the action of a nontrivial $\pi \in \gamma_{2n-1}$ can also be expressed by applying a homography (this depends on the cross ratios of the coordinate points in $\mathcal{P}$):

$$\pi^{-1}(\mathcal{P}) = H(\mathcal{P}) = (H(P_1), \cdots, H(P_{2n-1})),$$

i.e. that $T_\pi(\mathcal{P}) = \mathcal{P}$.

$$\mathcal{H}_\mathcal{P} := \{H \in \mathcal{H} \mid \underset{\pi \in \gamma_{2n-1}}{\exists} H\pi(\mathcal{P}) = \mathcal{P}\}$$

is a subgroup of $\mathcal{H}$, isomorphic to

$$\gamma_{2n-1}(\mathcal{P}) := \{\pi \in \gamma_{2n-1} \mid T_\pi(\mathcal{P}) = \mathcal{P}\}.$$

If $\mathcal{P} \equiv \mathcal{P}'$ mod T, i.e. if $\mathcal{P}' = T_\pi(\mathcal{P})$ for some $\pi \in \gamma_{2n-1}$, then the groups $\mathcal{H}_\mathcal{P}$ and $\mathcal{H}_{\mathcal{P}'}$ are conjugate subgroups of $\mathcal{H}$ and therefore $\Omega/G\mathcal{H}_\mathcal{P} = \Omega/G\mathcal{H}_{\mathcal{P}'}$. Hence we have the following

THEOREM 2.8. *Let*

$$\mathcal{P}^{2n-1}_{0,1,\infty} := \{(P_1, \cdots, P_{2n-1}) \in \mathbb{P}_1^{2n-1} \setminus \Delta_{2n-1} \mid P_1 = (1:0), P_2 = (1:1), P_3 = (0:1)\}.$$

Then the orbit space of the algorithm variety $\mathfrak{A}_{K/k}$ *of the simple field extension* K/k *of degree* n *modulo the extended isotropy group* $\Gamma^0_{K/k}\gamma_{2n-1}$ *can be represented by the disjoint union*

$$\coprod_{\tilde{\mathcal{P}}\in\mathcal{P}^{2n-1}_{0,1,\infty}/\gamma_{2n-1}} \Omega/G\mathcal{H}_{\mathcal{P}}$$

where $\mathcal{P}\in\tilde{\mathcal{P}}$ *and* $\mathcal{H}_{\mathcal{P}}=\{H\in\mathcal{H}\mid\exists_{\pi\in\gamma_{2n-1}}\, H\pi(\mathcal{P})=\mathcal{P}\}$.

If $n>2$ and if k is infinite, then "in general" the groups $\mathcal{H}_{\mathcal{P}}$ and $\gamma_{2n-1}(\mathcal{P})$ are trivial.

However, in case of *quadratic* extensions we obtain a quite nice structure: here the only $\mathcal{P}$ to be considered is

$$\mathcal{P}=((1:0),(1:1),(0:1)),$$

and we have $\gamma_3(\mathcal{P})=\gamma_3$. The group $\mathcal{H}_{\mathcal{P}}=:\mathfrak{G}$, isomorphic to γ_3, is

$$\mathfrak{G}=\left\{\begin{bmatrix}1&0\\0&1\end{bmatrix},\begin{bmatrix}0&1\\1&0\end{bmatrix},\begin{bmatrix}-1&1\\0&1\end{bmatrix},\begin{bmatrix}0&1\\-1&1\end{bmatrix},\begin{bmatrix}1&-1\\1&0\end{bmatrix},\begin{bmatrix}1&0\\1&-1\end{bmatrix}\right\}.$$

Thus for quadratic extensions the orbit space $\mathfrak{A}_{K/k}/\Gamma^0_{K/k}\gamma_3$ is represented by $\Omega/G\mathfrak{G}$.

We would like to make this quotient explicit in the special case $K=\mathbb{C}$, $k=\mathbb{R}$. $\mathfrak{G}$ is a finite subgroup of the extended modular group, and we shall exhibit a *fundamental region* of $G\mathfrak{G}$, i.e. a maximal set of pairwise inequivalent representatives of $\Omega/G\mathfrak{G}$.

Here is not the place to calculate such a fundamental region explicitly; we merely state the result:

THEOREM 2.9.

$$\mathfrak{A}_{\mathbb{C}/\mathbb{R}}/\Gamma^0_{\mathbb{C}/\mathbb{R}}\gamma_3$$

can be represented by the fundamental region

$$\{z\in\mathbb{C}\,\|z|\leqq 1,|z-1|<1,0<\operatorname{Im} z\}\cup\left\{\frac{1}{2}+\frac{\sqrt{3}}{2}i\right\}.$$

This fundamental region looks like the shaded set in Fig. 2.1.

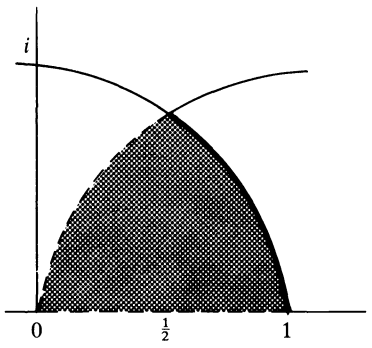


FIG. 2.1

3. Optimal algorithms for the multiplication of polynomials. In this section we would like to sketch how the space of orbits of optimal algorithms for polynomial multiplication under the action of the isotropy group Γ looks, and show its close relation to the orbit space of the algorithm variety of simple field extensions. Details of our considerations are contained in [6].

For $l \in \mathbb{N}$ let $k_l[X]$ be the space of polynomials $f \in k[X]$ of degree $\deg f \leqq l$,

$$\Phi_{m,n}: k_m[X] \times k_n[X] \to k_{m+n}[X],$$

the bilinear mapping defined by polynomial multiplication. Let $t^{m,n}$ be the tensor corresponding to $\Phi_{m,n}$. If we choose the ordered canonical basis $(1, X, \cdots, X^l)$ in $k_l[X]$, and if $(\chi_0, \chi_1, \cdots, \chi_l)$ is the dual basis to $(1, X, \cdots, X^l)$, then the layers $t^{m,n}_{\chi_\nu}$ of $t^{m,n}$ are given by

$$(t^{m,n}_{\chi_\nu})_{\lambda,\mu} = \delta_{\nu-\lambda,\mu},$$

where $0 \leqq \lambda \leqq m$, $0 \leqq \mu \leqq n$, $0 \leqq \nu \leqq m+n$. As $\{t^{m,n}_{\chi_0}, \cdots, t^{m,n}_{\chi_{m+n}}\}$ is linearly independent, we always have $\operatorname{rk}(t^{m,n}) \geqq m+n+1$, and it is not difficult to show

PROPOSITION 3.1 ([9], [12]). $\operatorname{rk}(t^{m,n}) = m+n+1$ *if and only if* $\#k \geqq m+n$.

Moreover, one has the following simple parametrization of the algorithm variety for $\Phi_{m,n}$:

PROPOSITION 3.2 (cf. [9], [12]). (i) *For $\alpha \in k$ let $u_\alpha \in k_m[X]^*$, $v_\alpha \in k_n[X]^*$ be defined by*

$$u_\alpha(f) := f(\alpha), \qquad v_\alpha(g) := g(\alpha),$$

and let $u_\infty(f) := \delta_{m,\deg f}$, $v_\infty(g) := \delta_{n,\deg g}$. Then $P_\infty := u_\infty \otimes v_\infty = t^{m,n}_{\chi_{m+n}}$ and for $\alpha \in k$,

$$P_\alpha := u_\alpha \otimes v_\alpha = \sum_{\nu=0}^{m+n} \alpha^\nu t^{m,n}_{\chi_\nu}.$$

(ii) *If $\#k \geqq m+n$ then every optimal algorithm for the computation of $\Phi_{m,n}$ is given by $(P_{\alpha_0}, \cdots, P_{\alpha_{m+n}})$ where $\alpha_0, \cdots, \alpha_{m+n}$ are pairwise distinct elements of $k \cup \{\infty\}$.*

P_∞ is called the product at infinity. At first glance it may appear that there are two different kinds of optimal algorithms for $\Phi_{m,n}$: those which contain the product at infinity and those which do not ("multiplication of polynomials by interpolation"). In what follows we shall see that this distinction is not adequate.

It is not difficult to determine the group $\Gamma^0_{m,n}$ of all automorphisms $A^* \otimes B^* \otimes C$ of $k_m[X]^* \otimes k_n[X]^* \otimes k_{m+n}[X]$ that leave $t^{m,n}$ fixed: $A^* \otimes B^* \otimes C\, t^{m,n} = t^{m,n}$. This condition is equivalent to

$$C(A(f)B(g)) = fg \quad \text{for all } f \in k_m[X],\ g \in k_n[X].$$

Using elementary arguments on divisibility of polynomials, one obtains from this condition the following result on the structure of the isotropy group $\Gamma^0_{m,n}$ of $\Phi_{m,n}$:

PROPOSITION 3.3. *Let k be a field. Then $A^* \otimes B^* \otimes C \in \Gamma^0_{m,n}$ if and only if there are linear polynomials $L, M \in k_1[X]$ such that $\{L, M\}$ is linearly independent, and for all $f \in k_m[X]$, $g \in k_n[X]$ and $h \in k_{m+n}[X]$,*

$$A(f) = M(X)^m H(f), \qquad B(g) = M(X)^n H(g),$$
$$C^{-1}(h) = M(X)^{m+n} H(h)$$

hold, where $H := L/M$.

If $m = n$, then $\Gamma_{m,n}/\Gamma^0_{m,n} \cong \{\mathrm{id}, \pi_{12}\}$ (π_{12} permutes the first two factors of the tensor product), otherwise $\Gamma_{m,n} = \Gamma^0_{m,n}$.

In other words, $\Gamma^0_{m,n}$ is isomorphic to the group of homographies $X \mapsto (\alpha X + \beta)/(\gamma X + \delta)$, $\alpha\delta \neq \beta\gamma$, i.e. to the automorphism group of the field $k(X)$ over k. A similar result for the special case $m = n$ and a related concept of equivalence of algorithms appear in [2].

From now on let k be large enough, say $\#k = \infty$ for simplicity. We would like to investigate the action of $\Gamma_{m,n}$ on the variety $\mathfrak{A}_{m,n}$ of scaling equivalence classes of

optimal algorithms for $\Phi_{m,n}$. For $N \in \mathbb{N}$, $\mathbb{P}_N$ denotes the N-dimensional projective space over k. Let

$$\mathscr{C}_N := \left\{ (\xi_0 : \cdots : \xi_N) \in \mathbb{P}_N \,\middle|\, \exists_{\alpha,\beta \in k} \; \forall_{i=0,\cdots,N} \; \xi_i = \alpha^i \beta^{N-i} \right\}.$$

It is an obvious consequence of Proposition 3.2 that the variety $\mathfrak{A}_{m,n}$ is isomorphic to $\mathscr{C}_m^{m+n+1} \backslash \Delta_{m+n+1}^m$, where

$$\Delta_{m+n+1}^m := \{(d_1, \cdots, d_{m+n+1}) \in \mathbb{P}_m^{m+n+1} \mid \#\{d_1, \cdots, d_{m+n+1}\} < m+n+1\}$$

is the weak diagonal of P_m^{m+n+1}. Moreover, it is easy to see that $\mathscr{C}_m$ is an irreducible curve in $\mathbb{P}_m$, isomorphic to $\mathbb{P}_1$:

$$\varphi : (\alpha : \beta) \mapsto (\beta^m : \alpha\beta^{m-1} : \cdots : \alpha^m);$$

is a bijective morphism from $\mathbb{P}_1$ onto $\mathscr{C}_m$ whose inverse can be defined by gluing together the morphisms

$$\psi_1 : (\xi_0 : \cdots : \xi_m) \mapsto (\xi_1 : \xi_0)$$

defined on $\mathscr{C}_m \cap \{(\xi_0 : \cdots : \xi_m) \mid \xi_0 \neq 0\}$, and

$$\psi_2 : (\xi_0 : \cdots : \xi_m) \mapsto (\xi_m : \xi_{m-1})$$

defined on $\mathscr{C}_m \cap \{(\xi_0 : \cdots : \xi_m) \mid \xi_m \neq 0\}$. Hence $\mathfrak{A}_{m,n}$ is isomorphic to the Zariski open set

$$\mathbb{P}_1^{m+n+1} \backslash \Delta_{m+n+1},$$

and this isomorphism translates the action of $\Gamma_{m,n}$ on $\mathfrak{A}_{m,n}$ into the action of $PGl(2)$ on $\mathbb{P}_1^{m+n+1} \backslash \Delta_{m+n+1}$. (Observe that the permutational map $\pi_{12} \in \Gamma_{m,m}$ operates identically on $\mathfrak{A}_{m,n}$.) *As* $PGl(2)$ operates sharply threefold transitive on $\mathbb{P}_1$, we have therefore proved the following.

THEOREM 3.4. $\mathfrak{A}_{m,n}/\Gamma_{m,n}$ *is isomorphic to*

$$(((0:1), (1:1), (1:0)) \times \mathbb{P}_1^{m+n-2}) \backslash \Delta_{m+n+1}.$$

Acknowledgment. The author is very much obliged to the referees for useful suggestions concerning the presentation and for pointing out some inadvertances in an earlier version of this paper.

Note added in proof. E. Feig (J. Algorithms, 2 (1981), pp. 261–281) has shown that if A is a division algebra of *minimal complexity*, i.e., if $L(A) = 2 \dim A - 1$, then every commutative optimal algorithm for A is already bilinear. This means that the results of our paper also hold for division algebras of minimal complexity.

REFERENCES

[1] A. ALDER AND V. STRASSEN, *On the algorithmic complexity of associative algebras*, Theoret. Comput. Sci., 15 (1981), pp. 201–211.

[2] R. W. BROCKETT AND D. DOBKIN, *On the optimal evaluation of a set of bilinear forms*, Linear Algebra and Appl., 19 (1978), pp. 207-235.

[3] C. M. FIDUCCIA AND I. ZALCSTEIN, *Algebras having linear multiplicative complexity*, J. Assoc. Comput. Mach., 24 (1977), pp. 311–331.

[4] 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.

[5] ———, *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.

[6] ———, *Multiplication of polynomials over infinite fields*, Technical Report, Eberhard-Karls-Universität, Tübingen, 1978.

[7] H. F. DE GROOTE AND J. HEINTZ, *Minimal complexity of finite direct sums of division algebras*, Technical Report, J. W. Goethe-Universität, Frankfurt am Main, 1980.
[8] I. N. HERSTEIN, *Noncommutative Rings*, Carus Math. Monographs, 15, Mathematical Association of America, Washington, DC, 1973.
[9] J.-C. LAFON, *Base tensorielle des matrices de Hankel (ou de Toeplitz); applications*, Numer. Math., 23 (1975), pp. 349–361.
[10] S. LANG, *Algebra*, Addison-Wesley, Reading, MA, 1978.
[11] V. STRASSEN, *Vermeidung von Divisionen*, J. Reine Angew. Math., 264 (1973), pp. 184–202.
[12] S. WINOGRAD, *Some bilinear forms whose multiplicative complexity depends on the field of constants*, Math. Systems Theory, 10 (1977), pp. 169–180.
[13] ———, *On multiplication in algebraic extension fields*, Theoret. Comput. Sci., 8 (1979), pp. 359–377.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0008 $01.25/0

DIVIDE AND CONQUER HEURISTICS FOR MINIMUM WEIGHTED EUCLIDEAN MATCHING*

KENNETH J. SUPOWIT† AND EDWARD M. REINGOLD‡

Abstract. We consider the following problem: Given n points in a unit square in the Euclidean plane, find a matching of the points such that the cost (i.e., the sum of the lengths of the edges between matched points) is minimum. In particular, we present a class of linear time heuristic algorithms for this problem and analyze their worst case performance. The worst case performance of an algorithm is defined as the greatest possible cost, as a function of n, of the matching produced by the algorithm on a set of n points. Each of the algorithms studied here divides the unit square into a few smaller regions, and then is applied recursively to the points in each of these regions.

Key words. matching, graph algorithms, divide and conquer heuristics, computational geometry

1. Introduction.

If P is a set of n points (where n is even), then a *matching* of P is a set of $n/2$ edges such that each point of P is an endpoint of exactly one edge of the matching. The *cost* of a matching is the sum of the lengths of its edges. The *Euclidean matching problem* is to find minimum cost matchings when $P \subseteq [0, 1]^2$ and the length of an edge is the Euclidean distance.

An algorithm to solve this problem in time $\Theta(n^3)$ is known [5], [10]. In fact, this algorithm works not only for the Euclidean matching problem, but also for arbitrary weighted, undirected graphs. In this paper, we consider a class of heuristic matching algorithms that take advantage of geometry in order to give very fast running times. Although these heuristics do not always give minimum cost matchings, we are able to put an upper bound on the cost, as a function of n, of the matching that they produce.

For motivation, Euclidean matching has direct applications to minimizing the time required to draw networks on a mechanical plotter (as described in [12]). In this application the $\Theta(n^3)$ optimizing algorithm is unacceptable since n can be very large.

Given a matching algorithm, we define its *worst-case performance* as a function $f: \mathbb{N} \to \mathbb{R}$ such that

$$f(n) = \sup_P \{\text{the cost of the matching produced by the algorithm on } P\},$$

where "sup" is the least upper bound and P ranges over all sets of n points in the unit square. We use the supremum in the definition of worst-case performance because it is possible (since there are infinitely many n-point sets) that there is no n-point set for which the cost of the algorithm's matching is maximized.

Note that this definition of worst-case performance (which is taken from [2]) measures the *absolute* cost of the matching in the worst case. On the other hand, for many optimization problems (see, e.g., [6]) the absolute cost of a solution cannot be bounded in any meaningful way, and one must settle for an analysis of the worst-case

* Received by the editors September 1, 1980, and in revised form March 1, 1982. Preliminary versions of some of the results contained in this paper were presented at the Twelfth Annual ACM Symposium on Theory of Computing, April, 1980. This research was supported in part by the National Science Foundation under grants NSF MCS 77-22830 and NSF MCS 79-04897.

† Hewlett-Packard Laboratories, Computer Research Center, Palo Alto, California 94304. This research was conducted while this author was at the Department of Computer Science, University of Illinois at Urbana-Champaign.

‡ Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

cost of a heuristic *relative* to the cost of an optimal solution. Since the Euclidean matching problem lies in a bounded region, we are able to provide a more informative analysis of the worst-case behavior.

In support of this definition of the worst-case performance, we can consider the following remarks due to Avis [3]: Let GRE and OPT be the costs of the greedy heuristic solution (see [12]) and the optimal solution, respectively. A (more conventional) worst-case bound of

$$\frac{\text{GRE}}{\text{OPT}} \leqq \frac{4}{3} n^{\lg 1.5}$$

was derived in [12]. However, for the plotter application cited there, the increased plotter costs will be proportional to GRE − OPT, if the plotter works in a region of fixed size. The example that leads to the bound on GRE/OPT is not particularly bad in this measure because it is a constant, independent of n. This suggests that the relative measure is not as appropriate as the absolute measure used in this paper.

The (absolute) worst-case performance of a greedy Euclidean matching heuristic has been analyzed in [2]. The worst-case performance of several other heuristics, as well as a lower bound (namely $\sqrt{n}/\sqrt{\sqrt{12}}$) on the worst-case performance of the optimizing algorithm, is given in [15] (this issue, pp. 144–156). The worst-case performance of many heuristics for the Euclidean matching problem and the more general matching problem on weighted graphs whose weights satisfy the triangle inequality are given in [14]. The expected performance of certain Euclidean matching heuristics is analyzed in [9].

In this paper, we present and analyze the worst-case performance of a class of divide and conquer heuristic algorithms. Each of these algorithms operates by partitioning the region containing the points into subregions and recursively solving the smaller matching problems thus obtained. If a subregion contains an odd number of points, then all but one are matched and the odd point is then matched with an odd point in another subregion (there must be another such point since there are an even number of points in total). The expected cost of the matching produced by these heuristics when the n points are randomly chosen from a uniform distribution has been analyzed in [11].

In discussing the time required by the heuristics we intend the *real* RAM model of computation as defined by [13]. This model of computation is essentially the random-access machine as described in [1], except that each storage location can hold a single real number, and the following operations are available at unit cost:

1. Addition, subtraction, multiplication, and division on real numbers.
2. The "$\leqq$" comparison between real numbers.
3. Trigonometric, exponential, and logarithmic functions on real numbers.
4. Indirect addressing of memory.

The real RAM has become the standard model used in computational geometry.

All of the heuristics presented in this paper can be implemented to run in $\Theta(n \log n)$ time under the real RAM model. However, they can all be implemented to run in $\Theta(n)$ time if we make the model more powerful by also allowing the floor function to be available at unit cost. It is, of course, more realistic to consider the floor function as primitive on most computers. In view of this, it is fair to say that all of the heuristics presented here are linear time.

2. The rectangle heuristic. The first of the algorithms that we consider is the *rectangle algorithm* which works as follows. n points are given in the unit square $[0, 1]^2$. Consider the rectangle $[0, \sqrt{2}] \times [0, 1]$, which contains the unit square. If $n \geqq 2$

then this rectangle is bisected to form two congruent subrectangles, each with ratio $\sqrt{2}:1$ between the long and the short sides. The heuristic is applied recursively to each of the two subrectangles. In general, when applied to a rectangle R, the heuristic does as specified in Algorithm 1. As an example, in Fig. 1, $n = 4$. The first split is on the heavy solid line and the left half is then split along the dashed line. The matching produced is shown as jagged line segments.

There is one more detail of the algorithm: the level of recursion is not allowed to go beyond $\lceil \lg n \rceil$. More precisely, define a *rectangle* to be either the original $\sqrt{2}$ by 1 region, or one of two rectangular subregions with sides having ratio $\sqrt{2}:1$ into which a rectangle can be split. Let $R(P)$ denote the subset of P contained in rectangle R. Define the *level* of a rectangle R, denoted level (R), as follows: level $(R) = 0$, if R is the original $\sqrt{2} \times 1$ rectangle; otherwise, level $(R) =$ level $(R') + 1$, where R' is a rectangle that was bisected to form R and its mate. The rectangle heuristic is given in Algorithm 2.

ALGORITHM 1. The unbounded rectangle heuristic.

if R contains at least 2 points
then
1. bisect R to form rectangles R_1 and R_2, each having the ratio $\sqrt{2}:1$ between the long and short sides.
2. apply the heuristic to R_1.
3. apply the heuristic to R_2.
4. **if** R_1 and R_2 each contain an odd number of points
   then
   add the edge (p_1, p_2) to the matching, where p_1 is the point in R_1 not matched in Step 2, and p_2 is that of R_2 not matched in Step 3.

ALGORITHM 2. The rectangle heuristic.

if level $(R) \leqq \lceil \lg n \rceil$
then
  if R contains at least 2 points
    then do Steps 1–4 as described in Algorithm 1.
else
  arbitrarily match the points in R until
  at most one is left unmatched.

This restriction on the depth of recursion enables the algorithm to run in time $O(n \log n)$. The time is dominated by the partitioning of the points, and for each point $p \in R(P)$, we can decide in which half of R it lies by a single comparison. For each point p, we make at most one such comparison at each level of recursion; hence at most $\lceil \lg n \rceil$ such comparisons are made in total per point. The total time is therefore $O(n \log n)$.

Note that the algorithm can be implemented by unraveling the recursion; that is, by first partitioning the n points into the $2^{\lceil \lg n \rceil}$ level n rectangles and then performing the matching in a bottom-up fashion. Since the partitioning can be accomplished with only two divisions and one floor function per point, this version of the algorithm is linear under the real RAM model if the floor function is available at unit cost.

In order to analyze the worst-case cost of the matching produced by the algorithm as a function of n, we first find the worst-case cost for arbitrary sets of points in the

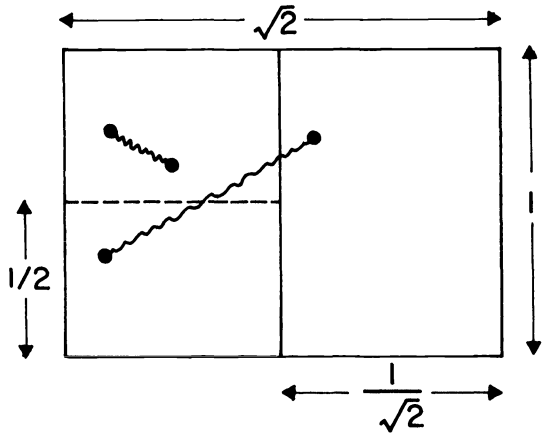


FIG. 1. *The rectangle heuristic performed on four points.*

$\sqrt{2}$ by 1 rectangle. Later, we will use this result to derive an upper bound on the cost for a set of points in a 1 by 1 square within the $\sqrt{2}$ by 1 rectangle.

If P is a set of points in the $\sqrt{2}$ by 1 rectangle, then let rcost (P) denote the sum of the lengths of the edges in the matching produced by the rectangle algorithm on P. For all $n \geqq 0$, let

$$C_n = \sup \{\text{rcost } (P): P \text{ is a set of } n \text{ points}\}.$$

By "set of points" we mean, here and throughout this section, a set of points in the $\sqrt{2} \times 1$ rectangle. Note that we are not primarily interested in C_n for odd n, but they are needed for the analysis. We first show that the restriction to $\lceil \lg n \rceil$ levels of recursion does not affect the C_n.

LEMMA 1. *Let P be a set of n points, $n \geqq 0$. Then there is a set of points Q for which $|Q| = n$ and* rcost $(Q) \geqq$ rcost (P) *and no level $\lceil \lg n \rceil + 1$ rectangle contains more than one point of Q.*

Proof. If no level $\lceil \lg n \rceil + 1$ rectangle contains more than one point of P, then we have nothing to prove. So let R_1 be a level $\lceil \lg n \rceil + 1$ rectangle such that $|R_1(P)| \geqq 2$. Then $R_2(P)$ is empty for some level $\lceil \lg n \rceil$ rectangle R_2, since otherwise $|P| \geqq 2^{\lceil \lg n \rceil} + 1 \geqq n$ (there are 2^l level l rectangles). Our strategy is to show that the points of P can be rearranged to produce a set Q of n points such that rcost$(Q) \geqq$ rcost(P) and $|R_1(Q)| = |R_1(P)| - 2$ and $|R_2(Q)| = 2$, but otherwise Q is just like P. Let $p_1, p_2 \in R_1(P)$ be points matched to each other by the algorithm. Define Q to be just like P *except* that p_1 and p_2 are not in Q, and instead Q has points p_1' and p_2' in diagonally opposite corners of R_2. This is illustrated in Fig. 2.

It is easily proved by induction on l that the dimensions of a level l rectangle are $1/\sqrt{2}^{l-1}$ by $1/\sqrt{2}^{l}$. The length of a diagonal in a level l rectangle is thus $\sqrt{3}/\sqrt{2}^{l}$, so

$$d(p_1, p_2) \leqq \frac{\sqrt{3}}{\sqrt{2}^{\lceil \lg n \rceil + 1}} < \frac{\sqrt{3}}{\sqrt{2}^{\lceil \lg n \rceil}} = d(p_1', p_2').$$

This "moving" of the two points into R_2 does not affect the way the algorithm matches of the other points. Therefore rcost $(Q) \geqq$ rcost (P). In this manner we continue to rearrange P until no level $\lceil \lg n \rceil + 1$ rectangle has more than one point in it. □

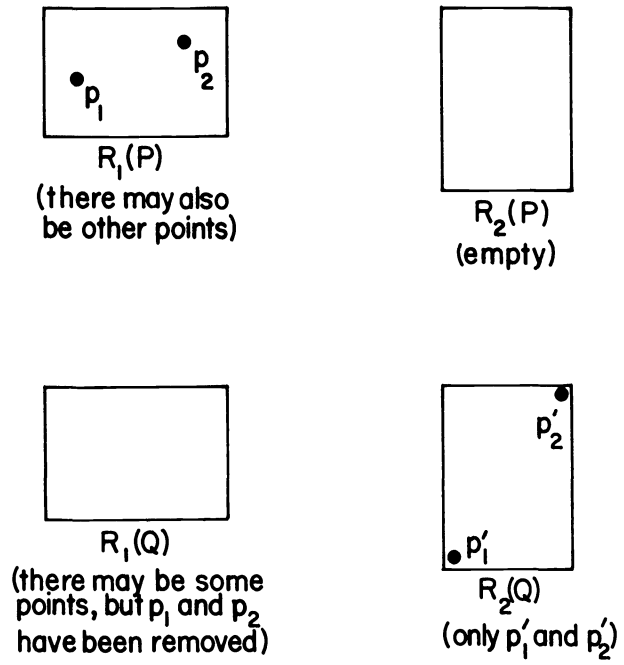


FIG. 2. *Illustration of the proof of Lemma* 1.

Lemma 1 tells us that we can analyze the algorithm as if there were no restriction on the depth of recursion, because such an assumption does not affect the worst-case costs, that is, the C_n.

Our strategy is to define a class of sets of points and then to show that these sets are the worst-case input for the algorithm. Specifically, we say that a set of points P is *balanced* if for all rectangles R such that $|R(P)| \geqq 2$, R splits into rectangles R_1, R_2 such that

(i) if 4 divides $|R(P)|$ then $|R_1(P)| = \frac{1}{2}|R(P)| - 1$ and $|R_2(P)| = \frac{1}{2}|R(P)| + 1$,

(ii) if 4 does not divide $|R(P)|$ then $|R_1(P)| = \lfloor \frac{1}{2}|R(P)| \rfloor$, $|R_2(P)| = \lceil \frac{1}{2}|R(P)| \rceil$,

(iii) if $|R(P)|$ is even then the point p_1 left unmatched by the algorithm on R_1 and the point p_2 left unmatched by R_2 are in diagonally opposite corners of R.

In other words, for a balanced set, each rectangle R with an even, nonzero number of points splits odd-odd, with the two subrectangles having almost the same number of points, and the edge produced at the end of the algorithm on R is along one of the diagonals of R. Intuitively, one might suspect such a set P to be a worst case for the algorithm; this is indeed the case, as is proved in the next two lemmas.

LEMMA 2. *Let P be a set of n points, $n \geqq 0$, n even. Then there is a set Q of n points such that* rcost $(Q) \geqq$ rcost (P) *and for all rectangles R such that $|R(Q)| \geqq 1$, we have*:

1. *$|R(Q)|$ even implies R is split into R_1, R_2 such that $|R_1(Q)|$, $|R_2(Q)|$ are both odd, and such that R_1 and R_2 each leave unmatched points of Q in diagonally opposite corners of R.*

2. *$|R(Q)|$ odd implies R leaves a point of Q unmatched in one of its own corners.*

3. *$|R(Q)| \geqq 2$ implies that the two subrectangles of R each contain at least one point of Q.*

Proof. We will construct Q by rearranging P (in the manner of Lemma 1). The process of rearrangement is as follows:

First, we consider all rectangles R containing a single point. Let R be such a rectangle, $R(P)=\{p_1\}$. Since n is even, the algorithm must match p_1 to some other point $p_2 \in P$ outside of R. Define P' to be like P except that instead of p_1, P' has point p_1' in the corner of R which is farthest from p_2. The situation is shown in Fig. 3. Hence $d(p_1, p_2) \leqq d(p_1', p_2)$. Since this "moving" of p_1 to p_1' affects no other matches made by the algorithm on P, we have rcost $(P) \leqq$ rcost (P').

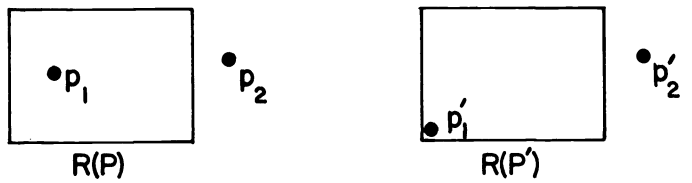


FIG. 3. *Lemma 2, rearranging a rectangle that contains one point.*

Having so rearranged, if necessary, all rectangles containing exactly one point of P, we now consider those containing two points. Let R be such a rectangle, $R(P)=\{p_1, p_2\}$. Since $|R(P)|$ is even, the arrangement of the points of P within R does not affect the matching of any points outside of R. Therefore if p_1 and p_2 are not in diagonally opposite corners of R, then move them there by letting P' be like P except that instead of having p_1 and p_2, P' has p_1' and p_2' in diagonally opposite corners of R (see Fig. 4). Since $d(p_1, p_2) < d(p_1', p_2')$, we have rcost $(P) <$ rcost (P'), as desired.

Now assume we have rearranged all rectangles R with $|R(P)| \leqq k$ for some integer $k \geqq 2$. We will now rearrange each rectangle R such that $|R(P)| = k+1$. Let R be such a rectangle.

FIG. 4. *Lemma 2, rearranging a rectangle that contains two points.*

Case 1. $k+1$ is odd. Then R splits into rectangles R_1, R_2 such that $|R_1(P)|$ is odd and $|R_2(P)|$ is even.

Case 1.1. $|R_2(P)| = 0$. Then $|R_1(P)| \geqq 3$. Therefore R_1 splits into some rectangles S_1, S_2 such that $|S_1(P)| \geqq 2$. Let p_1, p_2 be two points in S_1 matched to each other by the algorithm (such points must exist since S_1 leaves at most one point unmatched, in which case we would have $|S_1(P)| \geqq 3$). Now let P' be exactly like P except that P' has points p_1' and p_2' in opposite corners of R_2, and no point at p_1 or p_2 (see Fig. 5). Moving p_1 and p_2 out of S_1 does not affect the matching of the other points in R_1. Also, $d(p_1, p_2) < d(p_1', p_2')$, so that rcost $(P) <$ rcost (P'). $|R_1(P)|$ is now less than $k+1$, and we rearrange R_1, and then rearrange R, using Case 1.2.

Case 1.2. $|R_2(P)| > 0$. Then $|R_1(P)|$, $|R_2(P)| < k+1$ and hence both R_1 and R_2 have already been rearranged. In particular, R_1 leaves an unmatched point p_1 in one of its corners. The algorithm matches p_1 to some point p_2 outside of R. If p_1 is already in a corner of R, then we have nothing to rearrange. So, assume p_1 is not in a corner

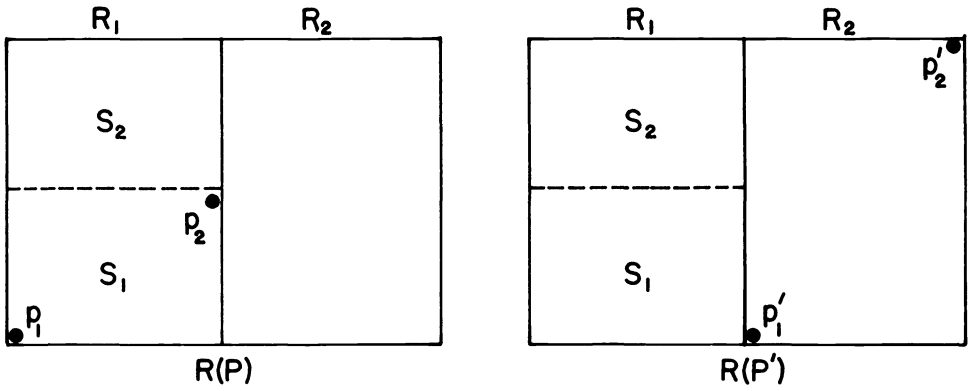


FIG. 5. *Lemma* 2, *Case* 1.1.

of R (an example of such a situation is shown in Fig. 6). Now let P' be like P except that the points in R_1 have been reflected (vertically and/or horizontally) and perhaps swapped with those in R_2, so that p_1 is now in the extreme corner from p_2 (see Fig. 7). This reflecting and swapping has no effect on the cost of the matching of the points in $R(P)$ other than p_1. Therefore $\text{rcost}(P) < \text{rcost}(P')$, and we continue with the rearranging.

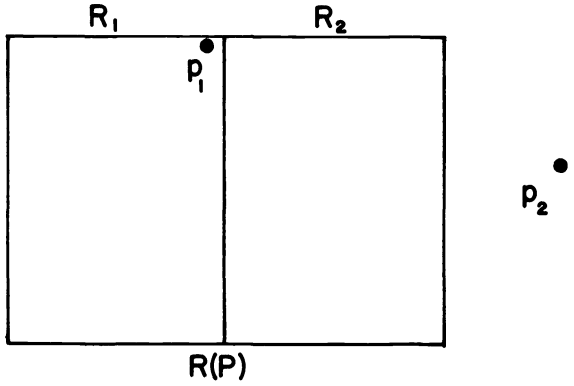


FIG. 6. *Lemma* 2, *Case* 1.2, *before rearranging.*

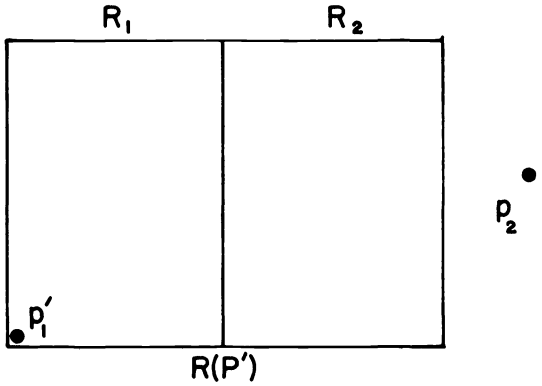


FIG. 7. *Lemma* 2, *Case* 1.2, *after rearranging.*

Case 2. $k+1$ is even. Let R_1, R_2 be the subrectangles of P, and assume, without loss of generality, that $|R_1(P)| \geqq |R_2(P)|$.

Case 2.1. $|R_2(P)| = 0$. Then proceed exactly as in Case 1.1.

Case 2.2. $|R_2(P)| > 0$. Then $|R_1(P)|$, $|R_2(P)| < k+1$. Therefore R_1 and R_2 have already been rearranged. Since $|R(P)| = |R_1(P)| + |R_2(P)|$ is even, we have two cases:

Case 2.2.1. $|R_1(P)|$, $|R_2(P)|$ are both even. This is the most interesting of all the cases, since it is the only one that depends on the shape of the rectangles. Since R_1, R_2 already satisfy the desired properties, we have the situation pictured in Fig. 8. That is, R is a rectangle of size $a\sqrt{2}$ by a for some real $a > 0$. R_1, a subrectangle of R, matches points p_1 and p_2 in opposite corners of R_1. R_2 similarly matches p_3 and p_4 in its opposite corners. S_2 is the even subrectangle of the subrectangle of R_1 which leaves p_2 unmatched. S_1 is the odd subrectangle of the subrectangle of R_2 which leaves p_3 unmatched. (We say a rectangle is *even* if it contains an even number of points, otherwise it is *odd*.)

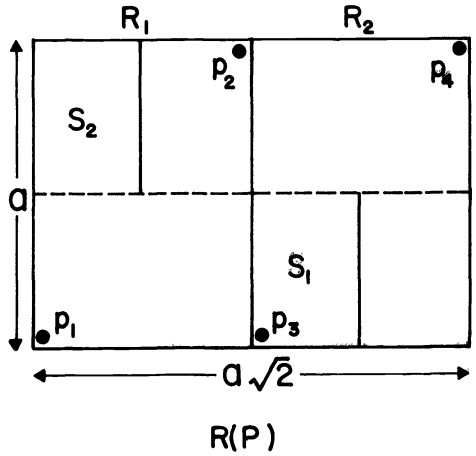


FIG. 8. *Lemma* 2, *Case* 2.2.1, *before rearranging.*

Now rearrange P into P' by swapping the points in S_1 with those in S_2, as shown in Fig. 9. Then for some real $c \geqq 0$,

$$\text{rcost}(P) = d(p_1, p_2) + d(p_3, p_4) + c,$$

and

$$\text{rcost}(P') = d(p_1, p_4) + d(p_2, p_3') + c.$$

Now

$$d(p_1, p_2) = d(p_3, p_4) = \sqrt{(a\sqrt{2}/2)^2 + a^2} = a\sqrt{3}/\sqrt{2}.$$

Also,

$$d(p_1, p_4) = \sqrt{a^2 + (a\sqrt{2})^2} = a\sqrt{3},$$

and

$$d(p_2, p_3') = \sqrt{(a/2)^2 + (a\sqrt{2}/2)^2} = a\sqrt{3}/2.$$

Therefore,

$$\text{rcost}(P) = 2(a\sqrt{3}/\sqrt{2}) + c = a\sqrt{6} + c < a3\sqrt{3}/2 + c = a\sqrt{3} + a\sqrt{3}/2 + c = \text{rcost}(P'),$$

and we continue to rearrange.

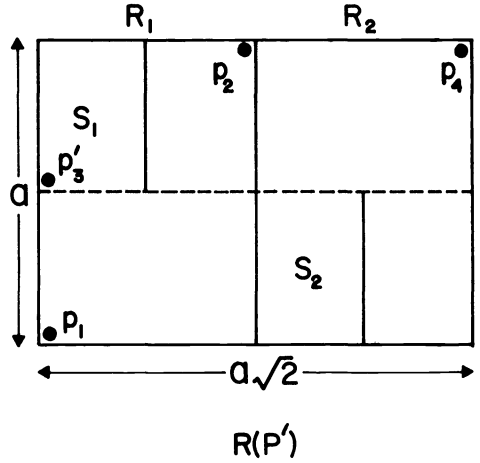


FIG. 9. *Lemma* 2, *Case* 2.2.1, *after rearranging.*

Case 2.2.2. $|R_1(P)|$, $|R_2(P)|$ are both odd. Since $|R_1(P)|$, $|R_2(P)| < k+1$, we already have that R_1 leaves a point p_1 unmatched in one of its corners, and that R_2 leaves a point p_2 unmatched in one of its corners. If p_1 and p_2 are not in opposite corners of R, then the appropriate reflections of $R_1(P)$ and $R_2(P)$ will produce a set P' of cost greater than that of P.

Thus, we continue to rearrange P until we have rearranged the original, level 0, rectangle. This final arrangement is Q, satisfying the properties stated in the lemma. □

The set Q constructed from P in Lemma 1 has some of the properties of a balanced set, but not all. The next lemma rearranges this Q so as to be balanced, without changing rcost (Q). This will complete the proof that balanced sets constitute a worst case for the algorithm.

LEMMA 3. *Let P be a set of n points,* $n \geq 0$, *n even. Then there is a set of n points* Q_1 *such that* rcost $(Q_1) \geq$ rcost (P) *and* Q_1 *is balanced.*

Proof. Let Q be a set satisfying the properties stated in Lemma 2. We will rearrange Q into a new set Q_1 such that for each rectangle R, if R_1, R_2 are the two subrectangles of R then $||R_1(Q_1)| - |R_2(Q_1)|| \leq 2$. Furthermore, Q_1 will still have the property of Lemma 2 that even, nonempty rectangles split odd-odd leaving unmatched points in opposite corners. Together, these properties tell us that Q_1 is balanced.

First, note that all rectangles R such that $|R(Q)| = 1$ or 2 are already balanced, and hence need no rearranging. Now assume that we have balanced all rectangles R such that $|R(Q)| \leq k$ for some integer k. Let R be a rectangle such that $|R(Q)| = k+1$. Let R_1, R_2 be the subrectangles of R. Let S_1, T_1 be the subrectangles of R_1. Let S_2, T_2 be the subrectangles of R_2.

Case 1. R is even. Then R_1, R_2 are odd, by our choice of Q. Assume, without loss of generality, that T_1 and S_2 are both odd (see Fig. 10). Then swap $S_1(Q)$ with $T_2(Q)$ to get, in the notation used in the proof of Lemma 2, what is pictured in Fig. 11. R_1, R_2 were balanced before this swap, since $|R_1(Q)|$, $|R_2(Q)| \leq k$. Therefore, letting $s_1 = |S_1(Q)|$, $s_2 = |S_2(Q)|$, $t_1 = |T_1(Q)|$, and $t_2 = |T_2(Q)|$, we have that $|s_1 - t_1| = 1$ and $|s_2 - t_2| = 1$. Therefore

$$||R_1(Q')| - |R_2(Q')|| = |(t_1 + t_2) - (s_1 + s_2)| \leq 2,$$

which is what we want. Now this swapping of $S_1(Q)$ with $T_2(Q)$ may have made R_1 or R_2 (or both) unbalanced. Therefore we now rearrange R_1 and R_2 (this process

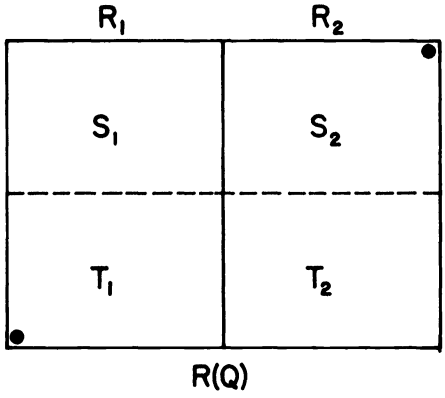


FIG. 10. *Lemma* 3, *Case* 1, *before rearranging.*

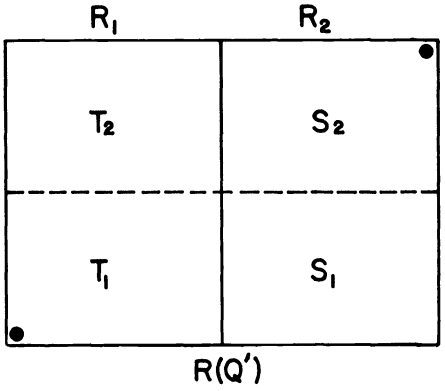


FIG. 11. *Lemma* 3, *Case* 1, *after rearranging.*

eventually terminates since $|R_1(Q')|, |R_2(Q')| < |R(Q')|$). Thus R is now balanced, and we continue to rearrange other rectangles.

Case 2. R is odd. Assume, without loss of generality, that R_1 is even and R_2 is odd. Define s_1, s_2, t_1, t_2 as in Case 1. By the choice of Q (in particular, property 3 of Lemma 2), we have $|R_1(Q)| > 0$ and hence $|R_1(Q)|, |R_2(Q)| \leqq k$. Therefore s_1, t_1 are odd. Assume, without loss of generality, that s_2 is odd and $s_1 \geqq t_1$. This situation is illustrated in Fig. 12.

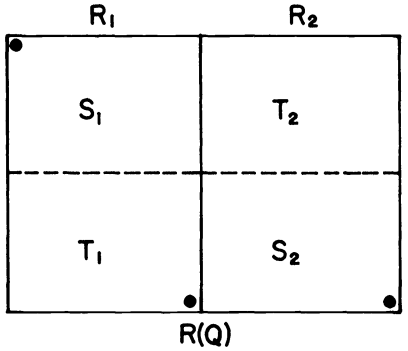


FIG. 12. *Lemma* 3, *Case* 2, *before rearranging.*

Case 2.1. $s_2 \geqq t_2$. Then since R_2 is balanced, we have $s_2 = t_2 + 1$. Swap $S_1(Q)$ with $S_2(Q)$ to get Q', as shown in Fig. 13. Note that we also may need to rotate $S_2(Q)$ so that its unmatched point is diagonally opposite that of T_1. Since $0 \leqq s_1 - t_1 \leqq 2$, we have

$$\begin{aligned} \big||R_1(Q')| - |R_2(Q')|\big| &= |(s_2 + t_1) - (s_1 + t_2)| \\ &= |(s_2 - t_2) + (t_1 - s_1)| = |1 + (t_1 - s_1)| \leqq 1, \end{aligned}$$

as desired.

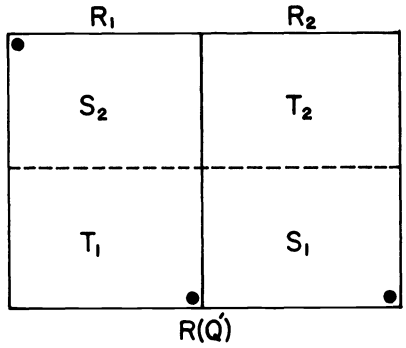


FIG. 13. *Lemma* 3, *Case* 2.1.

Case 2.2. $s_2 < t_2$. Then $s_2 = t_2 - 1$. Swap $T_1(Q)$ with $S_2(Q)$ to get Q', obtaining (after possibly rotating) the situation shown in Fig. 14. Now

$$\begin{aligned} \big||R_1(Q')| - |R_2(Q')|\big| &= |(s_1 + s_2) - (t_1 + t_2)| \\ &= |(s_2 - t_2) + (s_1 - t_2)| \\ &= |-1 + (s_1 - t_1)| \leqq 1, \end{aligned}$$

as desired. Continue to rearrange other rectangles.

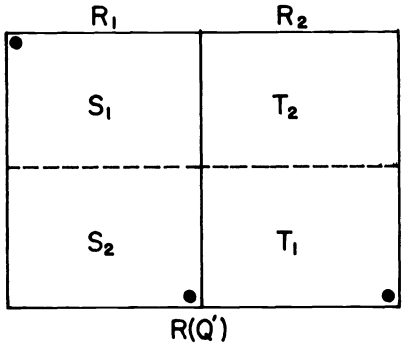


FIG. 14. *Lemma* 3, *Case* 2.2.

Finally, after balancing the main, level 0 rectangle, let Q_1 be the resulting arrangement, and we are finished. Note that the rearrangement can change neither the cost of the set, nor the assumed properties of Q. □

Thus the balanced sets constitute the worst case for the algorithm. Clearly, all balanced n-point sets have the same cost C_n. We now analyze the C_n:

$$C_0 = C_1 = 0, \quad C_2 = \sqrt{3}, \quad C_3 = \sqrt{3}/\sqrt{2}.$$

A balanced set of $4n$ points splits into two balanced sets—one with $2n+1$ points, and one with $2n-1$—and matches two points in its diagonally opposite corners. Thus for all $n \geqq 1$,

$$C_{4n} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n-1}) + \sqrt{3}.$$

The factor $1/\sqrt{2}$ is to scale down the cost from the $\sqrt{2} \times 1$ region to the $1 \times 1/\sqrt{2}$ region. Similarly, for all $n \geqq 1$

$$C_{4n+1} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n}).$$

And for all $n \geqq 0$,

$$C_{4n+2} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n+1}) + \sqrt{3}, \qquad C_{4n+3} = \frac{1}{\sqrt{2}}(C_{2n+2} + C_{2n+1}).$$

Although we have not been able to solve this recurrence for C_n, we show in the Appendix that

$$C_n \leqq \left(1 + \frac{1}{\sqrt{2}}\right)\sqrt{n} + \sqrt{3} - \sqrt{6} + O\left(\frac{1}{\sqrt{n}}\right)$$

$$\approx 1.707\sqrt{n} - 0.717 + O\left(\frac{1}{\sqrt{n}}\right),$$

and that

$$C_n \geqq \left(\frac{1}{2\sqrt{2}} - \frac{3}{2} + 2\sqrt{2}\right)\sqrt{n} + \sqrt{3} - \sqrt{6} - o(1)$$

$$\approx 1.682\sqrt{n} - 0.717 - o(1).$$

Furthermore, we show that the upper bound is achievable in that for an infinite class of n,

$$C_n \geqq \left(1 + \frac{1}{\sqrt{2}}\right)\sqrt{n} + \sqrt{3} - \sqrt{6} - o\left(\frac{1}{\sqrt{n}}\right).$$

So far we have considered the performance of the rectangle algorithm on points in the $\sqrt{2}$ by 1 rectangle. However, the fixed region matching problem is usually considered on the unit square. Therefore we now adapt the rectangle algorithm to the unit square as follows: Given a set of n points P in the unit square (i.e., for all $(x, y) \in P$, $0 \leqq x \leqq 1$ and $0 \leqq y \leqq 1$), we perform the rectangle algorithm by considering P to be in a $\sqrt{2} \times 1$ rectangle as shown in Fig. 15. In that figure, the unit square is depicted by a solid line; the $\sqrt{2} \times 1$ rectangle is shown as a dashed line.

For the analysis of this rectangle algorithm applied to the unit square, choose some even integer $k \geqq 0$ and let $r = \lceil \sqrt{2}^{k-1} \rceil$ and $s = \sqrt{2}^k$. Note that since k is even, each level k rectangle has vertical dimension $1/\sqrt{2}^k$ and horizontal dimension $1/\sqrt{2}^{k-1}$. Therefore the unit square, and hence P, lies within the leftmost set of rs level k rectangles, as shown in Fig. 16. Let $d = r/\sqrt{2}^{k-1}$. Our strategy is to derive an upper bound on the cost of the rectangle algorithm on an arbitrary set in the d by 1 rectangle. Since $d \geqq 1$, this bound will also be an upper bound on rcost (P).

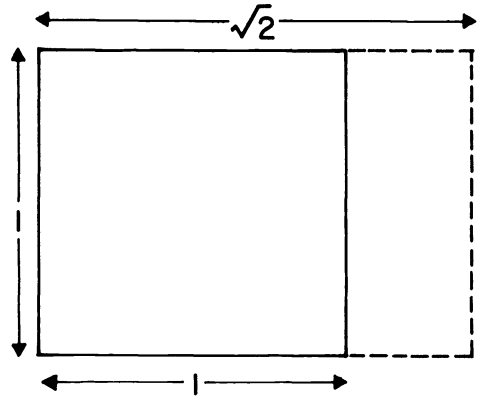


FIG. 15. *Applying the rectangle algorithm to the unit square.*

Let Q be a set of n points in the $d \times 1$ region, and let

$$\text{rcost}_k(Q) = \text{rcost}(Q) - \sum_{i=0}^{k-1} \left[\begin{array}{l} \text{the sum of the lengths of all edges produced at} \\ \text{the } i\text{th level of recursion by the algorithm on } Q \end{array} \right].$$

Since there are 2^i level i rectangles, and since the length of an edge produced at the ith of recursion is at most $\sqrt{3}/\sqrt{2^i}$, we have

$$\text{rcost}_k(Q) \geqq \text{rcost}(Q) - \sum_{i=0}^{k-1} 2^i \frac{\sqrt{3}}{\sqrt{2^i}} = \text{rcost} - \frac{\sqrt{3}}{\sqrt{2}-1}(\sqrt{2^k}-1).$$

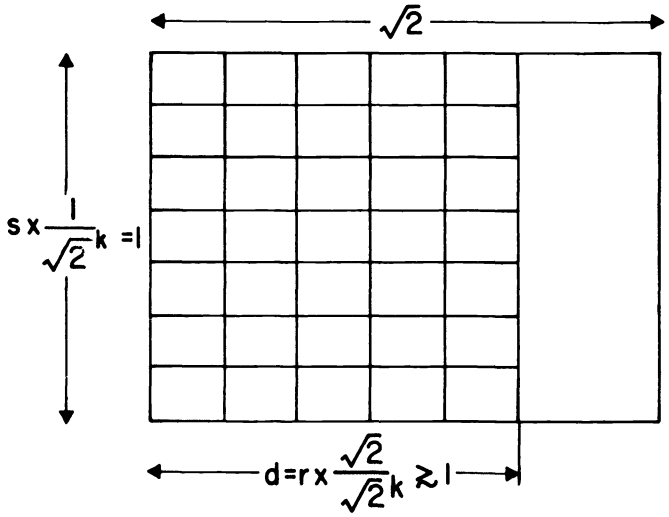


FIG. 16. *Covering the unit square with level k rectangles.*

Therefore

$$\text{rcost}(Q) \leqq \text{rcost}_k(Q) + \frac{\sqrt{3}}{\sqrt{2}-1}(\sqrt{2^k}-1).$$

We now derive an upper bound on $\text{rcost}_k(Q)$, the sum of the lengths of the edges produced at levels at or beyond k. There are $t = rs$ level k rectangles that compose the d by 1 region containing Q. Call these rectangles R_j, $1 \leqq j \leqq t$, and let $n_j = |R_j(Q)|$.

By the results of the Appendix the sum of the lengths of the edges produced within R_j is at most

$$\frac{1}{\sqrt{2}^k} C_{n_j} = \frac{1}{\sqrt{2}^k}\left[\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n_j}+\sqrt{3}-\sqrt{6}+O\left(\frac{1}{\sqrt{n_j}}\right)\right].$$

The factor $1/\sqrt{2}^k$ is to scale the cost down to level k. Therefore

$$\begin{aligned}\text{rcost}_k(Q) &\leqq \frac{1}{\sqrt{2}^k}\sum_{j=1}^{t}\left[\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n_j}+\sqrt{3}-\sqrt{6}+O\left(\frac{1}{\sqrt{n_j}}\right)\right]\\ &= \frac{1}{\sqrt{2}^k}\left(1+\frac{1}{\sqrt{2}}\right)\sum_{j=1}^{t}\sqrt{n_j}+O(t)\\ &= \frac{1}{\sqrt{2}^k}\left(1+\frac{1}{\sqrt{2}}\right)\left(\sum_{j=1}^{t-1}\sqrt{n_j}+\sqrt{n-\sum_{j=1}^{t-1} n_j}\right)+O(t).\end{aligned}$$

Define the function $f: \mathbb{R}^{t-1} \to \mathbb{R}$ by

$$f(x_1, x_2, \cdots, x_{t-1}) = \sum_{j=1}^{t-1}\sqrt{x_j}+\sqrt{n-\sum_{j=1}^{t-1} x_j}.$$

Taking partial derivatives shows that f is maximized at

$$x_1 = x_2 = \cdots = x_{t-1} = \frac{n}{t}.$$

Therefore

$$\begin{aligned}\text{rcost}_k(Q) &\leqq \frac{1}{\sqrt{2}^k}\left(1+\frac{1}{\sqrt{2}}\right)t\sqrt{n/t}+O(t) = \frac{1}{\sqrt{2}^k}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{rs}\sqrt{n}+O(rs)\\ &= \frac{1}{\sqrt{2}^k}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{\frac{d\sqrt{2}^k}{\sqrt{2}}\sqrt{2}^k}\sqrt{n}+O(2^k)\\ &= \frac{\sqrt{d}}{\sqrt{\sqrt{2}}}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n}+O(2^k).\end{aligned}$$

Therefore

$$\text{rcost}(Q) \leqq \text{rcost}_k(Q)+O(2^k) = \frac{\sqrt{d}}{\sqrt{\sqrt{2}}}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n}+O(2^k).$$

By the definition of d, we have that $d \to 1$ as $k \to \infty$. Thus, for all $\varepsilon > 0$, there is a least value of k, $k(\varepsilon)$, for which $d < 1+\varepsilon$; the $O(2^k)$ term becomes $O(2^{k(\varepsilon)}) = O(1)$ as $n \to \infty$. Thus we have

$$\text{rcost}(Q) \leqq (1+\varepsilon)\frac{1}{\sqrt{\sqrt{2}}}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n}+O(1) \approx (1+\varepsilon)1.436\sqrt{n}+O(1).$$

Notice that the $O(1)$ term grows unboundedly as $\varepsilon \to 0$. If we take $k = 10$, for instance, then $r = 23$, $s = 32$, and $d = 23\sqrt{2}/32 \approx 1.016$, we get that

$$\text{rcost}(Q) \leqq \sqrt{23/32}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n}+O(1) \approx 1.447\sqrt{n}+O(1).$$

In order to show the tightness of this bound, we again choose some even $k \geqq 0$, but this time let $r = \lfloor \sqrt{2}^{k-1} \rfloor$, $d = r/\sqrt{2}^{k-1} \leqq 1$, and $s = \sqrt{2}^{k}$ as before. Thus the unit square contains a d by 1 region. Construct a set Q' in the d by 1 region, so that each of the rs level k rectangles in that region contains a balanced n/rs point set. We choose $n = |Q'|$ so that $n/rs = b_i$ for some i (where b_i is as defined in the Appendix), thus making $C_{n/rs}$ asymptotic to $(1+1/\sqrt{2})\sqrt{n/rs}$. An analysis similar to the above shows that

$$\text{rcost}(Q') \geqq \frac{\sqrt{d}}{\sqrt{\sqrt{2}}}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n} - O(2^k).$$

Hence for all $\varepsilon > 0$, there is a set Q' of n points in the unit square for which

$$\text{rcost}(Q') \geqq (1-\varepsilon)\frac{1}{\sqrt{\sqrt{2}}}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n} - O(1) \approx (1-\varepsilon)1.436\sqrt{n} - O(1).$$

The reader may wonder why we did not simply choose some k such that the 1 by 1 square can be exactly tessellated by level k rectangles (i.e. we would have $d = 1$). Unfortunately, as is easily shown, no such k exists.

In summary,

$$\inf\{x: \text{for all } n\text{-point sets } P \text{ in the unit square, rcost}(P) \leqq x\sqrt{n} + o(\sqrt{n})\}$$
$$= \frac{1}{\sqrt{\sqrt{2}}}\left(1+\frac{1}{\sqrt{2}}\right) \approx 1.436,$$

where "inf" denotes the greatest lower bound.

3. The triangle algorithm. A square can be partitioned into two equal-sized 45°-45°-90° triangles. Also, a 45°-45°-90° triangle can be partitioned into two equal-sized 45°-45°-90° subtriangles. This suggests a second partitioning algorithm, which we call the *triangle algorithm*: given a set P of n points in the unit square, do exactly as the rectangle algorithm, except that when a region is split, it is split into two equal-sized 45°-45°-90° triangles. An example with $n = 4$ is shown in Fig. 17; the first split is along the main diagonal (shown as a solid line) and the second split is shown as a dashed line. The matching produced is shown in jagged lines.

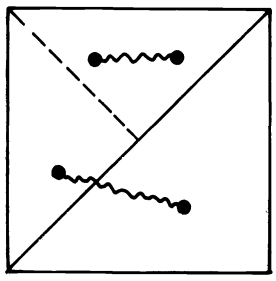


FIG. 17. *The triangle algorithm executed on four points.*

In analogy to the previous section, define a *triangle* T to be either one of the two main 45°-45°-90° triangles into which the unit square is split, or one of the two 45°-45°-90° subtriangles into which a triangle may be split. Furthermore, level (T) is defined as follows: level $(T) = 0$ if T is one of the two main triangles (i.e. if T has hypotenuse length $\sqrt{2}$); otherwise, level (T) = level $(T')+1$, where T' is the triangle that was bisected to form T and its mate.

Note that the level of a triangle is one less than the level of recursion on which the triangle lies (in contrast to the level of a rectangle in the previous section, which equals the level of recursion on which it lies). We define level in this way because our strategy is to analyze the worst-case cost of points in a main triangle, and then use that result to analyze the worst-case cost for points in the unit square.

If P is a set of points in the unit square, then let tcost (P) be the sum of the lengths of the edges in the matching produced by the triangle algorithm on P. For all $n \geqq 0$, let

$$E_n = \sup \{\text{tcost } (P): P \text{ is a set of } n \text{ points in a level 0 triangle}\}$$

and let

$$F_n = \sup \{\text{tcost } (P): P \text{ is a set of } n \text{ points in the unit square}\}.$$

As mentioned above, we will first analyze the E_n and then use that result to analyze the F_n.

First note that as for the rectangle algorithm, we can restrict the levels of recursion to at most $\lceil \lg n \rceil$ and so enable the algorithm to run in time $O(n \log n)$, or $O(n)$ if the floor function is available at unit cost. This restriction does not affect the worst-case cost, as can be proved by an argument parallel to that in Lemma 1.

Throughout the analysis of the E_n, let "set of points" mean a set of points in a main triangle. If T is a triangle, and P a set of points, then let $T(P)$ denote the subset of P contained in T. Define the notion of balanced exactly as in the analysis of the rectangle algorithm, except substituting the word "triangle" for "rectangle," and understanding the "diagonally opposite corners" of a triangle to be its two 45° corners. In analogy to the rectangle results, balanced sets are the worst case for the triangle algorithm.

LEMMA 2'. *Let P be a set of $n \geqq 0$ points, n even. Then there is a set of n points Q such that* tcost $(Q) \geqq$ tcost (P) *and for all triangles T such that $|T(Q)| \geqq 1$, we have*

1. *$|T(Q)|$ even implies T is split into T_1, T_2 such that $|T_1(Q)|$, $|T_2(Q)|$ are both odd, and such that T_1 and T_2 each leave unmatched points of Q in a 45° corner of T.*
2. *$|T(Q)|$ odd implies T leaves a point of Q unmatched in one of its own 45° corners.*
3. *$|T(Q)| \geqq 2$ implies the two subtriangles of T each contain at least one point of Q.*

Proof. By a rearranging argument, very similar to that of Lemma 2. We will give only the most important case of the argument, that corresponding to Case 2.2.1 of the proof of Lemma 2; the remaining cases are left to the reader. Let T be a triangle such that $|T(P)|$ is even. Let $T_1(P)$ and $T_2(P)$ be the subtriangles of T, and assume that $|T_1(P)|$ and $|T_2(P)|$ are both even and greater than 0. Assume that both $T_1(P)$ and $T_2(P)$ have already been rearranged to satisfy the stated properties. The situation is as pictured in Fig. 18. Let h be the length of the hypotenuse of T. T_1 matches points p_1 and p_2 in its opposite corners. T_2 matches points p_3 and p_4 in its opposite corners. S_2 is the even subtriangle of the subtriangle of T_1 which leaves p_2 unmatched. S_1 is the odd subtriangle of the subtriangle of T_2 which leaves p_3 unmatched.

Now rearrange P into P' by swapping the points in S_1 with those in S_2, as shown in Fig. 19. Then for some real constant $c > 0$,

$$\text{tcost } (P) = d(p_1, p_2) + d(p_3, p_4) + c$$

and

$$\text{tcost } (P') = d(p_1, p_4) + d(p_2, p_3') + c.$$

Also,

$$d(p_1, p_2) = d(p_3, p_4) = h/\sqrt{2}, \quad d(p_1, p_4) = h, \quad d(p_2, p_3') = h/2.$$

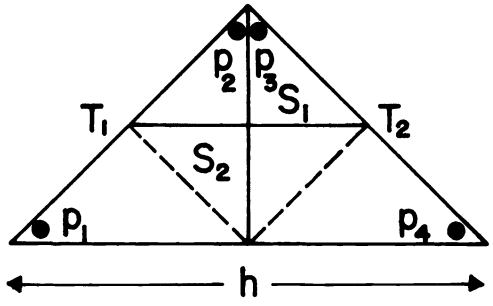


FIG. 18. *Lemma 2', before rearranging.*

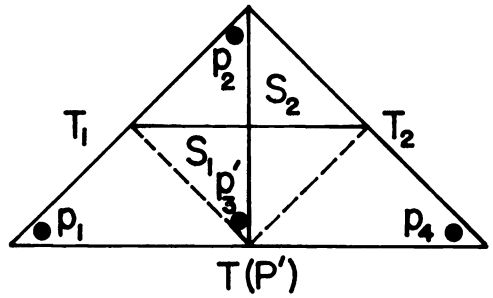


FIG. 19. *Lemma 2', after rearranging.*

Thus

$$\text{tcost}(P) = h\sqrt{2} + c < \tfrac{3}{2}h + c = \text{tcost}(P'),$$

as desired.

The other cases of the rearrangement are straightforward. □

LEMMA 3'. *Let P be a set of $n \geqq 0$ points, n even. Then there is a set of n points Q_1 such that* $\text{tcost}(Q_1) \geqq \text{tcost}(P)$ *and Q_1 is balanced.*

Proof. Identical to that for Lemma 3, substituting "triangle" for "rectangle" throughout. □

Thus for all even $n \geqq 0$, we have $E_n = \text{tcost}(P)$, where P is a balanced n-point set. The length of a level l hypotenuse is $\sqrt{2}/\sqrt{3}$ times the length of the diagonal in a level l rectangle so that for all even $n \geqq 0$,

$$E_n = \frac{\sqrt{2}}{\sqrt{3}} C_n \leqq \frac{\sqrt{2}}{\sqrt{3}} \left[\left(1 + \frac{1}{\sqrt{2}}\right)\sqrt{n} + \sqrt{3} - \sqrt{6} + O\left(\frac{1}{\sqrt{n}}\right) \right].$$

Note that for all odd $n \geqq 0$, $E_n \leqq E_{n-1}$. To see this, let P be a set of points, such $n = |P|$ is odd. Then there is some $p_1 \in P$ such that p_1 is not matched to any other point by the algorithm. Then $\text{tcost}(P) = \text{tcost}(P - \{p_1\})$, and hence $E_n \leqq E_{n-1}$. Therefore, for all $n \geqq 0$,

$$E_n \leqq \frac{\sqrt{2}}{\sqrt{3}} \left(1 + \frac{1}{\sqrt{2}}\right)\sqrt{n} + O(1).$$

We now analyze the F_n, which are our primary interest. Let $n \geqq 0$, and let P be a set of n points in the unit square. The square is split into two main triangles, one

with m points and one with $n-m$ points, for some m such that $0 \leqq m \leqq n$. Thus

$$\begin{aligned}\text{tcost}(P) &\leqq \max_{0\leqq m\leqq n}\{E_m + E_{n-m}\} + \sqrt{2} \\ &\leqq \max_{0\leqq m\leqq n}\left\{\frac{\sqrt{2}}{\sqrt{3}}\left(1+\frac{1}{\sqrt{2}}\right)(\sqrt{m}+\sqrt{n-m})\right\} + O(1).\end{aligned}$$

Treating the expression $\sqrt{m}+\sqrt{n-m}$ as a function of a real variable m and differentiating shows that $\sqrt{m}+\sqrt{n-m}$ is maximized at $m = n/2$. Therefore,

$$\text{tcost}(P) \leqq \frac{\sqrt{2}}{\sqrt{3}}\left(1+\frac{1}{\sqrt{2}}\right)2\sqrt{n/2} + O(1) = \frac{2}{\sqrt{3}}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n} + O(1).$$

Thus for all $n \geqq 0$,

$$F_n \leqq \frac{2}{\sqrt{3}}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n} + O(1) \approx 1.971\sqrt{n} + O(1).$$

This bound is asymptotically achievable for an infinite class of n defined by $n = 2b_r$ for some $r \geqq 0$ (b_r as defined in the Appendix), for which we can construct a set P such that the unit square splits into main triangles T_1, T_2 such that $T_1(P)$ and $T_2(P)$ are each balanced sets of b_r points. Therefore, since

$$\frac{C_{b_r}}{\sqrt{b_r}} \to 1 + \frac{1}{\sqrt{2}} \quad \text{as } r \to \infty,$$

as shown in the Appendix, we have

$$\frac{\text{tcost}(P)}{\sqrt{2}} \to \frac{2}{\sqrt{3}}\left(1+\frac{1}{\sqrt{2}}\right) \approx 1.971 \quad \text{as } n \to \infty.$$

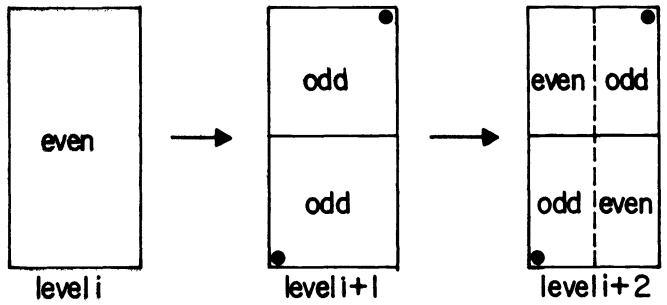
4. The square-rectangle algorithm. Our third divide and conquer method, the *square-rectangle algorithm*, works just like the rectangle or triangle heuristics, except that the regions are partitioned as follows: Starting with n points in the unit square, the square is split vertically to form two 1 by $\frac{1}{2}$ rectangles. These rectangles are then each split into two $\frac{1}{2}$ by $\frac{1}{2}$ squares. (As in the rectangle and triangle algorithms, we do this splitting only if the region has at least two points in it and is at or above the $\lceil \lg n \rceil$ th level of recursion). In general, each square is split vertically into two rectangles of ratio 2:1 between the vertical and horizontal sides, and each rectangle is split into two squares.

We do not have a tight upper bound on the cost of the matching produced by this algorithm, but a very crude upper bound can be derived by assuming that each region (whether square or rectangular) matches two points in its diagonally opposite corners. Thus

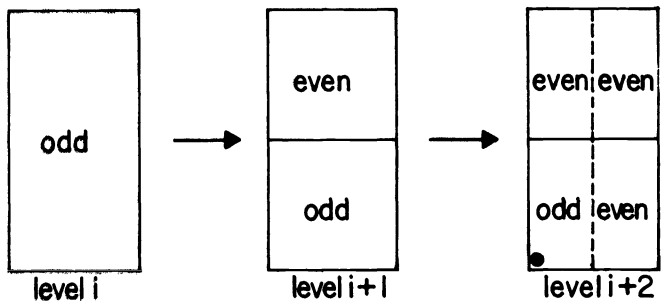
$$\begin{aligned}\text{cost} &\leqq \sum_{\substack{i \text{ even} \\ 0\leqq i\leqq \lceil \lg n\rceil + 1}} 2^i \frac{\sqrt{2}}{\sqrt{2}^i} + \sum_{\substack{i \text{ odd} \\ 0\leqq i\leqq \lceil \lg n\rceil + 1}} 2^i \frac{\sqrt{5}}{\sqrt{2}^{i+1}} \\ &\leqq (\sqrt{20}+\sqrt{8})\sqrt{n} + O(1) \approx 7.30\sqrt{n} + O(1).\end{aligned}$$

Certainly the least upper bound is much lower than this; we merely wanted to show the cost to be $O(\sqrt{n})$. We now construct an example for which the cost is asymptotic to $\frac{3}{2}\sqrt{n}$.

Let P be a set of points in the unit square such that each even square splits into two even rectangles, and each even rectangle R splits into odd squares S_1 and S_2 such that S_1 and S_2 leave unmatched points in opposite corners of R. (In analogy to the previous sections, a region is defined as *even* if it contains an even number of points of P, otherwise it is *odd*.) Assume that P is "full" to some level $2r+1$ in the sense that each level $2(r-1)+1$ rectangle has exactly 1 or 2 points of P in it. We can so construct P using the technique (described in the Appendix) used to construct "full" sets for the rectangle algorithm. Thus, if R is an even rectangle of level i, for some i such that $1 \leqq i \leqq 2(r-1)+1$, then at level $i+2$, R consists of two even and two odd



(a) *R even.*



(b) *R odd.*

FIG. 20. *Construction of a costly example for the square-rectangle algorithm.*

rectangles, as shown in Fig. 20(a). If R is odd, then at level $i+2$ R consists of three even and one odd rectangle as shown in Fig. 20(b). For all i, $0 \leqq i \leqq r-1$, let

$$E_i = \text{the number of even rectangles of level } 2i+1,$$

and let

$$O_i = \text{the number of odd rectangles of level } 2i+1.$$

(Note that a level k consists of rectangles rather than squares if and only if k is odd.) By the above remarks,

$$E_0 = 2, \qquad O_0 = 0,$$

and for $1 \leqq i \leqq r-1$

$$E_i = 2E_{i-1} + 30_{i-1}, \qquad O_i = 2E_{i-1} + O_{i-1}.$$

Observing that $E_i + O_i = 2 \times 4^i$, we find that

$$E_i = \tfrac{6}{5}4^i + \tfrac{4}{5}(-1)^i, \qquad O_i = \tfrac{4}{5}4^i - \tfrac{4}{5}(-1)^i.$$

Let $n = |P|$. Then

$$n = \sum_{i=0}^{r-1} 2E_i = \tfrac{4}{5}4^r + \tfrac{4}{5}(-1)^{r-1},$$

and hence

$$r = \log_4 (\tfrac{5}{4}n) + O(1).$$

Since the length of a level $2i+1$ diagonal is $\sqrt{5}/2^{i+1}$, we have

$$\text{cost}\,(P) = \sum_{i=0}^{r} E_i \frac{\sqrt{5}}{2^{i+1}} \geqq \frac{3}{2}\sqrt{n} - O(1).$$

We conjecture that the asymptotic worst-case cost for this algorithm is very close to $\frac{3}{2}\sqrt{n}$.

5. The four-square algorithm. The last divide and conquer algorithm we consider, the *four-square algorithm*, works as follows, given input points in the unit square. Each square S (initially the unit square) that has at least two input points in it, is split into four equal-sized subsquares. The algorithm is applied recursively to each of these subsquares. Then the best matching of the (at most four) unmatched points is made, the best matching of three points being the closest pair. In analogy to the above algorithms, if a square S contains at least two points and is on the $(\lceil \log_4 n \rceil + 1)$st level of recursion, then we arbitrarily match up the points in S until at most one is left. Thus this algorithm also runs in time $O(n \log n)$, or $O(n)$ if the floor function is available at unit cost.

As with the square-rectangle heuristic, we have no tight upper bound for this algorithm, but we know it to be $\Theta(\sqrt{n})$. For a lower bound, we now give an example of cost

$$\frac{\sqrt{2}}{\sqrt{3}}\left(1 + \frac{1}{\sqrt{2}}\right)\sqrt{n} - O(1) \approx 1.394\sqrt{n} - O(1).$$

Construct a set P of points in the unit square such that each even square S splits into S_1, S_2, S_3, S_4 such that S_1, S_3 are odd and S_2, S_4 are even, and S_1 and S_3 leave unmatched points in opposite corners of S; see Fig. 21(a). Also, as shown in Fig. 21(b), each odd square S splits into odd squares S_1, S_2, S_3 and an even square S_4 such that each of the three points left unmatched in S_1, S_2, S_3 is in a different corner of S. Thus at level i, each even square contributes an edge of length $1/\sqrt{2}^{i-1}$, and each odd square contributes an edge of length $1/2^i$. Make P such that for some integer r, each level $r-1$ square has either 1 or 2 points of P in it. For $0 \leqq i \leqq r-1$ let

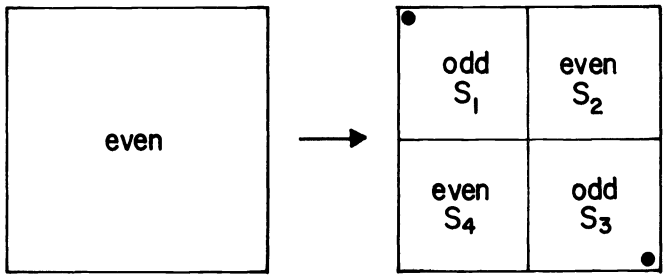
$$E_i = \text{the number of even level } i \text{ squares},$$

and let

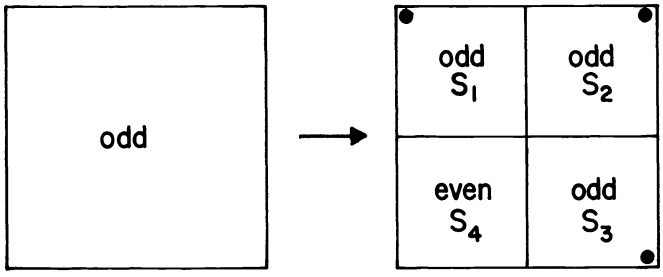
$$O_i = \text{the number of odd level } i \text{ squares}.$$

Then $E_0 = 1$, $O_0 = 0$, and for $1 \leqq i \leqq r-1$

$$E_i = 2E_{i-1} + O_{i-1}, \qquad O_i = 2E_{i-1} + 3O_{i-1}.$$



(a) *S even.*



(b) *S odd.*

FIG. 21. *Construction of a costly example for the four-square algorithm.*

Observing that $E_i + O_i = 4^i$, we find the solution

$$E_i = \tfrac{1}{3}4^i + \tfrac{2}{3}, \qquad O_i = \tfrac{2}{3}4^i - \tfrac{2}{3}.$$

Let $n = |P|$. Note that $n = O_{r-1} + 2E_{r-1}$, since each level $r-2$ square has 5 or 6 points of P, each level $r-1$ square has 1 or 2 points, and each level r square has 0 or 1 points. Thus $n = \frac{1}{3}4^r - \frac{2}{3}$, and hence $r = \log_4(3n+2)$. Therefore

$$\text{cost}(P) = \sum_{i=0}^{r-1} E_i + \sum_{i=0}^{r-2} O_i/2^i$$

$$= \frac{\sqrt{2}}{\sqrt{3}}\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{n} + \sqrt{2} - 2 + O\left(\frac{1}{\sqrt{n}}\right) \approx 1.394\sqrt{n} - O(1).$$

Incidentally, neglecting $O(1/\sqrt{n})$ terms, this expression is exactly the same as the upper bound for the cost of the triangle algorithm on n points in a main triangle. We have no geometric explanation for this coincidence.

6. Summary. The table summarizes the results (neglecting lower order terms) of this paper. From the table it can be seen that, of the four partitioning strategies we have examined, either the rectangle or the four-square gives the best worst-case performance. We leave as an open question whether there exist any other simple shapes that lead to better divide and conquer heuristics.

TABLE 1

Summary of results for matching in the unit square, neglecting lower order terms. The order given for the running time assumes that the floor function is available at unit cost. If it were not, then the times for the rectangle, triangle, square-rectangle, and four-square algorithms would be $\Theta(n \log n)$.

Algorithm	Order of running time	Worst known example cost	Upper bound on worst-case performance
Optimizing [5], [15]	n^3	$0.537\sqrt{n}$	$0.707\sqrt{n}$
Greedy [2], [4]	$n^{1.5} \log n$	$0.806\sqrt{n}$	$1.074\sqrt{n}$
Strip [15]	$n \log n$	$0.707\sqrt{n}$	$0.707\sqrt{n}$
Rectangle	n	$1.436\sqrt{n}$	$1.436\sqrt{n}$
Triangle	n	$1.971\sqrt{n}$	$1.971\sqrt{n}$
Square-Rectangle	n	$1.500\sqrt{n}$	?
Four-Square	n	$1.394\sqrt{n}$	?

7. Appendix. Bounds on C_n. Recall that

$$C_0 = C_1 = 0, \quad C_2 = \sqrt{3}, \quad C_3 = \sqrt{3}/\sqrt{2},$$

for all $n \geqq 1$,

$$C_{4n} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n-1}) + \sqrt{3}, \qquad C_{4n+1} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n}),$$

and for all $n \geqq 0$

$$C_{4n+2} = \frac{1}{\sqrt{2}}(C_{2n+1} + C_{2n+1}) + \sqrt{3}, \qquad C_{4n+3} = \frac{1}{\sqrt{2}}(C_{2n+2} + C_{2n+1}).$$

We get rather tight bounds on C_n by defining a special class of n and solving the recurrence for those values to within an $O(1/\sqrt{n})$ term.

Given an integer $r \geqq 0$, we say that a set of points P is *full to level r* if

1. P is balanced, and
2. For all rectangles R
   (i) if level $(R) \leqq r-1$ then $|R(P)| > 0$, and
   (ii) if level $(R) \geqq r$ then $|R(P)| \leqq 1$.

This definition implies that every level r rectangle has 0 or 1 points of P in it, and that every level $r-1$ rectangle has 1 or 2 points of P in it.

We say that an integer n is *full to level r* if there exists a set P such that $|P| = n$ and P is full to level r. We now show by induction on r that for each $r \geqq 0$ there is a pair $(n, n+1)$ for which both n and $n+1$ are full to level r. Clearly, 0 and 1 are both full to level 0. Let $r \geqq 0$ and assume that n and $n+1$ are both full to level r. Then there exist sets P_n, P_{n+1} such that $|P_n| = n$, $|P_{n+1}| = n+1$, and both P_n and P_{n+1} are full to level r.

Case 1. n is even. Let P_{2n+1} be the set consisting of P_n in its left subrectangle, and P_{n+1} in its right subrectangle (as shown in Fig. 22). Let P_{2n+2} be the set consisting of P_{n+1} as its left subrectangle and P_{n+1} as its right subrectangle. Both P_{2n+1} and P_{2n+2} are full to level $r+1$.

Case 2. n is odd. Let P_{2n} be the set with subrectangles consisting of P_n and P_n. Let P_{2n+1} be the set with subrectangles consisting of P_n and P_{n+1}. Both P_{2n} and P_{2n+1} are full to level $r+1$.

Thus 0, 1 are full to level 0. Also, if $t, t+1$ are full to level r, then t even implies $2t+1, 2t+2$ are full to level $r+1$, and t odd implies $2t, 2t+1$ are full to level $r+1$. The sequence $(0, 1,\ 1, 2,\ 2, 3,\ 5, 6,\ 10, 11,\ 21, 22,\ \cdots)$ thus consists of numbers full to some level. In fact, it is easily proved by induction that this sequence contains all numbers full to some level. Call the members of this sequence the *full numbers*. Incidentally, it is also easy to show that if P is a balanced set of points, then P is full to some level if and only if for each rectangle R such that $|R(P)|>0$, 4 does not divide $|R(P)|$.

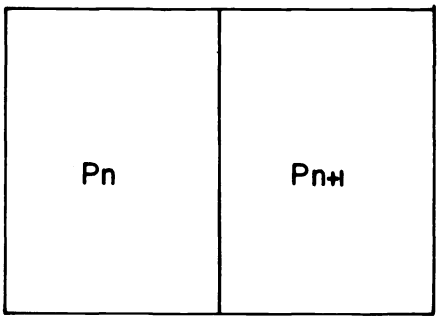


FIG. 22. *Constructing the set P_{2n+1} in Case 1 of the Appendix.*

Now let $r \geqq 0$ and let P be a set full to level r, such that $|P| = n$ is even. We must relate n and r. For all $i \geqq 0$, define

$$E_i = |\{\text{rectangle } R\text{: level } (R) = i \text{ and } |R(P)| \geqq 2 \text{ is even}\}|.$$

Similarly, define

$$O_i = |\{\text{rectangle } R\text{: level } (R) = i \text{ and } |R(P)| \text{ is odd}\}|.$$

Since n is even, we have that $E_0 = 1$, $O_0 = 0$. Since P is balanced we have that each nonempty even rectangle splits odd-odd, and (of course) each odd rectangle splits odd-even. Thus, for $1 \leqq i \leqq r-1$,

$$O_i = O_{i-1} + 2E_{i-1}, \qquad E_i = O_{i-1}.$$

Note that $O_i + E_i = 2^i$ because there are a total of 2^i level i rectangles; thus for $1 \leqq i \leqq r-1$ we have

$$O_i = \tfrac{2}{3}2^i - \tfrac{2}{3}(-1)^i, \qquad E_i = \tfrac{2}{3}2^{i-1} - \tfrac{2}{3}(-1)^{i-1}.$$

(Because P is full to level r, we have $E_i = 0$ for all $i \geqq r$.) Since P is balanced, we can associate with each even, nonempty rectangle R a pair of points $p_1, p_2 \in P$ such that p_1 and p_2 are in opposite corners of R and are matched by the algorithm. These $n/2$ pairs form a partition of P. Therefore

$$n = \sum_{i=0}^{r-1} 2E_i = 2\sum_{i=0}^{r-1}\left[\frac{2}{3}2^{i-1} - \frac{2}{3}(-1)^{i-1}\right] = \frac{2^{r+1}}{3} + \frac{2}{3}(-1)^{r+1}.$$

Define, for all $r > 0$,

$$b_r = \frac{2^{r+1}}{3} + \frac{2}{3}(-1)^{r+1}.$$

Then, as just shown, the sequence $(b_0, b_1, b_2, \cdots) = (0, 2, 2, 6, 10, 22, 42, \cdots)$ consists of all even full numbers. Define, for all $r \geqq 0$, $w_r = \lfloor 2^{r+1}/3 \rfloor$. The sequence $(w_0, w_1, w_2, \cdots) = (0, 1, 2, 5, 10, 21, \cdots)$ arises in connection with merge insertion [8, p. 187], and with an algorithm for finding the greatest common divisor of two integers [7, Exer. 4.5.2.–27]. Knuth points out that it is curious that this sequence arises in such different settings. We now add to this curiosity by observing that

$$w_r = \begin{cases} \dfrac{2^{r+1}}{3} - \dfrac{2}{3} = b_r & \text{if } r \text{ is even,} \\ \dfrac{2^{r+1}}{3} - \dfrac{1}{3} = b_r - 1 & \text{if } r \text{ is odd.} \end{cases}$$

Thus, w_r is the smaller of the two numbers full to level r.

Fix some $r \geqq 0$, and some set P full to level r such that $|P| = n$ is even (i.e., $n = b_r$). We now analyze rcost (P), that is, C_{b_r}.

$$\begin{aligned} \text{rcost}(P) &= \sum_{i=0}^{r-1} E_i \text{ (length of a level } i \text{ diagonal)} \\ &= \sum_{i=0}^{r-1} \left[\frac{2}{3}2^{i-1} - \frac{2}{3}(-1)^{i-1}\right] \frac{\sqrt{3}}{\sqrt{2}^i} \\ &= \frac{\sqrt{2}}{\sqrt{3}}\left(1 + \frac{1}{\sqrt{2}}\right)\sqrt{2}^r + \sqrt{3} - \sqrt{6} + \frac{\sqrt{2}}{\sqrt{3}}(2 - 2\sqrt{2})\left(-\frac{1}{\sqrt{2}}\right)^r. \end{aligned}$$

Now, $n = \frac{1}{3}2^{r+1} + \frac{2}{3}(-1)^{r+1}$, so that using Taylor series we have

$$r = \lg\left(\frac{3}{2}n\right) + O\left(\frac{1}{n}\right),$$

$$\sqrt{2}^r = \sqrt{2}^{\lg(3n/2) + O(1/n)} = \sqrt{3n/2}[1 + O(1/n)] = \sqrt{3n/2} + O(1/\sqrt{n}),$$

and

$$\left(-\frac{1}{\sqrt{2}}\right)^r = \left(-\frac{1}{\sqrt{2}}\right)^{\lg(3n/2) + O(1/n)} = O(1/\sqrt{n}).$$

Therefore,

$$C_n = \text{rcost}(P) = \left(1 + \frac{1}{\sqrt{2}}\right)\sqrt{n} + \sqrt{3} - \sqrt{6} + O\left(\frac{1}{\sqrt{n}}\right).$$

Thus we know (up to an $O(1/\sqrt{n})$ term) C_n for an infinite class of even n. Now we consider the other even values of n. Fix some $t \geqq 0$. We want to derive an upper bound on C_{2t}. For notational convenience, let $\alpha = 1/\sqrt{2}$, and let $D_n = C_n/\sqrt{3}$ for $n \geqq 0$. By induction on i it follows that for all $i \geqq 1$,

$$D_{i+1} - D_{i-1} = \alpha^{\lceil \lg(3i/4) \rceil}.$$

Let $2m$ be the largest integer such that $2m \leqq 2t$ and $2m = b_k$ for some $k \geqq 0$. We can write D_{2t} as

$$\begin{aligned} D_{2t} &= D_{2m} + \sum_{\substack{i \text{ odd,} \\ 2m+1 \leqq i \leqq 2t-1}} (D_{i+1} - D_{i-1}) \\ &= D_{2m} + \sum_{\substack{i \text{ odd,} \\ 2m+1 \leqq i \leqq 2t-1}} \alpha^{\lceil \lg(3i/4) \rceil}. \end{aligned}$$

Now [8, formulas (17), (18), p. 187] imply that for all $w_k < i \leqq w_{k+1}$, $\lceil \lg(3i/4) \rceil = k$. Therefore in particular, $\lceil \lg(3i/4) \rceil = k$ for all odd i such that $w_k \leqq 2m < 2m+1 \leqq i \leqq 2t-1 < 2t \leqq w_{k+1}$, so that

$$\sum_{\substack{i \text{ odd,} \\ 2m+1 \leqq i \leqq 2t-1}} \alpha^{\lceil \lg(3i/4) \rceil} = (t-m)\alpha^k.$$

We now express k in terms of m. Note that k is even if and only if w_k is even. Thus if k is even then $w_k = 2m = \frac{1}{3}2^{k+1} - \frac{2}{3}$ and hence $k = \lg(3m+1)$. If k is odd then $w_k = 2m-1 = \frac{1}{3}2^{k+1} - \frac{1}{3}$ and hence $k = \lg(3m-1)$. Thus,

$$\begin{aligned} D_{2t} = D_{2m} + (t-m)\alpha^k &= D_{2m} + (t-m)\left(\frac{1}{\sqrt{2}}\right)^k \\ &\leqq D_{2m} + (t-m)\left(\frac{1}{\sqrt{2}}\right)^{\lg(3m-1)} \\ &= \frac{1}{\sqrt{3}} C_{2m} + \frac{t-m}{\sqrt{3m-1}} \\ &= \frac{1}{\sqrt{3}}\left[\left(1+\frac{1}{\sqrt{2}}\right)\sqrt{2m} + \sqrt{3} - \sqrt{6} + O\left(\frac{1}{\sqrt{m}}\right)\right] + \frac{t-m}{\sqrt{3m-1}} \\ &= \frac{1}{\sqrt{3}}(\sqrt{2}+1)\sqrt{m} + 1 - \sqrt{2} + O\left(\frac{1}{\sqrt{m}}\right) + \frac{t-m}{\sqrt{3m-1}}. \end{aligned}$$

LEMMA 4.

$$\frac{1}{\sqrt{3}}(\sqrt{2}+1)\sqrt{m} + 1 - \sqrt{2} + O\left(\frac{1}{\sqrt{m}}\right) + \frac{t-m}{\sqrt{3m-1}} \leqq \frac{1}{\sqrt{3}}(\sqrt{2}+1)\sqrt{t} + 1 - \sqrt{2} + O\left(\frac{1}{\sqrt{t}}\right).$$

Proof. Let $d = (\sqrt{2}+1)/\sqrt{3}$. Since either $4m-2$ or $4m+2$ is a full number, we have $2t \leqq 4m+2$. However, if $2t = 4m+2$, then $2t$ would be a full number, implying that $2t = 2m$, contradicting the fact that $m \geqq 1$. Therefore $m \leqq t \leqq 2m$, and hence we have that $O(1/\sqrt{m}) = O(1/\sqrt{t})$. We need show only that

$$d\sqrt{m} + \frac{t-m}{\sqrt{3m-1}} \leqq d\sqrt{t},$$

i.e., that

$$d\sqrt{t} - d\sqrt{m} - \frac{t-m}{\sqrt{3m-1}} \geqq 0.$$

Define the function $f: [m, 2m] \to \mathbb{R}$ by

$$f(y) = d\sqrt{y} - d\sqrt{m} - \frac{y-m}{\sqrt{3m-1}}.$$

Differentiation shows that f has no local minima in the range $[m, 2m]$, so its absolute minimum on that range occurs at one of the endpoints. But $f(m) = 0$ and $f(2m) > 0$, so that $f(y) \geqq 0$ for $m \leqq y \leqq 2m$. □

By Lemma 4,

$$D_{2t} \leqq \frac{1}{\sqrt{3}}(\sqrt{2}+1)\sqrt{t}+1-\sqrt{2}+O\left(\frac{1}{\sqrt{t}}\right).$$

Therefore,

$$C_{2t} \leqq \left(1+\frac{1}{\sqrt{2}}\right)\sqrt{2t}+\sqrt{3}-\sqrt{6}+O\left(\frac{1}{\sqrt{n}}\right).$$

As shown above, this bound is achievable [neglecting $o(1)$ terms] when $2t = b_k$ for some $k \geqq 0$. An argument similar to the above, using $k = \lg(3m+1)$ instead of $\lg(3m-1)$, shows that

$$C_{2t} \geqq \left(\frac{1}{2\sqrt{2}}-\frac{3}{2}+2\sqrt{2}\right)\sqrt{2t}+\sqrt{3}-\sqrt{6}-o(1).$$

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. AVIS, *Worst case bounds for the Euclidean matching problem*, Internat. J. Comput. Math. Appl., 7 (1981), pp. 251–257.

[3] ———, Personal communication.

[4] J. L. BENTLEY AND J. B. SAXE, *Decomposable searching problems* 1: *Static-to-dynamic transformation*, J. Algorithms, 1 (1980), pp. 301–358.

[5] H. GABOW, *An efficient implementation of Edmond's algorithm for maximum matching on graphs*, J. Assoc. Comput. Mach., 23 (1976), pp. 221–234.

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

[7] D. E. KNUTH, *The Art of Computer Programming, Vol.* 2: *Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 2nd edition, 1981.

[8] ———, *The Art of Computer Programming, Vol.* 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[9] C. H. PAPADIMITRIOU, *The probabilistic analysis of matching heuristics*, in Proc. Fifteenth Annual Allerton Conf. on Communication, Control and Computing, 1977, pp. 368–378.

[10] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[11] E. M. REINGOLD AND K. J. SUPOWIT, *Probabilistic analysis of divide and conquer heuristics for minimum weighted Euclidean matching*, Networks, to appear.

[12] E. M. REINGOLD AND R. E. TARJAN, *On a greedy heuristic for complete matching*, this Journal, 10 (1981), pp. 676–681.

[13] M. I. SHAMOS, *Computational geometry*, Doctoral thesis, Dept. Computer Science, Yale Univ., New Haven, CT, 1978.

[14] K. J. SUPOWIT, D. A. PLAISTED AND E. M. REINGOLD, *Heuristics for weighted perfect matching*, in Proc. Twelfth Annual ACM Symposium on Theory of Computing, 1980, pp. 398–419.

[15] K. J. SUPOWIT, E. M. REINGOLD AND D. A. PLAISTED, *The traveling salesman problem and minimum matching in the unit square*, this Journal, this issue, pp. 144–156.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0009 $01.25/0

THE TRAVELING SALESMAN PROBLEM AND MINIMUM MATCHING IN THE UNIT SQUARE*

KENNETH J. SUPOWIT,† EDWARD M. REINGOLD‡ AND DAVID A. PLAISTED‡

Abstract. We show that the cost (length) of the shortest traveling salesman tour through n points in the unit square is, in the worst case, $\alpha_{\text{opt}}^{\text{tsp}}\sqrt{n}+o(\sqrt{n})$, where $1.075 \leqq \alpha_{\text{opt}}^{\text{tsp}} \leqq 1.414$. The cost of the minimum matching of n points in the unit square is shown to be, in the worst case, $\alpha_{\text{opt}}^{\text{mat}}\sqrt{n}+o(\sqrt{n})$, where $0.537 \leqq \alpha_{\text{opt}}^{\text{mat}} \leqq 0.707$. Furthermore, for each of these two problems there is an almost linear time heuristic algorithm whose worst case cost is, neglecting lower order terms, as low as possible.

Key words. traveling salesman problem, matching, analysis of algorithms, computational geometry, graph algorithms, heuristics

1. Introduction. Let P be a set of n points in the (Euclidean) unit square. Define a *traveling salesman tour* T of P as a set of n edges such that each point of P is an endpoint of exactly two edges, and the resulting graph (P, T) is connected. If n is even, then define a *matching* M of P as a set of $n/2$ edges such that each point of P is an endpoint of exactly one edge of M. If S is a tour or a matching then let cost(S) denote the sum of the lengths of the edges of S. The (*Euclidean*) *traveling salesman* (respectively, *matching*) *problem* is to find a minimum cost tour (respectively, matching).

The Euclidean traveling salesman problem is known to be NP-hard [7], [11] while the fastest known algorithm for Euclidean matching runs in time $\Theta(n^3)$ [6], [13]. This paper concerns fast heuristic algorithms for these two problems. Applications for heuristic Euclidean matching are described in [15].

In order to evaluate a heuristic, we use the following measure: the *worst-case performance* of a traveling salesman heuristic A is a function $f_A^{\text{tsp}}: \mathbb{N} \to \mathbb{R}$ such that

$$f_A^{\text{tsp}}(n) = \sup_P \{\text{the cost of } A\text{'s tour of } P\},$$

where P ranges over all sets of n points in the unit square. By "sup" we mean the supremum, i.e., the least upper bound; by "inf" we mean the infimum, the greatest lower bound. We use the supremum in the definition of worst-case performance because it is possible (since there are infinitely many n-point sets) that there is no n-point set P for which the cost of A's tour is maximized. If B is a matching heuristic then the worst case performance of B is the function f_B^{mat} defined analogously. The first question that arises is how good the worst-case performance of any traveling salesman (respectively, matching) heuristic can be? Let $f_{\text{opt}}^{\text{tsp}}$ denote the worst-case performance of the exhaustive optimizing traveling salesman problem algorithm. Let $f_{\text{opt}}^{\text{mat}}$ denote the worst-case performance of the $\Theta(n^3)$ optimizing matching algorithm.

* Received by the editors September 1, 1980, and in revised form May 28, 1982. Preliminary versions of some of the results contained in this paper were presented at the Twelfth Annual ACM Symposium on Theory of Computing, April; 1980. This research was supported in part by the National Science Foundation under grants NSF MCS 77-22830 and NSF MCS 79-04897.

† Hewlett-Packard Laboratories, Computer Research Center, Palo Alto, California 94304. This research was conducted while this author was at the Department of Computer Science, University of Illinois at Urbana-Champaign.

‡ Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801.

We will show that both $f_{\text{opt}}^{\text{tsp}}$ and $f_{\text{opt}}^{\text{mat}}$ are $\Theta(\sqrt{n})$. Let

$$\alpha_A^{\text{tsp}} = \inf\{x : (\forall n \geqq 0)[f_A^{\text{tsp}}(n) \leqq x\sqrt{n} + o(\sqrt{n})]\},^1$$

and

$$\alpha_B^{\text{tsp}} = \inf\{x : (\forall n \geqq 0)[f_B^{\text{tsp}}(n) \leqq x\sqrt{n} + o(\sqrt{n})]\},$$

adopting the convention that the infimum of the empty set is infinity. The statement that $f_{\text{opt}}^{\text{tsp}}$ and $f_{\text{opt}}^{\text{mat}}$ are $\Theta(\sqrt{n})$ may be rephrased as $\alpha_{\text{opt}}^{\text{tsp}}$ and $\alpha_{\text{opt}}^{\text{mat}}$ are both finite and nonzero. Thus, the answer to the question of how good the worst-case performance of a heuristic can possibly be is $\alpha_{\text{opt}}^{\text{tsp}}\sqrt{n} + o(\sqrt{n})$ for the traveling salesman problem and $\alpha_{\text{opt}}^{\text{mat}}\sqrt{n} + o(\sqrt{n})$ for the matching problem.

There are two main results of this paper:

1. $1.075 \approx 2/\sqrt{\sqrt{12}} \leqq \alpha_{\text{opt}}^{\text{tsp}} \leqq \sqrt{2} \approx 1.414$, $0.537 \approx 1/\sqrt{\sqrt{12}} \leqq \alpha_{\text{opt}}^{\text{mat}} \leqq 1/\sqrt{2} \approx 0.707$.

2. There exists a heuristic algorithm A for the traveling salesman problem such that A runs in time $O(n \log n)$ and $\alpha_A^{\text{tsp}} = \alpha_{\text{opt}}^{\text{tsp}}$. Analogously, for matching there exists a heuristic algorithm B that runs in $O(n \log n)$ time and has $\alpha_B^{\text{mat}} = \alpha_B^{\text{tsp}}$.

Furthermore, if the floor function is available at unit cost, then for each unbounded, nonnegative, nondecreasing, integer-valued function f such that $f(n)$ is computable in time $O(nf(n))$, the expression "$O(n \log n)$" can be replaced by "$O(nf(n))$" in the statement of (2). Examples of such functions f are $\lceil \lg \lg n \rceil$, $\lg^* n$, $\alpha(n, n)$ [18], and so on. In other words, (2) says that for each of these two problems, there exists an almost linear time heuristic algorithm whose worst-case performance is asymptotically optimal.

The worst-case performance (as defined above) of various traveling salesman problem and matching algorithms is given in Tables 1 and 2, respectively. For matching, the rectangle algorithm is the best of the simple divide-and-conquer algorithms; its worst-case behavior is analyzed in [17] (this issue, pp. 118–143) and its average-case behavior is analyzed in [14]. The greedy algorithm for matching works by iteratively matching the two closest unmatched points; the analysis of its worst-case performance is in [1] and its $O(n^{1.5} \log n)$ implementation is in [4]. The spiral rack matching algorithm and its analysis are in [9].

Our results on worst-case performance should also be compared with the known results on *expected* performance:

(i) The expected cost of the shortest tour of n points drawn from a uniform distribution in the unit square is $\beta_{\text{tsp}}\sqrt{n} + o(\sqrt{n})$, for some β_{tsp} satisfying $0.61 \leqq \beta_{\text{tsp}} \leqq 0.92$ [2].

(ii) The expected cost of the minimum matching of n points drawn from a uniform distribution in the unit square is $\beta_{\text{mat}}\sqrt{n} + o(\sqrt{n})$, for some β_{mat} satisfying $0.25 \leqq \beta_{\text{mat}} \leqq 0.402$ [12].

2. Lower bounds on $\alpha_{\text{opt}}^{\text{tsp}}$ and $\alpha_{\text{opt}}^{\text{mat}}$. We will show that $2/\sqrt{\sqrt{12}} \leqq \alpha_{\text{opt}}^{\text{tsp}}$ and that $1/\sqrt{\sqrt{12}} \leqq \alpha_{\text{opt}}^{\text{mat}}$. Our strategy is to construct an infinite class of sets of points P such that any tour of P has cost at least $(2/\sqrt{\sqrt{12}})\sqrt{|P|}$ and any matching of P has cost at least $(1/\sqrt{\sqrt{12}})\sqrt{|P|}$. Let $k \geqq 2$ be an even integer. Let P be the set of points

$$\bigcup_{\substack{0 \leqq i \leqq k-1 \\ 0 \leqq j \leqq \lfloor 2/(\delta\sqrt{3}) \rfloor}} \left\{\left(\left[i + \frac{j \bmod 2}{2}\right]\delta, j\frac{\sqrt{3}}{2}\delta\right)\right\},$$

[1] When we say that $(\forall n)[f(n) \leqq x\sqrt{n} + o(\sqrt{n})]$, we mean that

$$(\exists g : \mathbb{N} \to \mathbb{R})[g(n) = o(\sqrt{n}) \text{ and } (\forall n)[f(n) \leqq x\sqrt{n} + g(n)]].$$

TABLE 1

Summary of results for the traveling salesman problem in the unit square, neglecting lower order terms. f is any unbounded, nonnegative, nondecreasing, integer-valued function computable in $O(nf(n))$ time. The order given for the running time assumes that the floor function is available at unit cost. If it were not, then the time for the decomposition algorithm would be $\Theta(n \log n)$.

Algorithm	Order of running time	Worst known example cost	Upper bound on worst-case performance
Optimizing	$n2^n$	$1.075\sqrt{n}$	$\alpha_{\text{opt}}^{\text{tsp}}\sqrt{n}$
Strip	$n \log n$	$1.414\sqrt{n}$	$1.414\sqrt{n}$
Decomposition	$nf(n)$	$1.075\sqrt{n}$	$\alpha_{\text{opt}}^{\text{tsp}}\sqrt{n}$

TABLE 2

Summary of results for matching in the unit square, neglecting lower order terms. f is any unbounded, nonnegative, nondecreasing, integer-valued function computable in $O(nf(n))$ time. The order given for the running time assumes that the floor function is available at unit cost. If it were not, then the times for the rectangle, spiral rack, and decomposition algorithms would be $\Theta(n \log n)$.

Algorithm	Order of running time	Worst known example cost	Upper bound on worst-case performance
Optimizing [6], [13]	n^3	$0.537\sqrt{n}$	$\alpha_{\text{opt}}^{\text{mat}}\sqrt{n}$
Greedy [1], [4]	$n^{1.5} \log n$	$0.806\sqrt{n}$	$1.075\sqrt{n}$
Strip	$n \log n$	$0.707\sqrt{n}$	$0.707\sqrt{n}$
Rectangle [17]	n	$1.436\sqrt{n}$	$1.436\sqrt{n}$
Spiral rack [9]	n	$1.014\sqrt{n}$	$1.014\sqrt{n}$
Decomposition	$nf(n)$	$0.537\sqrt{n}$	$\alpha_{\text{opt}}^{\text{mat}}\sqrt{n}$

where $\delta = 1/(k-1/2)$ is a factor introduced so that the points of P all lie in the unit square. An example is shown in Fig. 1 with $k = 6$. The points of P are vertices of a hexagonal grid, which, incidentally, also gives the densest packing of the plane by unit circles [16] and the worst known example for the greedy matching heuristic [1].

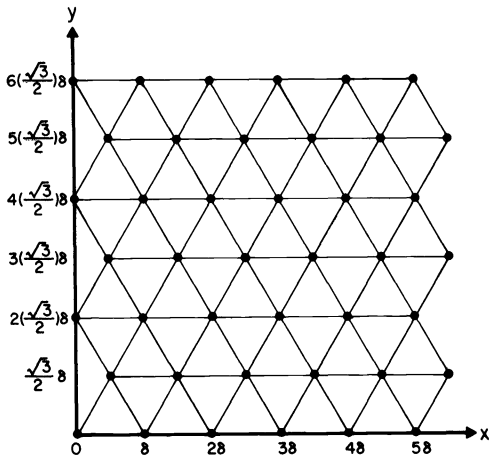


FIG. 1. *The vertices of a hexagonal grid.*

Let $n = |P|$ and let T be any tour of P. Since δ is the distance between the closest pair of points in P, each edge of T has length at least δ, so that

$$\text{cost}(T) \geqq n\delta.$$

Now

$$n = (\text{number of rows of } P) \times (\text{number of points per row})$$
$$= \left(\left\lfloor \frac{2}{\delta\sqrt{3}} \right\rfloor + 1\right) \times k > \frac{2k}{\delta\sqrt{3}}.$$

Therefore

$$\sqrt{n} > \frac{\sqrt{2k}}{\sqrt{\delta}\sqrt{\sqrt{3}}},$$

giving

$$\text{cost}(T) \geqq n\delta > \frac{\sqrt{2k}}{\sqrt{\delta}\sqrt{3}}\sqrt{n\delta} = \frac{2}{\sqrt{\sqrt{12}}}\sqrt{\delta k}\sqrt{n} > \frac{2}{\sqrt{\sqrt{12}}}\sqrt{n}.$$

Similarly, if M is a matching of P then M has $n/2$ edges, each of length at least δ, so that

$$\text{cost}(M) \geqq \frac{n}{2}\delta > \frac{1}{\sqrt{\sqrt{12}}}\sqrt{n}.$$

3. Upper bounds on $\alpha_{\text{opt}}^{\text{tsp}}$ and $\alpha_{\text{opt}}^{\text{mat}}$. We present a heuristic for the traveling salesman problem that we call the *strip algorithm*, and show that its worst-case performance is at most $\sqrt{2n} + O(1)$. The algorithm can be used for matching, when n is even, by taking the shorter of the two matchings contained in the tour found. Therefore the worst-case performance of the strip algorithm for matching is bounded above by $\sqrt{n/2} + O(1)$. This will show that $\alpha_{\text{opt}}^{\text{tsp}} \leqq \sqrt{2}$ and that $\alpha_{\text{opt}}^{\text{mat}} \leqslant 1/\sqrt{2}$.

The strip algorithm for the traveling salesman problem is a modification of one analyzed for its expected performance in [2]. We are given a set of n points in the unit square. Let $r = \lceil \sqrt{n/2} \rceil$. Divide the unit square into r vertical strips, each of width $1/r$. Construct a tour T_1 of the points by starting at the lowest point in the leftmost strip, going up that strip from point to point, over to the top point of the next strip, then down that strip point by point, up the next, and so on, finally returning to the starting point, as shown by the jagged line in Fig. 2. For simplicity, not all of the input points are pictured; in order to actually have 5 strips there would have to be between 50 and 71 points.

A second tour T_2 is constructed in the same way, except that now the strip boundaries are shifted by $1/(2r)$ to the right. There are $r+1$ strips used in constructing T_2, each of width $1/r$. In Fig. 3, the strip boundaries for T_1 are shown as solid lines, those for T_2 as dashed lines. Note that the leftmost of these strips contains none of the points in its left half. Similarly, the rightmost strip contains none of the points in its right half.

The strip algorithm outputs the shorter of the two tours T_1 and T_2. The algorithm can be implemented in time $O(n \log n)$ by appropriately sorting the points.

To derive an upper bound on the cost of the tour produced, we will bound the sum of the horizontal and vertical components, and then use the triangle inequality. Consider paths P_1 and P_2 defined as follows: P_1 starts at the bottom, on the median of the leftmost of the strips used in constructing T_1. P_1 follows the median of that

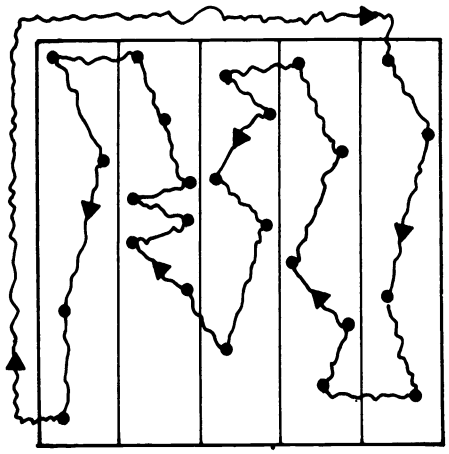


FIG. 2. *The construction of a tour, using strips.*

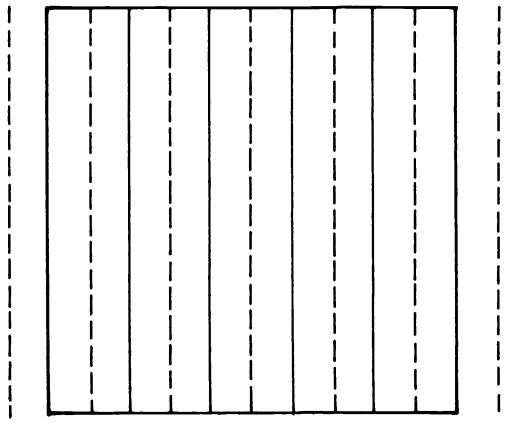


FIG. 3. *The two sets of strip boundaries.*

strip up to the top, then down the median of the next strip, up the median of the next, and so on. For each strip, for each point in that strip, the path P_1 juts out to that point and then back to the median, moving at right angles, as illustrated in Fig. 4 by the jagged line. The path P_2 is defined like P_1, except that P_2 follows the medians of the strips used to construct T_2. It follows from the triangle inequality that length $(T_1) \leqq$ length (P_1), and that length $(T_2) \leqq$ length (P_2). We now derive an upper bound on length (P_1) + length (P_2).

Consider some input point q; q must lie in some strip used for T_1 and for P_1 (shown in Fig. 5 between solid lines), and in some strip used for T_2 and for P_2 (shown in Fig. 5 between dashed lines). In Fig. 5, a segment of P_1 is shown as a bold line, and a segment of P_2 as a jagged line. It is clear that the total amount of horizontal line in P_1 or P_2 jutting out to q and back is $2 \times 1/(2r) = 1/r$. Since q was arbitrary, there is a total of n/r units of horizontal line in P_1 and P_2 together that juts out to points and back. Also, P_1 has r units of vertical line (that is, r strips of unit length). P_2 has $r+1$ strips and hence $r+1$ units of vertical line. P_1 has $1-(1/r)$ units of horizontal line that run from the end of one strip to the start of the next and P_2 has

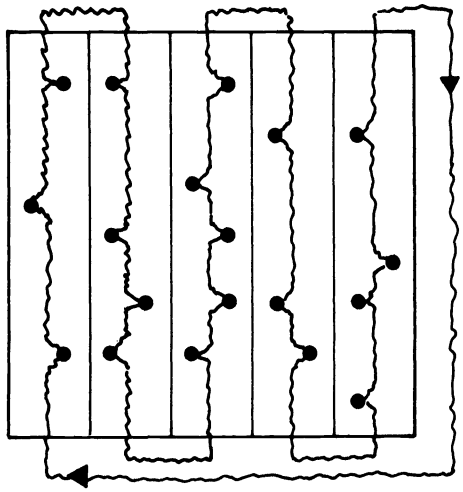


FIG. 4. *The construction of the path P_1, using strips.*

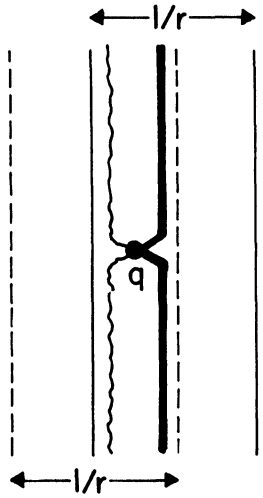


FIG. 5. *The paths P_1 and P_2 at a point q.*

1 unit of such line. Finally, P_1 and P_2 each have a segment of length at most $\sqrt{2}$ that joins the end of the last strip back to the starting position. Thus

$$\begin{aligned}\text{length}\,(T_1)+\text{length}\,(T_2) &\leqq \text{length}\,(P_1)+\text{length}\,(P_2)\\ &\leqq \frac{n}{r}+r+(r+1)+\left(1-\frac{1}{r}\right)+1+\sqrt{2}+\sqrt{2}\\ &= \frac{n}{r}+2r+O(1)\\ &= \frac{n}{\lceil\sqrt{n/2}\rceil}+2\lceil\sqrt{n/2}\rceil+O(1)\\ &= 2\sqrt{2n}+O(1).\end{aligned}$$

Therefore

$$\min\{\text{length}(T_1), \text{length}(T_2)\} \leqq \sqrt{2n} + O(1),$$

and if n is even, the cheaper of the two matchings contained in the shorter of $\{T_1, T_2\}$ has cost at most $\sqrt{n/2} + O(1)$.

These bounds are asymptotically achievable, as is suggested by the example pictured in Fig. 6. T_1 is shown as a jagged line; T_2 is not shown, but looks like T_1 shifted by $1/(2r)$ to the right. The points, which number $n = 2k^2$ for some even integer k, are arranged so that halfway between each solid vertical line and either of its two neighboring dashed vertical lines there is a vertical string of n/r points; these points are ε/r apart, for some $\varepsilon < 1/(2r)$. Intuitively, these points are placed so that T_1 and T_2 must zigzag, and hence look very much like P_1 and P_2, respectively. This attains the maximum amount (neglecting $O(1)$ terms) of horizontal line for T_1 and for T_2. There is a point at the bottom of each strip, so as to attain the maximum vertical length. To compute $\min\{\text{length}(T_1), \text{length}(T_2)\}$, note that by the Pythagorean theorem, each short, almost horizontal edge of the tour has length

$$\sqrt{\left(\frac{\varepsilon}{2r}\right)^2 + \left(\frac{1}{2r}\right)^2} > \frac{1}{2r}.$$

There are $r((n/r)-1)$ of these edges. There are r long vertical pieces, each of length $1-\varepsilon$. Recalling that $r = \lceil\sqrt{n/2}\rceil$, we have

$$\min\{\text{length}(T_1), \text{length}(T_2)\} > r\left(\frac{n}{r} - 1\right)\frac{1}{2r} + r(1-\varepsilon)$$

$$= \frac{n}{2r} - \frac{1}{2} + r - r\varepsilon > \frac{n}{2r} + r - 1 = \sqrt{2n} - 1.$$

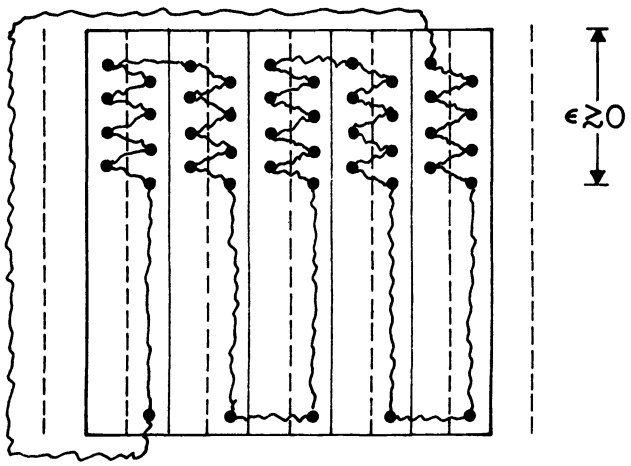


FIG. 6. *A set of points in the unit square for which length* $(T_1) \approx$ *length* $(T_2) \approx \sqrt{2n}$.

We can easily arrange the points in this example so that each of the matchings from T_1 or T_2 contains about half of the long vertical edges; hence the strip matching algorithm produces a matching for this example of cost at least $\sqrt{n/2} - \frac{1}{2}$.

4. The decomposition heuristic for the traveling salesman problem. In this section we present a decomposition heuristic for the traveling salesman problem that achieves (asymptotically) the best possible worst-case performance. The heuristic, given in Algorithm 1, is reminiscent of the traveling salesman problem heuristic given by Karp in [10]. Assuming a uniform distribution, Karp's heuristic runs in time $O(n \log n)$ almost everywhere, and, for all $\varepsilon > 0$, outputs a tour of cost at most $1+\varepsilon$ times that of the optimal tour, almost everywhere. Karp's heuristic requires exponential time on some input sets. Our heuristic, on the other hand, always runs in time $O(n \log n)$ and has the best possible worst-case performance, neglecting lower order terms. An argument similar to the one we give below proves that Karp's heuristic also has, asymptotically, the best possible worst-case performance.

In order to avoid the sorting required by the strip heuristic, Algorithm 1 uses a slightly crude approximation, the *modified strip heuristic*. It is essentially the serpentine algorithm of [9]. Each column of subsquares in the grid is a strip and we traverse the subsquares by going up the first strip, down the second, up the third, and so on. The tour thus constructed visits the points in some arbitrary order that is consistent with the cell order. Figure 7 shows an example of such a tour. The advantage of this heuristic is that it requires only $O(m)$ time for m points. It produces a tour of length $O(\sqrt{m})$ because an edge wholly contained in one of the subsquares has length at most $\sqrt{2}/c = O(1/\sqrt{m})$ (see Algorithm 1 for the definition of c).

ALGORITHM 1. The asymptotically optimal decomposition heuristic for the traveling salesman problem on a set P of n points in the unit square.

1. $c \leftarrow \lceil 2\sqrt{n}/\sqrt{\log_z f(n)} \rceil$, where $z > 2$ is some real, and $f(n)$ is a nonnegative, unbounded, nondecreasing, integer-valued function computable in $O(nf(n))$ time.
2. Divide the unit square into a regular grid of c^2 subsquares, each of side length $1/c$.
3. For each of the subsquares, do the following:
   $P' \leftarrow$ the subset of P inside the subsquare
   while $|P'| > 0$ **do**
   begin
   $k \leftarrow \min\{4\lceil n/c^2 \rceil, |P'|\}$
   $Q \leftarrow$ a set of k points chosen arbitrarily from P'
   Use dynamic programming to find the shortest traveling salesman tour of Q [3], [8].
   Distinguish one point of Q
   $P \leftarrow P' - Q$
   end
4. Perform the modified strip heuristic to find a tour of the distinguished points.
5. $T' \leftarrow$ the union of all tours found in Steps 3 and 4.
6. Convert T' to a tour T by the method of [5] (see [13])[2] and output T.

We first analyze the worst-case performance of Algorithm 1. Let α be a real number such that

$$(\forall n)[f_{\text{opt}}^{\text{tsp}}(n) \leqq \alpha\sqrt{n} + o(\sqrt{n})].$$

[2] Since T' is a union of tours, the degree of each vertex in T' is even so T' contains an Eulerian circuit. Start at an arbitrary vertex and follow the order of the Eulerian circuit, but skip any previously encountered point; the result is a Hamiltonian circuit. By the triangle inequality, the cost of this Hamiltonian circuit is no more than the cost of the Eulerian circuit we started with. The cost of the Eulerian circuit is the sum of the lengths of the edges in T'.

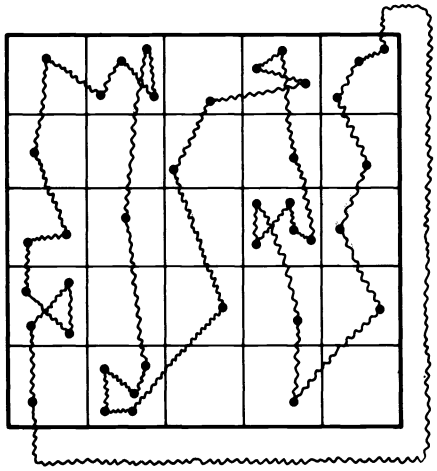


FIG. 7. *The modified strip heuristic.*

We will show that

$$(\forall n)[f^{\text{tsp}}_{\text{dec}}(n) \leqq \alpha\sqrt{n} + o(\sqrt{n})].$$

where "dec" denotes the decomposition algorithm; this will prove that $\alpha^{\text{tsp}}_{\text{dec}} = \alpha^{\text{tsp}}_{\text{opt}}$. In fact, it will prove that

$$\{x: (\forall n \geqq 0)[f^{\text{tsp}}_{\text{opt}}(n) \leqq x\sqrt{n} + o(\sqrt{n})]\} = \{x: (\forall n \geqq 0)[f^{\text{tsp}}_{\text{dec}}(n) \leqq x\sqrt{n} + o(\sqrt{n})]\},$$

which is not implied by the equality of the infima.

Fix some input set P of n points. For notational convenience, let

$$b = \lceil n/c^2 \rceil. \tag{1}$$

Number the subsquares from 1 to c^2. For all i, $1 \leqq i \leqq c^2$, let B_i denote the set of input points within the ith subsquare, and let $b_i = |B_i| \bmod 4b$. Thus the number of applications of the optimizing dynamic programming algorithm (that is, the number of executions of the body of the **while** loop) when working on subsquare B_i is at most

$$\frac{|B_i| - b_i}{4b} + 1.$$

Let

$$t = \sum_{i=1}^{c^2} \frac{|B_i| - b_i}{4b} = (n - \sum_{i=1}^{c^2} b_i)/(4b); \tag{2}$$

thus $t + c^2$ is the total number of executions of the body of the **while** loop.

Now for all $r \geqq 1$, the cost of the tour produced by the optimizing algorithm on r points in a $1/c$ by $1/c$ square is at most $[\alpha\sqrt{r} + o(\sqrt{r})]/c$. The factor $1/c$ scales down the cost from the unit square to the $(1/c) \times (1/c)$ square. Therefore the sum of the costs of all the tours produced by the optimizing algorithm is at most

$$\frac{1}{c}\left[t(\alpha\sqrt{4b} + o(\sqrt{b})) + \sum_{i=1}^{c^2} (\alpha\sqrt{b_i} + o(\sqrt{b_i}))\right] = \frac{\alpha}{c}\left[t\sqrt{4b} + \sum_{i=1}^{c^2} \sqrt{b_i} + o(c^2\sqrt{b})\right],$$

since $b_i \leqq 4b$ and $t \leqq c^2$.

The tour produced in Step 4 by the modified strip algorithm on at most c^2 points has cost $O(c)$. In Step 5, the tour T produced by the method of [5] (see [13]) from T', the union of the tours found in Steps 3 and 4, has cost at most $\sum_{e \in T'} \text{length}(e)$, by the triangle inequality. Therefore the total cost of the tour T produced by the algorithm is at most

$$\frac{\alpha}{c}\left[t\sqrt{4b} + \sum_{i=1}^{c^2} \sqrt{b_i} + o(c^2\sqrt{b})\right] + O(c). \tag{3}$$

Note that $\sqrt{b} = \sqrt{\lceil n/c^2 \rceil} < \sqrt{n}/c + 1$ and that $c = o(\sqrt{n})$ so (3) can be rewritten as

$$\frac{\alpha}{c}\left[t\sqrt{4b} + \sum_{i=1}^{c^2} \sqrt{b_i}\right] + o(\sqrt{n}). \tag{4}$$

Let $g : \mathbb{R}^{c^2} \to \mathbb{R}$ be defined by

$$g(b_1, b_2, \cdots, b_{c^2}) = t\sqrt{4b} + \sum_{i=1}^{c^2} \sqrt{b_i}$$

$$= \frac{1}{4b}\left(n - \sum_{i=1}^{c^2} b_i\right)\sqrt{4b} + \sum_{i=1}^{c^2} \sqrt{b_i} = \frac{1}{\sqrt{4b}}\left(n - \sum_{i=1}^{c^2} b_i\right) + \sum_{i=1}^{c^2} \sqrt{b_i}.$$

Taking partial derivatives shows that g is maximized at $b_1 = b_2 = \cdots = b_{c^2} = b$. In this case $n \geqq bc^2$, but because $b = \lceil n/c^2 \rceil$, we have $bc^2 \geqq n$ so that $n = bc^2$ giving

$$t = \left(n - \sum_{i=1}^{c^2} b_i\right)/(4b) = (n - bc^2)/(4b) = 0.$$

Therefore (4) is maximized when $t = 0$ and $b_1 = b_2 = \cdots = b_{c^2} = b$. Hence

$$\text{cost}(T) \leqq \frac{\alpha}{c} \sum_{i=1}^{c^2} \sqrt{b} + o(\sqrt{n}) = \alpha c\sqrt{b} + o(\sqrt{n}) = \alpha\sqrt{n} + o(\sqrt{n}),$$

since $\sqrt{b} < \sqrt{n}/c + 1$. Thus

$$f_{\text{dec}}^{\text{tsp}}(n) \leqq \alpha\sqrt{n} + o(\sqrt{n})$$

so that $\alpha_{\text{dec}}^{\text{tsp}} = \alpha_{\text{opt}}^{\text{tsp}}$ as claimed. Thus the decomposition algorithm has the asymptotically best possible worst-case performance.

We now analyze the running time of Algorithm 1. Under the real RAM model of computation, partitioning the n points into the c^2 subsquares can be done in $O(n \log n)$ time, since the subsquares form a grid. If we allow the floor function at unit cost, then this partitioning can be done in $O(n)$ time.

There is a dynamic programming algorithm that finds the shortest tour of r points in time $O(r2^r)$ [3], [8], hence in time $O(z^r)$ for $z > 2$. Step 3 makes at most $t + c^2$ calls on that algorithm, each with at most $4b$ points. Therefore the time required by Step 3 is

$$O((t + c^2)z^{4b}) = O\left(\left[\left(n - \sum_{i=1}^{c^2} b_i\right)\Big/(4b) + c^2\right]z^{4b}\right)$$

$$= O\left(\left(\frac{n}{b} + c^2\right)z^{4b}\right)$$

$$= O(c^2 z^{4\lceil n/(4n/\log_z f(n))\rceil}) = O\left(\frac{nf(n)}{\log f(n)}\right) = O(nf(n)).$$

There are at most

$$t+c^2<\frac{n}{4b}+c^2=O\left(\frac{n}{\log f(n)}\right)$$

points distinguished in Step 3. Therefore Step 4 can be performed in time $O(n/\log f(n))$.

Thus, the total running time of Algorithm 1 is $O(n \log n)$ under the real RAM model of computation. If the floor function is available at unit cost, the running time is $O(nf(n))$.

5. The decomposition heuristic for matching. In this section we present a decomposition heuristic for matching that, like Algorithm 1 for the traveling salesman problem, achieves (asymptotically) the best possible worst-case behavior. The heuristic, given in Algorithm 2, is almost an exact parallel to Algorithm 1, and its analysis is virtually identical. In particular, we can show, with the same argument as before, that if α is a real number such that

$$(\forall n)[f_{\text{opt}}^{\text{mat}}(n)\leqq\alpha\sqrt{n}+o(\sqrt{n})],$$

then

$$(\forall n)[f_{\text{dec}}^{\text{mat}}(n)\leqq\alpha\sqrt{n}+o(\sqrt{n})]$$

so that $\alpha_{\text{dec}}^{\text{mat}}=\alpha_{\text{opt}}^{\text{mat}}$.

ALGORITHM 2. The asymptotically optimal decomposition heuristic for matching.

1. $c\leftarrow\lceil\sqrt{n}/\sqrt{f(n)}\rceil$, where $f(n)$ is a nonnegative, unbounded, nondecreasing, integer-valued function computable in $O(nf(n))$ time.
2. Divide the unit square into a regular grid of c^2 subsquares, each of side length $1/c$.
3. For each of the subsquares, do the following:
   $P'\leftarrow$ the subset of P inside the subsquare
   if $|P'|$ is odd **then** distinguish an arbitrarily chosen point in P' and delete it from P'
   while $|P'|>0$ **do**
   begin
   $k\leftarrow$ the largest even integer less than or equal to $\min\{4\lceil n/c^2\rceil, |P'|\}$
   $Q\leftarrow$ a set of k points chosen arbitrarily from P'
   Use the optimizing matching algorithm [6], [13] to find the minimum cost matching of Q
   $P\leftarrow P'-Q$
   end
4. Perform the modified strip heuristic to find a tour of the distinguished points, then find the less costly of the two matchings contained in the tour.
5. Output the union of all matchings found in Steps 3 and 4.

As for the time required, the partitioning takes $O(n \log n)$ under the real RAM model of computation and $O(nf(n))$ if the floor function is available at unit cost: Let b and t be defined by (1) and (2), respectively; note that now $b=\Theta(\sqrt{f(n)})$. There are at most $t+c^2$ calls on the optimizing algorithm, each with at most $4b$ points and hence each requiring $O(b^3)$ time. Thus Step 3 requires time

$$O((t+c^2)b^3)=O\left(\left(\frac{n}{b}+c^2\right)b^3\right)=O(c^2b^3)=O(nf(n)).$$

There is at most one distinguished point in each subsquare, so Step 4 can be performed in time $O(c^2) = O(n/\sqrt{f(n)})$. The total time for Algorithm 2 is thus $O(n \log n)$ without the floor function and $O(nf(n))$ with it.

6. Open problems. Many questions remain unanswered. We know that

$$\frac{2}{\sqrt{\sqrt{12}}} \leqq 2\alpha_{\text{opt}}^{\text{mat}} \leqq \alpha_{\text{opt}}^{\text{tsp}} \leqq \sqrt{2}.$$

Does $\alpha_{\text{opt}}^{\text{tsp}} = 2\alpha_{\text{opt}}^{\text{mat}}$? We can define $\alpha_{\text{opt}}^{\text{mst}}$ for the minimum spanning tree problem in analogy to $\alpha_{\text{opt}}^{\text{tsp}}$ and $\alpha_{\text{opt}}^{\text{mat}}$; the hexagonal grid example of Fig. 1 establishes that $\alpha_{\text{opt}}^{\text{mst}} \geqq 2/\sqrt{\sqrt{12}}$. Furthermore, it is obvious that $\alpha_{\text{opt}}^{\text{mst}} \leqq \alpha_{\text{opt}}^{\text{tsp}}$. Does $\alpha_{\text{opt}}^{\text{mst}} = \alpha_{\text{opt}}^{\text{tsp}}$? How does $2\alpha_{\text{opt}}^{\text{mat}}$ compare with $\alpha_{\text{opt}}^{\text{mst}}$? We conjecture that

$$\alpha_{\text{opt}}^{\text{tsp}} = \alpha_{\text{opt}}^{\text{mst}} = 2\alpha_{\text{opt}}^{\text{mat}} = \frac{2}{\sqrt{\sqrt{12}}}.$$

Finally, our decomposition algorithms are not quite linear time; are there linear time algorithms A and B for the traveling salesman problem and matching, respectively, for which $\alpha_A^{\text{tsp}} = \alpha_{\text{opt}}^{\text{tsp}}$ and $\alpha_B^{\text{mat}} = \alpha_{\text{opt}}^{\text{mat}}$?

7. Acknowledgment. We gratefully acknowledge suggestions by the referee that helped sharpen one of the time bounds, improve the notation, and clarify the exposition.

REFERENCES

[1] D. AVIS, *Worst case bounds for the Euclidean matching problem*, Internat. J. Comput. Math. Appl., 7 (1981), pp. 251–257.

[2] J. BEARDWOOD, J. H. HALTON AND J. M. HAMMERSLEY, *The shortest path through many points*, Proc. Cambridge Phil. Soc., 55 (1959), pp. 299–327.

[3] R. BELLMAN, *Dynamic programming treatment of the travelling salesman problem*, J. Assoc. Comput. Mach., 9 (1962), pp. 61–63.

[4] J. L. BENTLEY AND J. B. SAXE, *Decomposable searching problems*, 1: *Static-to-dynamic transformation*, J. Algorithms, 1 (1980), pp. 301–358.

[5] N. CHRISTOFIDES, *Worst-case analysis of a new heuristic for the travelling salesman problem*, Technical Report of the Graduate School of Industrial Administration, Carnegie–Mellon Univ., Pittsburgh, PA, 1976.

[6] H. GABOW, *An efficient implementation of Edmond's algorithm for maximum matching on graphs*, J. Assoc. Comput. Mach., 23 (1976), pp. 221–234.

[7] M. R. GAREY, R. L. GRAHAM AND D. S. JOHNSON, *Some* NP*-complete geometric problems*, in Proc. Eighth ACM Symposium on Theory of Computing, 1976, pp. 10–22.

[8] M. HELD AND R. M. KARP, *A dynamic programming approach to sequencing problems*, J. Soc. Indust. Appl. Math., 10 (1962), pp. 196–210.

[9] M. IRI, K. MUROTA AND S. MATSUI, *Linear-time approximation algorithms for finding the minimum-weight perfect matching on a plane*, Inform. Process. Lett., 12 (1981), pp. 206–209.

[10] R. M. KARP, *The probabilistic analysis of some combinatorial search algorithms*, in Algorithms and Complexity: New Directions and Recent Results, J. F. Traub, ed., Academic Press, New York, 1977.

[11] C. H. PAPADIMITRIOU, *The Euclidean traveling salesman problem is* NP*-complete*, Theoret. Comput. Sci., 4 (1977), pp. 237–244.

[12] ———, *The probabilistic analysis of matching heuristics*, in Proc. Fifteenth Annual Allerton Conf. on Communication, Control and Computing, 1977, pp. 368–378.

[13] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[14] E. M. REINGOLD AND K. J. SUPOWIT, *Probabilistic analysis of divide and conquer heuristics for minimum weighted Euclidean matching*, Networks, to appear.

[15] E. M. REINGOLD AND R. E. TARJAN, *On a greedy heuristic for complete matching*, this Journal, 10 (1981), pp. 676–681.
[16] C. A. ROGERS, *Packing and Covering*, Cambridge Univ. Press, Cambridge, 1964.
[17] K. SUPOWIT AND E. M. REINGOLD, *Divide and conquer heuristics for minimum weighted Euclidean matching*, this Journal, this issue, pp. 118–143.
[18] R. E. TARJAN, *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comput. Mach., 22 (1975), pp. 215–225.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0010 $01.25/0

SIMPLE CONSTRUCTIONS FOR MULTI-TERMINAL NETWORK FLOW SYNTHESIS*

DAN GUSFIELD†

Abstract. The multi-terminal network flow synthesis problem is one of the few nicely solved problems in network design, and is used widely in courses and texts on combinatorial optimization as an example of an elegantly solved problem. The solution used in these texts is due to R. E. Gomory and T. C. Hu. We present two simpler algorithms which improve the original method in speed, simplicity of the needed data structures and, most importantly, in the simplicity of the networks that are constructed. The networks constructed are planar and "uniformly optimal," permit simple flow routing methods and simple solutions to many sensitivity and postoptimality questions, and have as few edges as any networks produced by the Gomory–Hu method. Further, one algorithm constructs networks with only one node of degree larger than three, while the other algorithm constructs networks in which no node has degree greater than four.

Key words. network flow, graph algorithms, network synthesis, sensitivity analysis, planar graphs

1. Introduction. The multi-terminal network flow synthesis problem is one of the few nicely solved problems in the area of network design. It is used widely in courses and texts [2], [3], [6], [7] on network flows and combinatorial optimization as an example of an elegantly solved combinatorial optimization problem. The solution used in these texts is due to Gomory and Hu [5], and is also cited as an example of a nondirect application of maximum spanning trees. In this paper we refer to the Gomory and Hu method as Algorithm A. For examples where this problem arises, see also Chien [1]. For NP-hard network design problems see Wong [8] or Garey and Johnson [4].

We present two simpler algorithms which improve Algorithm A in speed, simplicity of needed data structures and, most importantly, in the simplicity of the networks constructed. The networks produced are "uniformly" optimal (defined in the next section), planar, have low node degrees, have as few edges as any produced by Algorithm A, and allow efficient and direct solutions to problems in sensitivity analysis. Further, flow routing algorithms for the networks are simple and can be implemented as rules applied locally at each node. Finally, it is shown that uniformly optimal networks with as few edges as any produced by Algorithm A can be derived from any triangulated polygon with the correct number of nodes.

The simplicity of the constructions suggest applications in computer networks and data transfer problems, particularly for bursty data transfer in local networks, where the primary costs are associated with providing sufficient channel capacity.

2. Problem set up and main result. Let R be a symmetric $n \times n$ matrix, and let $r(i, j) = r(j, i) \geqq 0$ be the i, j entry in R. R is called the *flow requirements matrix* and $r(i, j)$ is the i, j *flow requirement.* Let e denote the number of nonzero entries in R. In the case where e is small, we will assume that the nonzero entries of R are represented as an undirected graph to allow more efficient algorithms.

For G an undirected network with n nodes and a flow capacity on each edge, let $f(i, j)$ denote the maximum achievable flow in G between nodes i and j. In such a flow, i is the source, j is the sink, there is flow conservation at every other node, and the amount of flow sent along any edge is at most equal to the capacity of the

* Received by the editors June 22, 1981. This research was supported by the National Science Foundation under grants MCS77-09906, MCS78-07291, MCS 81-05894.

† University of California, Berkeley, California 94720. Present address: Computer Science Department, Yale University, New Haven, Connecticut, 06520.

edge. Of course, $f(i, j) = f(j, i)$. Note also that when flow is sent from i to j, the entire network is available for the i, j flow. See [2], [3], [6], [7] for a basic discussion of flow.

A network G with n nodes is called *feasible* for R if $f(i, j) \geqq r(i, j)$ for all node pairs i, j.

Given R, we seek a network G with edge capacities, which is feasible for R, and whose sum of edge capacities is minimum among all networks feasible for R. Any such network is called *optimal* for R. We further seek an optimal network H with flow function h, such that for any other optimal network G and its flow function f, $h(i, j) \geqq f(i, j)$ for all i, j pairs. Such a network is called *uniformly optimal*, and always exists [5].

2.1 Main result. We give two algorithms to solve the network synthesis problem. The first, Algorithm A′, runs in time $\text{Max}[e, n \log n]$ and constructs a network G' with the following desirable properties:

1. G' is uniformly optimal.
2. G' is planar.
3. At most one node in G' has degree greater than three.
4. G' has as few edges as any uniformly optimal network produced by Algorithm A.
5. The routing of flow in G' can be done by a distributed algorithm with each node making independent decisions on where to send the incoming flow it receives.
6. The structure of G' is easily expressed, and local modifications of R result in local and easily identified modifications of G'. This is useful for purposes of sensitivity analysis and for design problems where successive instances of the network flow synthesis problem are solved in the inner loop of a larger algorithm.

We then give a second algorithm, Algorithm A*, which runs in the same time as Algorithm A′, and which constructs a network G^* with the same six properties except that Property 3 is changed to:

3*. No node of G^* has degree greater than four.

Of the above seven properties, only the first one is guaranteed to hold for networks produced by Algorithm A. Further, Algorithms A′ and A* show that the use of the maximum spanning tree, and the revision of the original requirements needed by Algorithm A to achieve uniform optimality are unnecessary and undesirable.

3. Algorithms and constructions. We first present Algorithm A′, which constructs a planar uniformly optimal network with one node of high degree and all other nodes of degree three or less.

ALGORITHM A′

1) For each index i, compute $u(i) = \text{Max}_k [r(i, k)]$, and define $u(n+1) = 0$.
2) Sort the $u(i)$ values. Assume $u(i) \geqq u(i+1)$ for $i = 1, \cdots, n$. Note that it then follows that $u(1) = u(2) = \text{Max}[r(i, k)]$.
3) For $i = 2$ through n repeat the following:

3a) Create edge $(i, i-1)$ with capacity $u(i)/2$.

3b) Create edge $(i, 1)$ with capacity $[u(i) - u(i+1)]/2$, provided that the capacity is nonzero.

Figure 1 shows the result of Algorithm A′. The algorithm ignores most of the requirements, depending only on the $u(i)$ values. Therefore R is not displayed in Fig. 1, but the $u(i)$ values are written next to each node.

Algorithm A′ requires time $O(e)$ to find the $u(i)$ in step 1), $O(n \log n)$ to sort in step 2), and $O(n)$ time to construct G' in step 3). Note that the network produced is always planar, and only node 1 has degree greater than three.

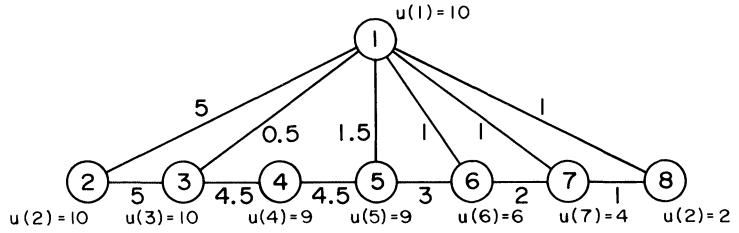


FIG. 1. G'.

We now show the correctness of Algorithm A'. Given R, let G' be the network produced by Algorithm A', and let f' be the flow function of G'.

LEMMA 3.1. $f'(i, j) = \text{Min}\,[u(i), u(j)]$ *for all node pairs* i, j.

Proof. Let i, j be two arbitrary nodes, and $u(i) \geqq u(j)$. Consider the path $P_{i,j}$ from i to j along the edges $(k, k+1)$ for $k = i$ through $j-1$. The edge with least capacity on $P_{i,j}$ is $(j-1, j)$, with capacity $u(j)/2$. Therefore, a flow of $u(j)/2$ is possible along the $P_{i,j}$ path.

Now consider $P_{1,i}$, the path with edges $(k, k+1)$ for $k = 1$ through $i-1$. The edge with least capacity on $P_{1,i}$ is $(i-1, i)$, with capacity $u(i)/2 \geqq u(j)/2$. G' is undirected, and so a flow of $u(j)/2$ from node i to node 1 is possible along the reverse of path $P_{1,i}$.

To complete the proof, we claim that a flow of $u(j)/2$ between nodes 1 and j is achievable without using any edge of $P_{1,j}$, the union of $P_{1,i}$ and $P_{i,j}$. The proof is by backward induction on the index j. For $j = n$, the claim is true, since the edge $(1, n)$ has capacity $u(n)/2$. Suppose the claim is true for $j = k+1 < n$, and consider node k. Let $F(k+1)$ be the flow of $u(k+1)/2$ from 1 to $k+1$ which avoids edges of $P_{1,k+1}$. By definition, $F(k+1)$ doesn't use edges $(k+1, k)$ or $(k, k-1)$, and so $F(k+1)$ also doesn't use edge $(1, k)$. Edge $(1, k)$ has capacity $u(k)/2 - u(k+1)/2$, and edge $(k+1, k)$ has capacity $u(k+1)/2$, for a total capacity of $u(k)/2$. Then to send $u(k)/2$ from 1 to k avoiding $P_{1,k}$, send $u(k+1)/2$ from $k+1$ to k along edge $(k+1, k)$, and send the rest along edge $(1, k)$. The flow of $u(k+1)/2$ to node $k+1$ is sent via $F(k+1)$, and the proof is complete. □

THEOREM 3.1. G' *is uniformly optimal for* R.

Proof. By Lemma 3.1, G' is clearly feasible for R since $u(i) \geqq r(i, j)$ and $u(j) \geqq r(i, j)$. To show optimality, note that the total capacity of the edges incident to any node i is $u(i)$, which is the minimum capacity possible in any feasible network. Now suppose there is an optimal network G with flow function f, such that $f(i, j) > f'(i, j)$, for some node pair i, j. Then $f(i, j) > \min\,[u(i), u(j)]$, and if $u(i) \geqq u(j)$, node j must be incident in G to edges with total capacity exceeding $u(j)$. Therefore, G can't be optimal, and G' is uniformly optimal. □

The network G' constructed by algorithm A' has the undesirable property that node 1 has high degree. For applications involving ports into a computer, or wires wrapped on pins, small node degrees are desired. We now present an algorithm A* which requires the same time as algorithm A' and constructs a planar, uniformly optimal network G^* with the same number of edges as G', and with the property that no node of G^* has degree greater than four.

ALGORITHM A*

1) For each index i, compute $u(i) = \text{Max}_k\,[r(i, k)]$. Call $u(i)$ the *node weight* of node i.
2) Let $\{w(1), \cdots, w(t)\}$ be the set of distinct node weights. Then for each i, there is a unique index j such that $u(i) = w(j)$. Order these t distinct node

weights so that $w(1) > w(2) > \cdots > w(t)$. We first construct a network $G(t)$ containing $t+1$ nodes, one for each of the distinct node weights $w(2)$ through $w(t)$, and two for the node weight $w(1)$. The former set of nodes is $\{2, \cdots, t\}$ and latter set is $\{0, 1\}$. $G(t)$ is constructed in steps 3) through 5).

3) Create an edge from node 0 to node 1 of capacity $w(1) - w(2)/2$, and create an edge from node 0 to node 2 of capacity $w(2)/2$.
4) For $i = 1$ to $t-2$ do the following:
4a) Create an edge from node i to node $i+1$ of capacity $[w(i+1) - w(i+2)]/2$.
4b) Create an edge from node i to node $i+2$ of capacity $w(i+2)/2$.
5) Create an edge from node $t-1$ to node t of capacity $w(t)/2$.
6) If the requirements R specify more than one node of weight $w(i)$, $i > 1$, then insert the new nodes of weight $w(i)$ into the $(i, i-2)$ edge of G, creating a path of edges between node i and $i-2$, each with capacity $w(i)/2$. If R contains more than two nodes of weight $w(1)$, then first split the (0, 1) edge into two parallel edges, one with capacity $w(1)/2$, and the other with capacity $[w(1) - w(2)]/2$. Then insert the new nodes of weight $w(1)$ into the edge with capacity $w(1)/2$, creating a path of edges between nodes 0 and 1, each with capacity $w(1)/2$.
7) The result of steps 1) through 6) is G^*.

Figure 2 shows the result of steps 1 through 5 for $t = 6$. Figure 3 shows G^* for the $u(i)$ values given in Fig. 1.

THEOREM 3.2. *G^* is uniformly optimal for R.*

Proof. We first prove the claim that in $G(t)$, $f(i, j) = \text{Min}[w(i), w(j)]$. The proof is by induction on t. $G(1)$ consists of a single edge of capacity $w(1)$ between nodes 0 and 1, and $G(2)$ is the graph shown in Fig. 4, and it is immediate that the claim is correct for $G(1)$ and $G(2)$.

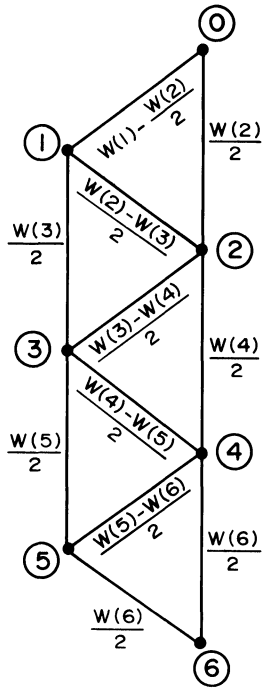


FIG. 2. *$G(t)$ for $t = 6$.*

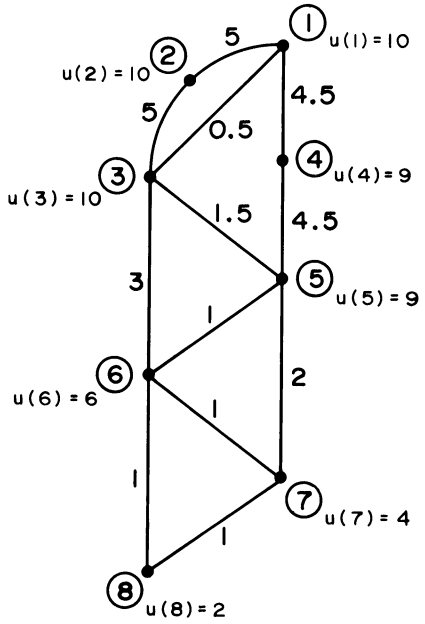


FIG. 3. *G^* Using the same requirements as in Fig. 1.*

Suppose that $f(i, j) = \text{Min}\,[w(i), w(j)]$ in $G(t)$ for $t \geqq 2$. $G(t+1)$ differs from $G(t)$ by the change in capacity of the $(t, t-1)$ edge from $w(t)/2$ to $[w(t)-w(t+1)]/2$, and by the addition of a node $t+1$ with two edges $(t-1, t+1)$ and $(t, t+1)$, each with capacity $w(t+1)/2$. Then any amount of flow that can be sent along the $(t, t-1)$ edge in $G(t)$ can be sent between t and $t-1$ in $G(t+1)$ via the $(t, t-1)$ edge together with the $(t, t+1, t-1)$ path. Hence the claim is proved for $i, j \neq t+1$. If $i = t+1$ and $j = t$, then a flow of $w(t+1)$ in $G(t+1)$ is possible by sending half via the $(t+1, t)$ edge, and half via the $(t+1, t-1, t)$ path. Then if $i = t+1$, a flow of $w(t+1)$ can be sent to any other node j by first sending it to node t, since $w(t+1) < w(t)$, and by induction a flow of $w(t)$ is possible from t to any other node j in $G(t)$. Hence the claim is proved for $G(t)$.

Now we can extend the claim to G^*, i.e., that in G^*, $f(i, j) = \text{Min}\,[u(i), u(j)]$. The theorem will then follow as in the proof of Theorem 3.1. Suppose k is a node of weight $w(i)$ in R but not in $G(t)$. Then k is inserted into the $(i, i-2)$ edge of $G(t)$,

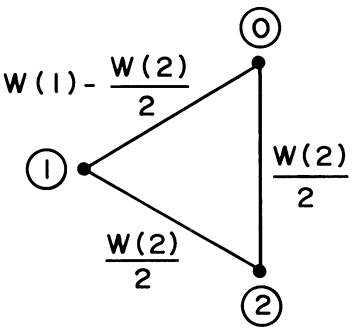


FIG. 4. $G(2)$.

and is incident with two edges of capacity $w(i)/2$. In $G(t)$ the capacity of the $(i, i-2)$ edge is $w(i)/2$, and the sum of the capacities of the other edges incident with node i is also $w(i)/2$. Hence flow to nodes i or $i-1$ is unaffected by the insertion of node k, and it is clear also that $f(k, j) = f(i, j)$ for any node j such that $u(j) = w(i)$. To complete the proof it is sufficient to show that $f(i, k) = w(i) = u(k)$. This is done by sending $w(i)/2$ from i to k along the $(i, i-1)$ path, together with a flow of $w(i)/2$ which is sent first from i to $i-1$ without using any edge of the $(i, i-1)$ path, and then from $i-1$ to k along the $(i-1, i)$ path. This completes the proof. □

3.1 Flow routing. The regularity and simplicity of G' and G^* permit simple flow routing algorithms which can be implemented locally at each node in the networks. Thus there is no need to run standard flow routing algorithms, or to store large routing tables at each node. We will give, without proof, partial routing rules for G^*. The full rules for G^* and the rules for G' are equally simple. Suppose $u(i) \leqq u(j)$, and a flow of value $v(i, j) \leqq u(i)$ is sent from node i to node j. Then for any node k such that $u(k) \leqq u(i)$, k forwards flow using the following three rules:

1. Node k sends no flow to nodes which have sent flow to k.
2. Any edge used to send flow out of k is used to saturation if possible.
3. Node k attempts to send flow to its neighboring nodes in order of their node weights, largest first.

For any node k such that $u(k) > u(i)$, k forwards flow using rules 1 and 2, but modifies rule 3 as follows:

3′. If node k is adjacent to node j, then k sends all of its flow directly to node j. Else, node k attempts to send flow to its neighbors of equal node weight, and then to its other neighbors in order of their node weights, largest first.

To implement this routing scheme, a broadcast message is sent giving the source and destination of the flow, before the actual flow begins. If the flow consists of messages that can be prefixed with a header giving that information, then the broadcast is unnecessary. It can be proven that the above scheme works under any order in which the nodes receive flow and decide how to reroute. Hence there are no synchronization problems in the implementation.

3.2 Sensitivity analysis. Both constructions G' and G^* have the useful property that many local modifications of R induce only local modifications in G' or G^*. Therefore many postoptimality and sensitivity analysis questions can be answered without rerunning the full construction algorithm. If the flow requirements of R are changed, but no node weights are changed, then G' and G^* remain unchanged. If node weights are changed, but their order remains unchanged, then only the capacities and not the underlying networks change. Further, since the edge capacities in both G' and G^* are expressed as functions of at most two node weights, the update of the capacities is simple. If new nodes are added to R, but the set of distinct node weights remains unchanged, then new nodes are added to G' as in step 3) of Algorithm A′, and to G^* as in step 6) of A*. If nodes are deleted from R without changing the set of node weights, then those nodes are deleted from G' or G^* by the obvious reversal of steps 3) and 6). If all the nodes of a given node weight $w(i)$ are removed from R, then all those nodes and the edges incident to them are removed from G^*, two new edges are inserted, and one additional edge capacity is changed. In G' the deleted nodes and the incident edges are removed, one new edge is inserted, and one edge capacity is changed. Further, every node in either G' or G^* that is affected by these changes is a neighbor of a node of weight $w(i)$. Similarly, if nodes of a new node weight are added to R, then the updated G^* differs from the previous G^* by the

insertion of a connected subgraph containing the new nodes, by the deletion of two edges running between neighbors of the new nodes and by the modification of one other edge capacity. The updated G' differs from the previous G' by the insertion of a subgraph with the new nodes and the deletion of one old edge and the modification of one capacity. If R is changed so that order of the node weights changes, then G^* and G' can be updated by considering this change as a deletion of all nodes of a given node weight, and the insertion of new nodes of a different node weight.

4. Number of edges. We now consider the number of edges in the network produced by Algorithms A′ and A*. We show that there are uniformly optimal networks with fewer edges, but that none of these networks are constructable by the Gomory–Hu method (Algorithm A). We begin with a description of Algorithm A and an analysis of the number of edges in the networks constructed by it.

ALGORITHM A

0) For each index i in R, compute $u(i) = \text{Max}_k\,[r(i,k)]$. For every pair i, j change $r(i,j)$ to $\text{Min}\,[u(i), u(j)]$, creating matrix R'. Let H be a weighted graph with n nodes, defined by considering R' as the weighted adjacency matrix of H.
1) Compute a maximum weight spanning tree T of H.
2) Decompose T into a sum of subtrees, each having edges of equal weight. To do this, define the decomposition of a tree T_i recursively. If c_i is the smallest edge weight in T_i then T_i is decomposed into one copy of T_i with weight c_i on each edge, plus the decomposition of each subtree of T_i resulting from deleting all edges of weight c_i from T_i, and subtracting c_i from the weights of all the remaining edges.
3) For every tree T_i in the decomposition, create a cycle C_i containing all the nodes of T_i. Set the capacity of every edge in C_i to $c_i/2$, where c_i is the weight of each edge in T_i. Superimpose all of the cycles, merging common edges and summing the capacities. The resulting network G is uniformly optimal for R.

For further explanation of Algorithm A see one of [2], [3], [5], [6], [7].

Let H and T be as above, and let G be the network constructed by Algorithm A. To establish the number of edges in G we first examine the structure of T.

LEMMA 4.1. *Let x be any edge in T. If x has weight c_x, then the removal of x from T creates two connected components, at most one of which contains an edge of weight greater than c_x.*

Proof. The lemma is trivially true if one of the endpoints of x is a leaf of T, so suppose this is not the case. Let G_y and G_z be the two components of $T-x$, with edge y of weight $c_y > c_x$ in G_y, and edge z of weight $c_z > c_x$ in G_x. By the definition of H, $c_y = u(i)$ for some node i in G_y, and $c_z = u(j)$ for some node j in G_z. Then H contains the edge (i,j) across the G_y, G_z cut, and (i,j) has weight greater than c_x, hence T can't be a maximum weight spanning tree of H. □

THEOREM 4.1. *For a given requirements matrix R, G' and G^* contain the same number of edges, and G contains at least that many.*

Proof. We first examine the number of edges of G. Recall that $w(1) > \cdots > w(t)$ are the t distinct node weights defined by R. From the definition of R', the only edge weights in T are $w(1)$ through $w(t)$, and all edges incident with any node i in T have weight less than or equal to $u(i)$. Further since T is a maximum spanning tree, every node i is incident in T with at least one edge of weight $u(i)$. It follows then from Lemma 4.1 that, for any $k < t$, the deletion from T of all edges of weight $w(k)$ or less leaves one connected subtree containing all nodes with node weights greater than $w(k)$ and no nodes with weight $w(k)$ or less.

We now examine the edges generated by the synthesis step 3), ignoring the capacities assigned. We claim that G is the superposition of t cycles, C_1 through C_t. C_t connects all the n nodes of T, and for $k<t$, C_k contains all nodes of weight $w(k)$ or more and no nodes of weight $w(k+1)$ or less. To see this, recall that the decomposition step 2) of Algorithm A generates a sequence of subtrees of T, by beginning with T itself and successively deleting all edges of weight $w(t)$ down to $w(1)$. Step 3) creates a cycle through the nodes of every new subtree generated in this way, and the claim follows from the structure of these subtrees, which was established above.

We can now count the number of edges of G. For k from 1 through t, let N_k be the number of nodes of weight $w(k)$. For $k \geqq 2$, cycle C_k contains all nodes of weight $w(k)$, and at least one node of greater weight. Therefore, at least N_k+1 edges of C_k are incident with some node of weight $w(k)$. None of these N_k+1 edges can appear in any other cycle C_j, for $j<k$, and so the cycles C_2 through C_t must contain at least $(n-N_1)+(t-1)$ distinct edges. Cycle C_1 contains N_1 edges, and so G contains at least $n-1+t$ edges if $N_1>2$, and $n-2+t$ edges if $N_1=2$.

We now examine the number of edges in G' and then G^*. Step 3b) of Algorithm A′ produces the edge 1, i if and only if $u(i)>u(i+1)$, and so produces t edges in total. Step 3a) produces $n-1$ edges, and so G' contains no more than $n-1+t$ edges. However, $u(1)=u(2)$, so the edge $(1, 2)$ is counted twice if $N_1>2$. Therefore, G' contains $n-1+t$ edges if $N_1>2$ and $n-2+t$ otherwise.

To count the number of edges in G^* note that $G(t)$ contains $2t-1$ edges. In the case where $N_1=2$, step 6) adds $n-t-1$ edges to $G(t)$, and otherwise it adds $n-t$ edges. Hence G^* and G' contain the same number of edges, and G contains at least that many. □

COROLLARY 4.1. *G contains the same number of edges as G' and G^* if and only if the nodes of weight $w(k)$ form a single subpath in C_k, for all k from 1 to t.*

Given an $n \times n$ requirements matrix R, an undirected graph K on n nodes is called a *shell* of R if capacities can be assigned to the edges of K so that K is uniformly optimal for R.

COROLLARY 4.2. *Let $n=t+1$ with $N_1=2$, and $N_k=1$ for all $k=2, \cdots, t$. Then any triangulated polygon K with n nodes and $2n-3=2t-1$ edges is a shell for R.*

To see this, note that in any triangulated polygon K there is always a node v of degree two. We can assign v the node weight $w(t)$. Then the outermost cycle of K is C_t. Removing v from K gives another triangulated polygon with a node of degree two. This node gets weight $w(t-1)$ and the outermost cycle is C_{t-1}. In this way, K decomposes into cycles C_k for k from t to 2, and a single edge connecting the last two nodes of K which are given node weights $w(1)$. This last edge is given capacity $W(1)/2$, and the capacity of any other edge x in K is half the sum of all $w(k)$ such that x is contained in cycle C_k. It is clear by the proof of Theorem 4.1 that this decomposition could have been produced by Algorithm A and hence is uniformly optimal.

In the case where there are more than two nodes of weight $w(1)$ or more than one node of any other weight, these additional nodes can be inserted into the shell in a way similar to the insertion of nodes in step 6) of Algorithm A*.

Note that there are uniformly optimal networks with fewer edges than G'. See Fig. 5. Such networks are, of course, not produced by Algorithm A, A′ or A*, and it is an open question whether there exist fast algorithms to minimize the number of edges in a uniformly optimal network.

5. A related problem. A related problem, also solved first in [5], is to construct, if possible, an optimal network which exactly satisfies the requirements in R. Analogous

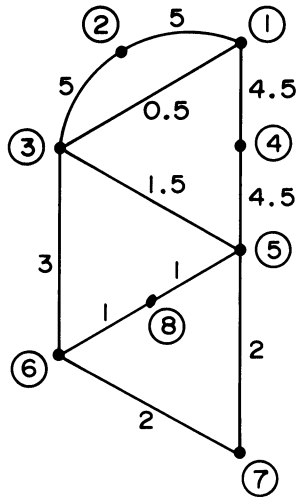


FIG. 5. *G" has the same flow function as G' and G* of Figs.* 1 *and* 3, *but uses one less edge.*

to the results in this paper, constructions exist for this problem, which are planar, have low average node degree, and have as few edges as any networks produced by the algorithm in [5]. A simple algorithm to generate these constructions is the subject of a forthcoming paper by the author.

Acknowledgment. I would like to thank L. Babai for pointing out Corollary 4.2, and many thanks to David Lichtenstein for his helpful comments.

REFERENCES

[1] R. T. CHIEN, *Synthesis of a communication net*, I.B.M. J., July 1960.
[2] L. R. FORD AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.
[3] H. FRANK AND İ. T. FRISCH, *Communication, Transmission and Transportation Networks*, Addison-Wesley, Reading, MA, 1972.
[4] M. GAREY AND D. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
[5] R. E. GOMORY AND T. C. HU, *Multi-terminal network flows*, SIAM J. Appl. Math., 9 (1961), pp. 551–570.
[6] T. C. HU, *Integer Programming and Network Flows*, Addison-Wesley, Reading, MA, 1969.
[7] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
[8] R. T. WONG, *A Survey of Network Design Problems*, Operations Research Center working paper OR 080-78, Massachusetts Institute of Technology, Cambridge, MA, August 1978.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0011 $01.25/0

FORMAL SEMANTICS AND ABSTRACT PROPERTIES OF STRING PATTERN OPERATIONS AND EXTENDED FORMAL LANGUAGE DESCRIPTION MECHANISMS*

A. C. FLECK† AND R. S. LIMAYE‡

Abstract. Two formal models of string patterns are introduced. One is based on an idealization of patterns as systems of set equations, and the other is based on an abstract procedure model of a pattern. Each model is then shown to yield certain insights and in particular is used to explore an operation not commonly considered in conjunction with string patterns. These two new pattern operations are (set) complementation and reversal of cursor direction. Each is shown to have a dramatic effect on both the expressive power and the complexity of patterns in which they are included. These pattern operations may in turn be regarded as extensions to the usual formal language definition mechanisms (i.e., grammars and equation systems), and our results interpreted in terms of formal language description.

Key words. string patterns, cursor functions, cursor reversal, complementation, formal language description

Introduction. We deal in this paper with the concept of a string pattern. This concept is best exemplified and implemented in greatest generality in the programming language SNOBOL4 [16]. Two things about this facility are notable for our purposes. First of all, the pattern mechanism is widely recognized to be extremely difficult to explain and use [15], [23], [25]. Secondly, despite these difficulties there has been essentially no evolution of the pattern concept since the SNOBOL4 language was first developed, an undue period in relation to progress in other areas of programming languages. We feel that this situation has been largely brought about by two facts. First the semantics of SNOBOL4 patterns have usually been described in terms of a specific pattern matching algorithm (see for instance [13], [16]). This we view as akin to defining a programming language by its compiler, a situation we feel must be changed (i.e., augmented) before much progress is possible. Secondly, the generality of the mechanism has been achieved by adding a great many functional pattern elements, essentially a new function for each special purpose which occurs.

Hence our goals are two-fold. First of all we are interested in developing abstractions which allow for the specification of the complex pattern semantics without becoming involved in the excessive detail and overspecification of a particular algorithm. This should be of indirect benefit in such work as the search for more efficient pattern matching algorithms (e.g., [20], [21]). Secondly, we suggest the evolution of pattern mechanisms in the alternative direction of a few very general pattern operations which may be adapted to numerous purposes. Furthermore our work develops the interaction of these two goals in that we follow the naturalness of each of the two abstractions we develop to lead us to a proposal for a specific new operation that seems to have great promise of generality and usefulness.

While we are greatly indebted to the idea of the string pattern in the sense of SNOBOL4 [16], we do not intend to remain faithful to any particular implementation or semantic algorithm. Rather, our approach is to seek formalisms which fairly well model existing structures, but then use the formalism to obtain a consistent and general

* Received by the editors April 2, 1980, and in revised form May 25, 1982. Work on this paper was supported in part by the National Science Foundation under grants DCR75-05296 and MCS77-03902.

† Computer Science Department and Weeg Computing Center, University of Iowa, Iowa City, Iowa 52242.

‡ Computer Science Department, University of Iowa, Iowa City, Iowa 52242.

conceptualization which is not fraught with idiosyncracies. Our concern is thus both for developing formal semantic models which are appropriate, and for identifying a few operations which may be shown to have substantial effect on expressive power.

Our assumption will be that the reader is familiar with the concepts and terminology of [16]. We emphasize that the models we explore here isolate the string pattern (and the pattern matching process) and treat it in a manner apart from any other programming features.

1. Patterns augmented with reversal: The procedure model and basic definitions. It will be helpful in this section if the reader is familiar with Gimpel's model of patterns [12], [13]. The central abstraction is to view a pattern as a description of the way in which the "cursor" is advanced in an arbitrary subject string. Thus Gimpel's model views a pattern as a function from a pre-cursor position (and subject string) to post-cursor positions. Gimpel admits, in fact, arbitrary such functions regardless of their computability (or noncomputability) attributes. This is an assumption in which we do not concur, and we present an implementation abstraction for cursor manipulation which implicitly contains computability constraints.

As the first step we define the syntax for a collection of string patterns. For a given (nonempty) alphabet of characters C, we use the notations C^* to represent the set of all finite strings (sequences) over C, and λ to denote the null sequence. For $s = c_1c_2 \cdots c_n \in C^*(n \geqq 0)$ with $c_i \in C$ $(1 \leqq i \leqq n)$, we say that the *length* of s is n; also we use the indexing notation $s[i] = c_i$.

The following definition is phrased in terms of two binary operations *alternation* and *concatenation* and the unary operation *reversal* which are all as yet unspecified; concatenation is denoted simply by juxtaposition of its two operands, alternation is denoted by the symbol "|". Often we will abuse this notation in writing such expressions as $a|b|c$. For the time being the reader can always supply leftmost association (i.e., $(a|b)|c$); we will see shortly that our specification leads to both these operations being associative hence avoiding any actual ambiguity in these abuses.

The idea of a string scanning mechanism that can move in either a left-to-right or a right-to-left direction with respect to any string under consideration is very well known in automata theory (see Rabin and Scott [22] and Gray, Harrison and Ibarra [14]). Doyle [8] has suggested ideas along these lines for patterns, with specific pattern primitives in the form of functions controlling scan (cursor) direction. Our suggestions differ from both these ideas in that instead of specifying an absolute direction for the scan (i.e., left-to-right or right-to-left), we provide for the specification of a relative change (i.e., reversal) of a direction currently in effect (the initial direction is left-to-right). This we show leads to more natural rules of pattern calculus. Our suggestion is closer in spirit to that of Stewart [25] (in fact we borrow his notation in the definition below). Stewart was guided by a model with a strong algebraic orientation and was seeking an operation which would function as a multiplicative (i.e., concatenation) inverse. However, he found that true inverses do not exist in his model, and the "semi-inverse" Stewart does achieve fails to exist for many patterns and is not unique for many others. In contrast, our (cursor) reversal operation may be applied to any pattern and, as we shall show, always yields a unique result. As reversal has a number of formal properties in common with an inverse operation, we adopt the usual inverse notation for it; the reader is cautioned not to be misled as this is not the appropriate conceptualization.

DEFINITION 1. Let disjoint nonempty, finite sets C (*alphabet*) and I (*identifiers*) be given. A *pattern equation system with reversal* $\mathcal{S} = \mathcal{S}(C, I)$, is defined hierarchically

as follows:

a) a *factor* is any element from $C \cup I \cup \{a^{-1} | a \in C\} \cup \{X^{-1} | X \in I\}$;
b) a *term* is either Λ, Φ, a factor or any concatenation of factors;
c) an *expression* is either a term or any alteration of terms;
d) an *equation* is written $X = \mathscr{E}$ and consists of an ordered pair of elements where $X \in I$ and $\mathscr{E}$ is an expression;
e) a *pattern equation system* $\mathscr{S}$ consists of a collection of equations where each $X \in I$ occurs exactly once as the left (i.e., first) element of an equation in $\mathscr{S}$.

In case the only factors which occur in a pattern equation system are from $C \cup I$, we say the system is *context-free*.

Example 1. Let $C = \{a, b\}$ and $I = \{X, Y\}$. Then

$$\mathscr{S}_1 = \begin{Bmatrix} X = ab | aY \\ Y = b | Xb \end{Bmatrix}$$

is a pattern equation system.

The usual convention for "matching a string against a pattern" consists of advancing (i.e., moving the cursor left-to-right) across a substring which conforms to the description of the pattern. With the addition of cursor reversal, this matching process can in fact cause the retreating (i.e., moving the cursor right-to-left), and so substrings can be scanned repeatedly for multiple purposes. The descriptive power of this facility will be illustrated shortly. For the moment we give a simple example of its use.

Example 2. Let $C = \{a, b\}$ and $I = X$. Then $\mathscr{S}_2 = \{X = aX | bb^{-1}\}$ is a pattern equation system.

Note that in Example 2, the intent is for the pattern to designate a sequence of the form $a^n b (n \geqq 0)$, but that after the matching process, the terminating "b" is still the next character to be scanned (even though its presence has been verified). Our development below is intended to provide the machinery to enable us to make such informal statements precise and unambiguous.

It should be noted that while we impose some superficial restrictions on the form of equations so that the technical details can be developed more succinctly, there is no real loss of generality. For instance, an equation such as $X = (a^{-1}Y)^{-1}$ is not allowed (reversal can only be applied to individual characters and identifiers), but one can simply add a new identifier, say Z, and write the equivalent system

$$X = Z^{-1}, \qquad Z = a^{-1}Y.$$

In the following we will not write the formal set notation for pattern equation systems but will simply list the equations. As we remarked above, Definition 1 specifies only the syntax or form of the pattern definitions we consider. The more interesting aspect is the semantics. Our approach to this is a constructive one, and we first define a simple programming language to be used to specify these semantics. We give only an informal description of the syntax and semantics of this programming language. We trust that its simplicity makes this sufficient, so that we will not be unduly diverted from our pursuit of the string pattern ideas.

DEFINITION 2. Conventions in the *programming language for pattern system definition*, PLPSD:

1) Expressions. Expressions may be either integer-valued, string-valued, or Boolean-valued; they may involve the usual constants, variables, operations, tests and function calls (in particular the string indexing notation described above). In addition

we allow the special notation θ to occur as an expression (and interpret it as the undefined or failure value). Also function calls are written in the usual way (i.e., "function-ident (argument list)") and parameters are transmitted by value.

2) Function return statement, written

return (⟨expression⟩).

The expression is evaluated and the value (if any) is returned to the point of invocation.

3) Conditional statement, written

if ⟨Boolean-valued expression⟩
then ⟨Function return statement⟩
else ⟨Function return statement⟩.

The Boolean expression is evaluated and if its value is true the statement following **then** is executed (and so control is next returned to the appropriate invocation); if the Boolean value is false the statement following the **else** is executed.

4) Nondeterministic selection, written

select
⟨Function return statement⟩;
⟨Function return statement⟩;
⋮
⟨Function return statement⟩
end.

Exactly *one* of the enclosed statements is executed, and in the case of any of the choices a *valid run step* is said to have occurred. That is, the **select** plays a role analogous to the **case**-statement, except that rather than including an expression to determine the selection, the selection is left to be made on an unspecified basis. Thus there is a close analogy of the **select** semantics with that of Floyd's "choice function" [11] (but a statement rather than a value is chosen). Our form is rather different, however, more like the nondeterministic control structure of Allison [2], but still devised to suit our particular purposes.

5) Function definition, written

define ⟨identifier⟩(⟨parameter list⟩);
⟨select statement⟩|⟨conditional statement⟩
end ⟨identifier⟩.

Parameters are restricted to being strings or integers (and are transmitted by value as noted above). Recursion is allowed (in fact, relied on) in our function definitions.

With the inclusion of the nondeterministic control structure (i.e., **select**), each PLPSD program may admit a whole collection of valid runs rather than defining a single uniquely specified execution (a **run** may be finite or infinite and refers to a sequence of statement executions, where each statement in the sequence is a valid successor to the previous). Each finite run will necessarily terminate with a value and the resulting collection of values is regarded as the set of potential outcomes of any single execution. Note that finite and infinite runs are not mutually exclusive, and in particular that the existence of infinite runs (i.e., nontermination) does not preclude the occurrence of a nonempty set of outcomes. For a formal treatment of the interactions between recursion and nondeterminism and their influence on such issues as program termination and equivalence, the reader may wish to consult [7]; these are not matters which we pursue here. Our purpose here is to develop a suitable semantic model. While this is an operationally oriented model, it definitely does not solve all the implementation problems.

Of course one could allow much greater generality in the combination of the programming elements than we have introduced here, but this will be sufficient for our purposes.

One last preliminary which will aid the presentation of our semantics is the idea of a simple code generation function.

DEFINITION 3. The function "code" is defined hierarchically to operate on (pattern) terms and expressions (see Definition 1) to produce statements in PLPSD as follows:

1) for a term t
   a) if $t = \Phi$, then code $(t) =$ **return** (θ)
   b) if $t = \Lambda$, then code $(t) =$ **return** $(\Lambda(s, c))$
   c) if $t = f_1 f_2 \cdots f_p$, where $p \geqq 1$ and $f_i (1 \leqq i \leqq p)$ is a factor, then code $(t) =$ **return** $(f_p(s, f_{p-1}(s, \cdots, f_2(s, f_1(s, c)) \cdots)))$ and code $(t^{-1}) =$ code $(\bar{f}_p \bar{f}_{p-1} \cdots \bar{f}_2 \bar{f}_1)$, where

$$\bar{f}_i = \begin{cases} a^{-1} & \text{if } f_i = a \in C, \\ X^{-1} & \text{if } f_i = X \in I, \\ a & \text{if } f_i = a^{-1} \quad \text{with } a \in C, \\ X & \text{if } f_i = X^{-1} \quad \text{with } X \in I; \end{cases} \qquad (1 \leqq i \leqq p)$$

2) for an expression $\mathscr{E} = t_1 | t_2 | \cdots | t_q$, where $q \geqq 1$ and $t_i (1 \leqq i \leqq q)$ is a term,

code $(\mathscr{E}) =$ **select**
code (t_1);
code (t_2);
⋮
code (t_q)
end

and

$$\text{code } (\mathscr{E}^{-1}) = \text{code } (t_1^{-1} | t_2^{-1} | \cdots | t_q^{-1}).$$

The required preliminaries have now been completed. We now associate with each pattern equation system a collection of function definitions (written in PLPSD) which will serve as the semantic interpretation.

DEFINITION 4. Let $\mathscr{S} = \mathscr{S}(C, I)$ be a pattern equation system. We associate with $\mathscr{S}$ a set of *function definitions* $\mathscr{F}_{\mathscr{S}}$ (each definition has two parameters, a string (the *subject*) and an integer (the *pre-cursor position*), and returns an integer (a *post-cursor position*) or θ) in PLPSD that includes exactly:

1) the function definition
   define $\Lambda(s, p)$;
   /* the NULL pattern element */
   if $p \neq \theta$ and $0 \leqq p \leqq$ length (s)
   then return (p)
   else return (θ)
   end Λ
2) for each $a \in C$ the definition
   define $a(s, p)$;
   /* forward character-matching pattern elements */

if $p \neq \theta$ and $0 \leqq p <$ length (s) and $s[p+1] = 'a'$
then return $(p+1)$
else return (θ)
end a

3) suppose $I = \{X_1, X_2, \cdots, X_n\}$ and $\mathscr{S} = \{X_i = \mathscr{E}_i\}_{1 \leqq i \leqq n}$; for each $X_i \in I$
define $X_i(s, p)$
/* pattern elements constructed from pattern equations */
code $(\mathscr{E}_1)$
end X_i

4) for each $a \in C$ the definition
define $a^{-1}(s, p)$;
/* backward character-matching pattern elements */
if $p \neq 0$ **and** $0 < p \leqq$ length (s) **and** $s[p] = 'a'$
then return $(p-1)$
else return (θ)
end a^{-1}

5) suppose $I = \{X_1, X_2, \cdots, X_n\}$ and $\mathscr{S} = \{X_i = \mathscr{E}_i\}_{1 \leqq i \leqq n}$; for each X_i
define $X_i^{-1}(s, p)$;
/* reversal pattern elements constructed from pattern equations */
code $(\mathscr{E}_i^{-1})$
end X_i^{-1}.

For each of the identifiers $X_i \in I$ in the system $\mathscr{S}$ and string $s \in C^*$, consider the outcome of a function call such as $X_i(s, p)$. Such a call may initiate a number of possible valid runs, each of which may be either finite or infinite; each of the finite runs must return a value—either θ or an integer p' where $0 \leqq p' \leqq$ length (s). We think of this collection of values as the potential outcomes for the cursor position after the pattern expression $\mathscr{E}_i$ has been "applied" to string s starting at position p.

Example 3. Let $C = \{a, b\}$, $I = \{X_1, X_2\}$, $\mathscr{S}_3$ be

$$X_1 = a | aX_1, \qquad X_2 = X_1 b$$

and $\sigma = baab$.

The function definitions which result (we omit those in clauses 1 and 2 of Definition 4) are:

define $X_1(s, p)$;
  select
  return $(a(s, p))$;
  return $(X_1(s, a(s, p)))$
  end
end X_1;

define $X_2(s, p)$;
  select
  return $(b(s, X_1(s, p)))$
  end
end X_2.

Then

$$X_1(\sigma, 0) = \{\theta\}, \quad X_1(\sigma, 1) = \{\theta, 2, 3\}, \quad X_2(\sigma, 1) = \{\theta, 4\}.$$

To see, for instance, that $3 \in X_1(\sigma, 1)$, we note that

(a) $X_1(\sigma, a(\sigma, 1)) \subseteq X_1(\sigma, 1)$,
(b) $a(\sigma, 1) = \{2\}$ so $X_1(\sigma, a(\sigma, 1)) = X_1(\sigma, 2)$,
(c) $a(\sigma, 2) \subseteq X_1(\sigma, 2)$ and $a(\sigma, 2) = \{3\}$.

Similarly then, since $X_2(\sigma, 1) = b(\sigma, X_1(\sigma, 1))$ and $b(\sigma, 3) = \{4\}$, $4 \in X_2(\sigma, 1)$, etc.

Example 4. Let $C = \{a, b\}$, $I = \{X_1, X_2\}$ and $\mathscr{S}_4$ be

$$X_1 = ab | aX_2^{-1}, \qquad X_2 = b^{-1}X_1^{-1}.$$

Then the associated function definitions are (we omit those for clauses 1, 2 and 4):

define $X_1(s, p)$;
  select
  return $(b(s, a(s, p)))$;
  return $(X_2^{-1}(s, a(s, p)))$
  end
end X_1;
define $X_1^{-1}(s, p)$;
  select
  return $(a^{-1}(s, b^{-1}(s, p)))$;
  return $(a^{-1}(s, X_2(s, p)))$
  end
end X_1^{-1};
define $X_2(s, p)$;
  select
  return $(X_1^{-1}(s, b^{-1}(s, p)))$
  end
end X_2;
define $X_2^{-1}(s, p)$;
  select
  return $(b(s, X_1(s, p)))$
  end
end X_2^{-1}.

Then, for instance, for $\sigma = aabb$ and $X_1(\sigma, 0)$,

$$X_2^{-1}(\sigma, a(\sigma, 0)) = X_2^{-1}(\sigma, 1) \subseteq X_1(\sigma, 0);$$

$$X_2^{-1}(\sigma, 1) = b(\sigma, X_1(\sigma, 1));$$

$$b(\sigma, a(\sigma, 1)) = b(\sigma, 2) = \{3\} \subseteq X_1(\sigma, 1);$$

$$b(\sigma, 3) = \{4\},$$

and hence $4 \in X_1(\sigma, 0)$.

We shall extend the natural functional equivalence of the associated pattern functions to patterns themselves (i.e., to identifiers in the context of a pattern equation system) in the natural way.

DEFINITION 5. Let $\mathcal{S} = \mathcal{S}(C, I)$ be a pattern equation system and $X, Y \in I$. Then we say X is *equivalent to* Y in $\mathcal{S}$, $X \equiv_{\mathcal{S}} Y$, if for the associated functions in $\mathcal{F}_{\mathcal{S}}$ we have $X(s, p) = Y(s, p)$ for all $s \in C^*$ and $0 \leqq p \leqq \text{length}(s)$ (we omit the designation of $\mathcal{S}$ whenever no confusion will result). We also extend the equivalence concept to pattern expressions as follows: $\mathcal{E}_1 \equiv \mathcal{E}_2$ if for all pattern equation systems $\mathcal{S}$ which contain the equations $X = \mathcal{E}_1$ and $Y = \mathcal{E}_2$ we have $X \equiv_{\mathcal{S}} Y$.

Lastly we present the manner in which we take a pattern definition to relate to string matching (in the sense of SNOBOL4).

DEFINITION 6. Let $\mathcal{S} = \mathcal{S}(C, I)$ be a pattern equation system and $X \in I$. Then the *language associated with* X *in* $\mathcal{S}$, written $L_{\mathcal{S}}(X)$, is defined with the functions $\mathcal{F}_{\mathcal{S}}$ by $L_{\mathcal{S}}(X) = \{s \mid \text{length}(s) \in X(s, 0)\} \subseteq C^*$.

In the sense of SNOBOL4, $L_{\mathcal{S}}(X)$ is the set of strings which will successfully match X in fully anchored mode (i.e., anchored from both left and right). Unanchored matching would be successful for $C^* L_{\mathcal{S}}(X) C^*$. From a formal languages point of view there is little difference in the abstract properties of these two sets. We restrict our attention to the set $L_{\mathcal{S}}(X)$.

Before proceeding to the statement of results, there are a number of comments that we feel may be useful to the reader. We list these here.

Note 1. To this point the abstraction is quite faithful to SNOBOL4. That is, if $\mathcal{S}$ is a context-free pattern equation system, $L_{\mathcal{S}}(X)$ is usually exactly the collection of strings that will be successfully matched if one writes the same equations (i.e., assignments) in SNOBOL4; however all identifiers on the right-hand side must be written with the unevaluated expression operator to conform to the relational view (and the recursion) of our idealization. There are certain troublesome cases in the standard implementations of SNOBOL4 such as left-recursion and/or various quick-scan heuristics, but these problems are not inherent in the process and can be avoided with the use of other algorithms (e.g., see [20], [21]).

Note 2. The conventions of Definition 1 correspond to writing patterns in a sort of "normal form." That is, something such as $(a|aX)Y$ is not a valid pattern expression in these conventions. However, we will permit such an abuse of notation and assume that it represents a shorthand for the expression ZY, where Z is a new identifier, together with the augmentation of the pattern equation system under consideration by the equation $Z = a|aX$.

Note 3. The family of languages $L_{\mathcal{S}}(X)$ arising from the context-free pattern equation systems $\mathcal{S}$ of Definition 1 is precisely the family of context-free languages (see [10]). Also if the context-free system is one-sided linear, then the language is regular.

At the beginning of this section we pointed out that we were suggesting a model similar in nature to Gimpel's [12], [13] but which has essential constructability characteristics. In fact, as our first result shows, there are some technical differences which place our assumptions more in line with the algebraic version of Gimpel's model put forth by Stewart [25].

THEOREM 1. *Let P, Q and R be pattern expressions with reversal. Then*

(i) $(PQ)R \equiv P(QR)$,
(ii) $P|(Q|R) \equiv (P|Q)|R$,
(iii) $P|Q \equiv Q|P$,
(iv) $P(Q|R) \equiv PQ|PR$,
(v) $(P|Q)R \equiv PR|QR$,
(vi) $P\Lambda \equiv \Lambda P \equiv P$,
(vii) $P\Phi \equiv \Phi P \equiv \Phi$,
(viii) $\Phi^{-1} \equiv \Phi$,
(ix) $\Lambda^{-1} \equiv \Lambda$,
(x) $(PQ)^{-1} \equiv Q^{-1}P^{-1}$,
(xi) $(P|Q)^{-1} \equiv P^{-1}|Q^{-1}$,
(xii) $(P^{-1})^{-1} \equiv P$.

We provide only the proof of selected parts, as they are representative of the organization of details in the other parts.

Proof of (iv). This part is one of the properties which distinguishes this model from Gimpel's.

We first must express the identity in terms of a proper system in the way suggested in Note 2 above. Thus we assume that we are considering a system $\mathcal{S}$ which has U, V, W, X, Y, $Z \in I$ and includes the equations

$$X = P, \quad Y = Q, \quad Z = R,$$

$$U = Y|Z, \quad V = XU, \quad W = XY|XZ,$$

and we wish to show that $V \equiv_{\mathcal{S}} W$.

The procedures associated with $\mathcal{S}$ will include

define $X(s, p)$;
  code (P)
end X;

define $Y(s, p)$;
  code (Q)
end Y;

define $Z(s, p)$;
  code (R)
end Z;

define $U(s, p)$;
  select
  return $(Y(s, p))$;
  return $(Z(s, p))$
  end
end U;

define $V(s, p)$;
  return $(U(s, X(s, p)))$
end V;

define $W(s, p)$;
  select
  return $(Y(s, X(s, p)))$;
  return $(Z(s, X(s, p)))$
  end
end W.

Hence if $q \in V(s, p)$, then there exists $p' \in X(s, p)$ and $q \in U(s, p')$. But if $q \in U(s, p')$, then either $q \in Y(s, p')$ or $q \in Z(s, p')$ and so in either case $q \in W(s, p)$. Similarly for the argument in the converse direction. Hence $V \equiv W$.

We consider now the details for the identities explicitly involving reversal. Parts (viii) and (ix) are directly due to the definitions.

Proof of (x): We suppose a system $\mathcal{S}$ which includes the equations

$$V = P, \quad W = Q, \quad X = VW,$$

$$Y = X^{-1}, \quad Z = W^{-1}V^{-1}.$$

Then we wish to show that $Y(s, p) = Z(s, p)$ for all $s \in C^*$ and $0 \leqq p \leqq \text{length}(s)$. For such a system we have the function definitions:

define $Y(s, p)$;
  select
  return $(X^{-1}(s, p))$
  end
end Y;

define $Z(s, p)$;
  select
  return $(V^{-1}(s, W^{-1}(s, p)))$
  end
end Z;

define $X^{-1}(s, p)$;
  select
  return $(V^{-1}(s, W^{-1}(s, p)))$
  end
end X^{-1};

and so clearly the result follows.

Proof of (xi). We suppose a system $\mathcal{S}$ which includes the equations

$$V = P, \quad W = Q, \quad X = V|W, \quad Y = X^{-1}, \quad Z = V^{-1}|W^{-1}.$$

Then a direct application of the definition of the code function shows that $Y \equiv_{\mathcal{S}} Z$.

Proof of (xii). We suppose a system $\mathcal{S}$ which includes the equations

$$X = P, \quad Y = X^{-1}, \quad Z = Y^{-1}.$$

For such a system we would have the function definitions

define $X(s, p)$;
  select
  code (P)
  end
end X,
define $Y^{-1}(s, p)$;
  select
  return $(X(s, p))$
  end
end Y^{-1},
define $Z(s, p)$;
  select
  return $(Y^{-1}(s, p))$
  end
end Z.

So again we have $X \equiv_{\mathscr{P}} Z$, and the result follows. □

The rules of pattern calculus given by parts (ix)–(xii) are strong motivation for our choice of notation for the reversal operation. We strongly prefer this means of specifying the direction of cursor movement and its clarity of expression over the suggestions in Doyle [8]. However, care must be taken not to regard reversal as a true inverse for concatenation, as our subsequent results will indicate.

As the above result illustrates, the operations we have defined behave according to a desirable (and familiar) calculus. The ability to specify only context-free pattern systems is much too restrictive to be used exclusively in practice. This has been overcome in SNOBOL4 by adding a great many functional elements which are allowed to occur as constituents. These numerous added elements serve to both expand the set of primitives (each with its own particular properties) and to defeat a systematic pattern calculus through special individual definition. The alternative we explore in this section is the addition of the single reversal operation which, as we have demonstrated, behaves nicely. Our next goal is to explore how this operation enhances the expressive power of pattern systems.

Example 5. *Let* $C = \{a, b\}$, $I = \{X\}$ *and consider the pattern system with the single equation* $X = b|ab$. *Then* $\Lambda \not\equiv XX^{-1} \not\equiv X^{-1}X \not\equiv \Lambda$. *This follows from the observations that* $XX^{-1}(ab, 0) = \{\theta, 0, 1\}$ *and* $X^{-1}X(ab, 0) = \{\theta\}$ *whereas* $\Lambda(s, p) = \{p\}$ *for all* $s \in C^*$ *and* $0 \leqq p \leqq \text{length}(s)$.

The above example clearly contrasts the reversal operation with a multiplicative (i.e., concatenation) inverse. It can be verified that if for a pattern expression P, $PP^{-1}(s, p) \not\subseteq \{\theta\}$, then $\Lambda(s, p) = \{p\} \subseteq PP^{-1}(s, p)$. A closer connection with the multiplicative identity cannot be made.

The principle working property for the reversal operation is captured in the next theorem.

THEOREM 2. *Let* P *be any pattern with reversal,* $s \in C^*$, m *and* n *be natural numbers with* $0 \leqq m, n \leqq \text{length}(s)$. *Then* $m \in P(s, n)$ *if and only if* $n \in P^{-1}(s, m)$.

Proof. We provide just the proof outline here. Clearly from property (xii) of Theorem 1, we need only show that $m \in P(s, n)$ implies $n \in P^{-1}(s, m)$. This is established by a structural induction argument as follows:

Anchor. If P is any of the atomic patterns (see Definition 1) a, a^{-1}, Λ or Φ (where $a \in C$), then the result is clearly true.

Induction. Suppose the result is true for all patterns defined with fewer than r occurrences of the operations of alternation, concatenation and reversal and let P be a pattern involving r occurrences.

Case 1. $P = Q^{-1}$. Follows by Theorem 1 (xii) and the induction assumption.

Case 2. $P = QR$. Then $m \in P(s, n)$ if and only if there exists k with $k \in Q(s, n)$ and $m \in R(s, k)$. But then by the induction assumption, $n \in Q^{-1}(s, k)$ and $k \in R^{-1}(s, m)$ and hence $n \in R^{-1}Q^{-1}(s, m)$. But by Theorem 1 (x), $R^{-1}Q^{-1} = P^{-1}$.

Case 3. $P = Q|R$. Then $m \in P(s, n)$ if and only if either $m \in Q(s, n)$ or $m \in R(s, n)$. But then, respectively, either $n \in Q^{-1}(s, m)$ or $n \in R^{-1}(s, m)$ by induction. But by Theorem 1 (xi), $P^{-1} = Q^{-1}|R^{-1}$ and hence $n \in P^{-1}(s, m)$ in either case.

Note that since recursive definitions are allowed, a formalization of this outline must be carried out by an induction on $|m - n|$. □

For the connection of the operations with the sets of strings described we have:

THEOREM 3. *Let P and Q be pattern expressions with reversal, occurring in the pattern equation system $\mathcal{S}$. Then*

(i) $L_{\mathcal{S}}(P|Q) = L_{\mathcal{S}}(P) \cup L_{\mathcal{S}}(Q)$,

(ii) $L_{\mathcal{S}}(P) \cdot L_{\mathcal{S}}(Q) \subseteq L_{\mathcal{S}}(PQ)$, *and*

(iii) $L_{\mathcal{S}}(P) \cdot L_{\mathcal{S}}(Q) = L_{\mathcal{S}}(PQ)$, *if $\mathcal{S}$ is context-free (i.e., does not contain reversal).*

Proof. We again omit details which follow directly from the definitions. But to establish that equality does not hold in part (ii), the reader can verify that for $P = abc$ and $Q = (bc)^{-1}bc|a$ we have $L(P) = \{abc\}$, $L(Q) = \{a\}$, and $L(PQ) = \{abc, abca\}$. □

The main reason for the addition of the reversal operation to patterns is to gain expressive power. This goal is indeed achieved and our next several results are intended to show the extent of the success.

We have earlier pointed out the connection between context-free patterns (i.e., without reversal) and context-free grammars. Because of this connection it is natural to categorize patterns in the same way one does grammars. In particular the facts that two-way finite automata accept no more languages than one-way finite automata and that these regular sets are exactly the languages defined by the right-linear grammars (see [24] for definitions and results) might lead one to conjecture that right-linear patterns augmented with reversal define only the regular sets. The observant reader will have noted that we have already presented a counterexample to this conjecture in Example 4 where $L(X) = \{a^n b^n | n \geqq 1\}$. In fact such systems can even define non-context-free languages as we point out in the next example.

Note that for pattern systems $\mathcal{S}(C, I)$ with reversal we say that the system is *right-linear* if each constituent term is of one of the forms Λ, Φ, α or $\alpha\beta$ and that the system is *linear* if it is of one of the forms $\Lambda, \Phi, \alpha, \alpha\beta, \beta\gamma$ or $\alpha\beta\gamma$, where $\alpha, \gamma \in C \cup C^{-1}$ and $\beta \in I \cup I^{-1}$.

Example 6. Let $C = \{a, b, c, d, e\}$ and $I = \{S, T, U, V, W, X, Y, Z\}$ and consider the pattern system with equations

$$\begin{aligned} S &= aT, & W &= b^{-1}W|a^{-1}X, \\ T &= bU^{-1}|cV, & X &= aY, \\ U &= e^{-1}T^{-1}, & Y &= bZ^{-1}|c, \\ V &= c^{-1}W, & Z &= d^{-1}Y^{-1}, \end{aligned}$$

While the details are tedious, it can be verified for this system that $L(S) = \{ab^n cd^n e^n | n \geqq 0\}$.

Note that if one does not insist on the right-linear restriction, the above system can be written more succinctly as

$$S = aT, \qquad T = bTe|cc^{-1}W,$$
$$W = b^{-1}W|a^{-1}aY, \qquad Y = bYd|c.$$

An equally important point we believe should be strongly made, namely, that often patterns which could be written without the use of reversal become more clear and more succinct when expressed using reversal. We illustrate with

Example 7. Let $C = \{\#, a, b, c\}$ and suppose one is interested in specification of the set of strings $L = \{\#X|X \in \{a, b, c\}^*$ and X contains at least one of each of the letters $a, b, c\}$.

We take $I = \{\sigma, \alpha, \beta, \gamma, R, U\}$ and define the system

$$\sigma = \#\alpha\beta\gamma U, \qquad \gamma = a\gamma|b\gamma|c,$$
$$\alpha = b\alpha|c\alpha|aR, \qquad U = \Lambda|aU|bU|cU,$$
$$\beta = a\beta|c\beta|bR, \qquad R = a^{-1}R|b^{-1}R|c^{-1}R|\#^{-1}\#.$$

The informal approach of this system is that each of the patterns continues to scan right until it finds an occurrence of one of the desired letters (α searches for a, β for b, and γ for c), failing if none is found and invoking R when it is; R "rewinds" the cursor position back to the marker $\#$ and the next search can then be made. Formally one can verify that length $(s) \in \sigma(s, 0)$ if and only if $s \in L$.

Now L is regular so that a right-linear pattern without reversal can be written. As is pointed out in [3], this would require a variable for each of the subsets of $\{a, b, c\}$. If one uses the full generality of context-free patterns (without reversal), a fairly succinct pattern can be written. But one need only expand the collection of search characters by one or two before the complexity of patterns without reversal become overly burdensome, while the scheme given above using reversal is simply expanded by one new equation for each additional search character; the correctness of the description remains apparent.

We now develop our last major result for patterns with reversal. This result will lead to a number of additional observations.

THEOREM 4. *A language $L \subseteq C^*$ is the language associated with a pattern equation system with reversal if and only if L is accepted by a two-way, nondeterministic pushdown automata (without endmarkers).*

Proof (outline). As proof we provide the constructions in both directions, but in the tradition of formal language theory we omit the proofs of the correctness of the constructions. This is compensated for in large part by the similarity of the constructions to those which are standard in the identification of ordinary context-free grammars and the ordinary (i.e., one-way) pushdown automata. We do not present the definitional details of two-way pushdown acceptors (without endmarkers). The conventions we follow are those of [14]. Our one departure is that we use empty stack acceptance (i.e., machine exists from right end of the input and stack is empty) in place of final state acceptance and thus require the luxury of null moves on input. This alternative convention for acceptance may be shown to agree with final state acceptance by techniques similar to those used for this demonstration in the usual one-way case.

First of all, suppose we are given a pattern definition with reversal $\mathcal{S}(C, I)$, and that $X \in I$. We construct a two-way pushdown automata which accepts $L(X)$ (by

empty stack) as follows:

(i) the input alphabet is C;

(ii) the stack alphabet is $C \cup I \cup C^{-1} \cup I^{-1}$ (i.e., C^{-1} is a set of symbols written a^{-1} for each $a \in C$ and similarly for I^{-1});

(iii) the initial stack symbol is X;

(iv) the state set is taken to consist of $\{q_0, q_1\}$ (q_0 is initial);

(v) the transition function δ is defined by:

(a) for each $Y \in I$, if t is a term other than Φ and Λ occurring in the equation for Y, then $(0, q_0, t) \in \delta(q_0, \lambda, Y)$ and $(0, q_0, t^{-1}) \in \delta(q_0, \lambda, Y^{-1})$ (where the sequence for t^{-1} is obtained as indicated in Definition 8);

(b) for each $Y \in I$, if Λ is a term occurring in the equation for Y, then $(0, q_0, \lambda) \in \delta(q_0, \lambda, Y)$ and $(0, q_0, \lambda) \in \delta(q_0, \lambda, Y^{-1})$;

(c) for each $a \in C$, $(1, q_0, \lambda) \in \delta(q_0, a, a)$;

(d) for each $a \in C$, $(-1, q_1, a^{-1}) \in \delta(q_0, \lambda, a^{-1})$;

(e) for each $a \in C$, $(0, q_0, \lambda) \in \delta(q_1, a, a^{-1})$.

Conversely suppose we are given a two-way pushdown acceptor A that accepts the language $L(A)$ by empty stack. The input alphabet of A is taken as the set C and the set I is taken to consist of all triples of the form $\langle s, \gamma, s' \rangle$, where s and s' are states of A, together with one additional abstract symbol, say α. Then the equations of the corresponding pattern equation system $\mathcal{S}(C, I)$ are as follows:

(i) $\alpha = \langle s_0, \gamma_0, s_0 \rangle | \langle s_0, \gamma_0, s_1 \rangle | \cdots | \langle s_0, \gamma_0, s_n \rangle$ where γ_0 is the initial stack symbol of A and $s_0, s_1, \cdots, s_n$ are its states (with s_0 initial);

(ii) for each $(1, s', \gamma_1 \cdots \gamma_k) \in \delta(s, a, \gamma)$, where $a \in C$ and γ_i is a stack symbol for $1 \leqq i \leqq k$ (and $k \geqq 0$), we include the following terms in the equation for $\langle s, \gamma, s'' \rangle$ for each state s'': for each sequence of states $t_1, \cdots, t_{k-1}$ the term $a\langle s', \gamma_1, t_1 \rangle \langle t_1, \gamma_2, t_2 \rangle \cdots \langle t_{k-1}, \gamma_k, s'' \rangle$ (note that for $k = 1$ this term is $a\langle s', \gamma_1, s'' \rangle$ and for $k = 0$ this term is a);

(iii) for each $(0, s', \gamma_1 \cdots \gamma_k) \in \delta(s, a, \gamma)$, where $a \in C$ and γ_i is a stack symbol for $1 \leqq i \leqq k$ ($k \geqq 0$), we include the following terms in the equation for $\langle s, \gamma, s'' \rangle$ for each state s'': for each sequence of states $t_1, \cdots, t_{k-1}$ the term $aa^{-1}\langle s', \gamma_1, t_1 \rangle \langle t_1, \gamma_2, t_2 \rangle \cdots \langle t_{k-1}, \gamma_k, s'' \rangle$;

(iv) for each $(-1, s', \gamma_1 \cdots \gamma_k) \in \delta(s, a, \gamma)$, where $a \in C$ and γ_i is a stack symbol for $1 \leqq i \leqq k$ ($k \geqq 0$), we include the following terms in the equation for $\langle s, \gamma, s'' \rangle$ for each state s'': for each sequence of states $t_1, \cdots, t_{k-1}$ the term $aa^{-1}(c_1^{-1} | c_2^{-1} | \cdots | c_n^{-1}) \langle s', \gamma_1, t_1 \rangle \langle t_1, \gamma_2, t_2 \rangle \cdots \langle t_{k-1}, \gamma_k, s'' \rangle$ (where $C = \{c_1, \cdots, c_n\}$);

(v) for each $(0, s', \gamma_1 \cdots \gamma_k) \in \delta(s, \lambda, \gamma)$, where γ_i is a stack symbol for $1 \leqq i \leqq k$ ($k \geqq 0$), we include the following terms in the equation for $\langle s, \gamma, s'' \rangle$ for each state s'': for each sequence of states $t_1, \cdots, t_{k-1}$ the term $\langle s', \gamma_1, t_1 \rangle \langle t_1, \gamma_2, t_2 \rangle \cdots \langle t_{k-1}, \gamma_k, s'' \rangle$ (note for $k = 1$ this term is $\langle s', \gamma_1, s'' \rangle$ and for $k = 0$ this term is Λ).

Then in the pattern equation system we have constructed, $L(\alpha)$ is the set accepted by A by empty stack. □

The two-way pushdown acceptors are known to accept a considerably broader class of languages than context-free, and so we have firm evidence of the strength of the extension provided by the cursor reversal operation. In fact, the association of the preceding theorem is sufficiently direct that techniques used with two-way pushdown acceptors may be translated fairly directly. Of course, simply following the construction given in the above proof is one way to accomplish the translation, but it is generally a very extravagant one. Usually if one uses the insight of the acceptor

approach and then develops the pattern, succinctness may be achieved as well. We illustrate with two examples.

Example 8. $C=\{a, b, c\}$ and $K=\{a^n b^n c^n \mid n \geqq 1\}$. This is one of the standard examples of a non-context-free language, and the technique for accepting K with a two-way pushdown acceptor is shown in [14]. This approach may be adapted to patterns with the result

$$W = aXbY, \qquad X = \Lambda | aXb,$$
$$Y = a^{-1}Z | b^{-1}Yc, \qquad Z = aZ | bZ | cc^{-1},$$

which has $L(W)=K$.

Example 9. $C=\{a, b\}$ and $K=\{ba^{2^n} \mid n \geqq 0\}$. This is another non-context-free language whose acceptance is illustrated in [14]. Again, that approach adapts directly to patterns with reversal, giving the succinct description

$$P = baQ, \quad Q = \Lambda | aa^{-1}RQ, \quad R = a^{-1}Raa | b^{-1}b,$$

which has $L(P)=K$.

It is of some significance that our identification is with two-way PDAs without endmarkers rather than those with endmarkers, as there are technical differences. For instance, with endmarkers one obtains closure under intersection while without endmarkers this is unknown. This is largely a technical rather than a substantive difference, however, as if we have $b \notin C$ and patterns P and Q over C, then for $R = bPbb^{-1}Q^{-1}b^{-1}U$, where U is a pattern which matches bC^*b, we have $L(R) = b(L(P) \cap L(Q))b$. The techniques of [14] may be adapted to deduce the closure of the family of languages defined by context-free patterns with reversal under union and inverse-gsm map and nonclosure under homomorphism. Also the absence of the endmarkers has no effect on the validity of the basic undecidability results: it is undecidable if the languages specified by these devices are empty, finite or infinite.

On the other hand there is the result of [1] that such languages can be accepted on a random access machine in time n^3. This implies that there is a pattern matching algorithm for this extended class of patterns whose worst case time exceeds by very little that which is required for only the context-free case. This is an important factor which makes reversal a very attractive extension.

Lastly we might note that most of the work on two-way pushdown acceptors has dealt with the model which includes endmarkers. As can be observed in the examples we presented here, endmarkers are often very handy. In fact the concept of endmarkers could be included in that of the pattern (in fact SNOBOL4 effectively has such elements—POS(0) and RPOS(0)), but this treatment is somewhat more clumsy and contributes no essentially different results, so we have preferred to omit them. For the development as it would be done with endmarkers, the reader can see [19].

2. Patterns augmented with complementation: The set system model. In this section we explore an alternative extension to the context-free pattern equation systems. We examine the extension provided by augmenting these patterns with a different additional operation: complementation. As is well known, the context-free languages are not closed under complementation, so we anticipate an increase in expressive power. However, considerable technical difficulty is encountered, and in fact we are forced to introduce a new model to carefully account for semantic details.

DEFINITION 7. A *pattern equation system with complementation*, $\mathcal{S}=\mathcal{S}(C, I)$, is defined as in Definition 1 with the sole exception that in part a) for a *factor* we allow any element of the form $C \cup I$ plus elements of the form $\neg X$, where $X \in I$.

While it is a commonly considered operation in formal languages, the use of complementation as a constituent in definition mechanisms has been considered relatively little previously (see [4], [18], [21]). Making use of the procedure model we introduced in the preceding section for semantics does not appear technically feasible. The difficulties arise because of the nondeterminism which occurs as an inherent aspect of that model. This trouble arises for classical models as well; for instance the context-free languages are accepted by nondeterministic push-down automata, and closure under complementation fails (i.e., languages defined with complementation cannot be expressed with the push-down model), and the context-sensitive languages are accepted by nondeterministic linear bounded automata; it is still not known if complementation can be incorporated in this model (i.e., the question of closure under complementation is open). In fact we have not found a way in which to express the desired semantics of complementation in the procedure model described earlier. Hence we abandon that approach to semantics and pursue an alternative.

Of course, as a set operation, complementation is easily understood, but things become more complicated when it can be used in a recursive way. Nevertheless, it is the basic set nature that leads us to explore the following approach to the precise semantics of complementation.

DEFINITION 8. Let disjoint nonempty, finite sets C (*alphabet*) and I (*identifiers*) be given. A *set equation system* $\mathcal{S} = \mathcal{S}(C, I)$ is defined hierarchically as follows:

a) a *factor* consists of any (constant) set $c \subseteq C^*$, identifier $X \in I$ or complemented identifier $\neg X$;
b) a *term* is any finite concatenation of one or more factors;
c) an *expression* is any finite union of one or more terms;
d) an *equation* is an ordered pair $(X, \mathscr{E})$ with $X \in I$ and $\mathscr{E}$ an expression, and is written $X = \mathscr{E}$;
e) a *set equation system* $\mathcal{S}$ consists of a finite collection of equations where each $X \in I$ occurs exactly once as a left element in the set.

If each of the constant sets is finite (or cofinite), the system is called *finitary*.

Example 10. Let $C = \{a, b\}$ and $I = \{X, Y\}$. Then

$$\mathcal{S} = \begin{Bmatrix} X = \{ab\} \cup \{a\}Y, \\ Y = \{b\} \cup X\{b\} \end{Bmatrix}$$

is a (finitary) set system.

DEFINITION 9. Let $\mathcal{S} = \mathcal{S}(C, I)$ be a set equation system where $I = \{X_1, \cdots, X_n\}$. A *solution* to $\mathcal{S}$ is an n-tuple $(\sigma_1, \cdots, \sigma_n)$, where $\sigma_i \subseteq C^* (1 \leqq i \leqq n)$, with the property that if, in $\mathcal{S}$, X_i is uniformly replaced by $\sigma_i (1 \leqq i \leqq n)$, then each of the equations is a valid set equality over C^*. For an identifier $X_i \in I$, we say σ_i is a *language* of X_i, written $L_{\mathcal{S}}(X_i) = \sigma_i$. If $\mathcal{S}$ has no solution it is said to be *inconsistent*; otherwise it is *consistent*. For instance, $(\{a^m b^m | m \geqq 1\}, \{a^m b^{m+1} | m \geqq 0\})$ is a solution of the system of Example 10.

There is a clearly intended isomorphism between pattern equation systems and set equation systems—pattern alternation, concatenation and complementation correspond, respectively, to set union, concatenation and complementation. The set of strings which successfully match a pattern is taken to be the language of the corresponding set equation system. Under this isomorphism, context-free pattern equation systems (or context-free grammars) correspond to finitary set systems not involving complementation (this relation is illustrated by Examples 1 and 10). It is well known that such set systems are consistent. One can define a partial order on solutions

$(\sigma_1, \cdots, \sigma_n) \leqq (\sigma'_1, \cdots, \sigma'_n)$ by requiring that $\sigma_i \subseteq \sigma'_i (1 \leqq i \leqq n)$. Then, while in general there may be many solutions, there is always a unique *smallest* solution. This gives precisely the language of the corresponding grammar or, as pointed out earlier, of the context-free pattern equation system. These facts are well known (see [5], [6], [24]), but this view is infrequently used in the formal languages literature. The addition of the complementation operator does not disturb the intuition of set equations, but it does greatly complicate other views (e.g., grammars, acceptors, etc.), and so we are urged towards set equations.

It may again be noticed that we impose some superficial restrictions on the form of set system equations so that the technical details can be developed more succinctly. However, there is no real loss of generality. For instance an equation such as $X = a \neg (aX)$ is not allowed (complementation can only be applied to identifiers), but one can simply add a new identifier, say Y, and write the equivalent system

$$X = a \neg Y, \qquad Y = aX.$$

When set equations systems are augmented to allow complementation (as in Definition 8), two new problems do arise, however. First of all, the systems may be inconsistent ($X = \neg X$ is an obvious example), and a system such as

$$X = \{a\} \cup Y, \qquad Y = \{b\} \cup \neg X$$

is just a little less obvious—more complicated instances can of course be given. Secondly, consistent systems still may have many solutions but need not any longer have a unique smallest solution. For instance,

Example 11.

$$X_1 = \neg X_2, \qquad X_2 = X_2$$

has solutions $(\varnothing, C^*)$ and $(C^*, \varnothing)$ but not $(\varnothing, \varnothing)$.

To illustrate that there does not appear to be any clear intuitive ground for choosing among several solutions, we include one more example.

Example 12. $C = \{a, b\}$,

$$X_1 = \neg X_1 \neg X_2, \qquad X_2 = X_2 \neg X_1$$

has solutions $(\varnothing, C^*)$ and $((a+b)^* b (a+b)^*, a^*)$ and possibly others. Motivation leading to a preference here seems missing.

In [20] and [21], Liu and Fleck show that if complementation is avoided on recursive definition chains in the systems, a consistent system must result. Moreover, they arrive at a preferred solution to such a system (as there still may be several) *via* a rewriting scheme approach. This restriction against using complement recursively, however, rules out some intuitively understandable systems which give very concise descriptions in some cases. Our goal here is specifically to explore systems where complementation is allowed to occur in a recursive way.

Example 13. $C = \{a, b\}$,

$$X = \neg X \{a, b\}.$$

$L(X) =$ all strings of odd length. This is quite easily surmised since clearly $\lambda \notin L(X)$ (as all strings in $L(X)$ end with either "a" or "b") and so $\lambda \in \neg L(X)$. But then $\{a, b\} \subseteq L(X)$. Hence $\neg L(X) \cap \{a, b\} = \varnothing$, etc. And this is a unique solution.

Example 14. $C = \{a, b\}$,

$$X_1 = \{a\} \cup \{a\} \neg X_2, \qquad X_2 = \{b\} \cup \{b\} \neg X_1$$

has a unique solution with $L(X_1) = (ab)^* a (\lambda + a(a+b)^*)$.

Example 15. $C = \{a, b\}$,

$$X = \neg X\{ab\}$$

has a unique solution, namely $L(X) = b^*aa^*b((a + bb^* + abb^*)aa^*b)^*$.

We will shortly present the techniques for dealing with such examples.

One of the motivations for the addition of the complementation operator was the increase in expressive power that could be achieved. In the preceding examples we have given an informal indication that in addition to greater expressive power we are able to achieve much more concise descriptions for languages that could be expressed without complementation. In fact formal results have been developed which show that the relative conciseness that can be achieved through the use of complementation is in fact remarkable [17], [26].

Our goal of allowing complementation to be used, possibly in a recursive way, as in the last examples, while avoiding inconsistent systems or those such as Examples 11 and 12, suggests that we seek conditions under which we can be sure that a unique solution exists. We next arrive at sufficient conditions for this desirable circumstance.

DEFINITION 10. Let $\mathcal{S} = \mathcal{S}(C, I)$ be a set equation system and $I = \{X_1, \cdots, X_n\}$. Then we define

$\Lambda_0 = \{X_i |$some term t in the equation for X_i is a product of constant sets and $\lambda \in t\}$

and

$\bar{\Lambda}_0 = \{X_i |$every term in the equation for X_i has as a factor a constant set c where $\lambda \notin c\}$,

and then inductively for $k \geqq 0$

$\Lambda_{k+1} = \Lambda_k \cup \{X_i |$some term in the equation for X_i consists of only factors which are either: 1) constant sets which include λ, or 2) $X_j \in \Lambda_k$, or 3) $\neg X_j$ where $X_j \in \bar{\Lambda}_k\}$

and

$\bar{\Lambda}_{k+1} = \bar{\Lambda}_k \cup \{X_i |$every term in the equation for X_i contains some factor which is either: 1) a constant set which does not include λ, or 2) $X_j \in \bar{\Lambda}_k$, or 3) $\neg X_j$ where $X_j \in \Lambda_k\}$.

We note that $\Lambda_0 \subseteq \Lambda_1 \subseteq \cdots$ and $\bar{\Lambda}_0 \subseteq \bar{\Lambda}_1 \subseteq \cdots$ and so we must have $\Lambda_n = \Lambda_m$ and $\bar{\Lambda}_n = \bar{\Lambda}_m$ for all $m > n$. Also of course $\Lambda_n \cap \bar{\Lambda}_n = \varnothing$. If $\Lambda_n \cup \bar{\Lambda}_n = I$ we shall say that $\mathcal{S}$ is *λ-determined.*

The following result is clear from the definitions and is presented without proof.

THEOREM 5. *Let $\mathcal{S} = \mathcal{S}(C, \{X_1, \cdots, X_n\})$ be a set equation system and $(\sigma_1, \cdots, \sigma_n)$ be any solution of $\mathcal{S}$. Then $X_i \in \Lambda_n$ implies $\lambda \in \sigma_i$ and $X_i \in \bar{\Lambda}_n$ implies $\lambda \notin \sigma_i (1 \leqq i \leqq n)$. Hence for a finitary, λ-determined system, $\lambda \in \sigma_i (1 \leqq i \leqq n)$ may be algorithmically determined.*

DEFINITION 11. *Let $\mathcal{S} = \mathcal{S}(C, I)$* be a λ-determined set equation system and $I = \{X_1, \cdots, X_n\}$. Then we define

$\pi_0 = \{X_i |$every nonconstant term in the equation for X_i either: 1) includes some factor which is a constant set that does not contain λ, or 2) includes at least two factors each of which is either $X_j \in \bar{\Lambda}_n$ or $\neg X_j$ with $X_j \in \Lambda_n\}$,

and then inductively for $k \geqq 0$,

$\pi_{k+1} = \pi_k \cup \{X_i |$every nonconstant term in the equation for X_i either: 1) includes some factor which is a constant set that does not contain λ, or 2) includes only factors which are either constant sets, X_j or $\neg X_j$ with $X_j \in \pi_k\}$.

Again of course $\pi_0 \subseteq \pi_1 \subseteq \cdots$ so that $\pi_n = \pi_m$ for all $m > n$. If $\pi_m = I$ and either $m = 0$ or $\pi_{m-1} \neq I$, then $\mathcal{S}$ is said to be *reductive* of degree m.

The next several results generalize some recent work by Leiss [18]. The systems of Leiss allow only a very restricted form of concatenation (i.e., one-sided concatenation of a single variable with a single constant set) and forbid the occurrence of the null string in constant sets. We impose neither of these restrictions, but note that the reductive property does put restrictions on the interactions of these phenomena.

THEOREM 6. *Each reductive set equation system has at most one solution.*

Proof. Suppose that $\mathcal{S} = \mathcal{S}(C, I)$ is the set equation system, where $I = \{X_1, \cdots, X_n\}$. We show that a solution, if one exists, must be unique. So we suppose that $(\sigma_1, \cdots, \sigma_n)$ and $(\sigma_1', \cdots, \sigma_n')$ are two solutions to $\mathcal{S}$. We show by induction on $|z|$ that $z \in \sigma_i$ if and only if $z \in \sigma_i' (1 \leqq i \leqq n)$.

Main anchor step ($|z| = 0$ or $|z| = 1$). First of all, $\lambda \in \sigma_i$ and $\lambda \in \sigma_i'$ if and only if $X_i \in \Lambda_n (1 \leqq i \leqq n)$ as $\mathcal{S}$ is λ-determined. Now for $a \in C$ suppose $a \in \sigma_i$. Then we show that $a \in \sigma_i'$ by another induction on j, the smallest index so that $X_i \in \pi_j$ (since $\mathcal{S}$ is reductive there is some such index).

Anchor step ($j = 0$). Since $a \in \sigma_i$, there is some term t in the equation for X_i so that $a \in t(\sigma_1, \cdots, \sigma_n)$ ($t(\sigma_1, \cdots, \sigma_n)$ denotes the set obtained by replacing each identifier X_i in t by the set σ_i, for $1 \leqq i \leqq n$). Now if t is constant then clearly $a \in \sigma_i'$. Otherwise t is a product of factors, say $t = f_1 f_2 \cdots f_p$, and as $X_i \in \pi_0$ there are two possibilities:

(1) Some factor, say $f_q (1 \leqq q \leqq p)$ is a constant and $\lambda \notin f_q$. But then for $a \in t(\sigma_1, \cdots, \sigma_n)$ we must have $a \in f_q$ and $\lambda \in f_k(\sigma_1, \cdots, \sigma_n)$ for $k \neq q (1 \leqq k \leqq p)$ and so $a \in t(\sigma_1', \cdots, \sigma_n') \subseteq \sigma_i'$.

(2) Two factors are variables in $\bar{\Lambda}_n$ or complemented variables from Λ_n. But this contradicts t denoting a string of length 1 and hence is in fact impossible.

Induction step ($j = j + 1$). Now we assume that for $a \in C$, $a \in \sigma_i$ if and only if $a \in \sigma_i'$ for all i such that $X_i \in \pi_j (j \geqq 0)$. Then suppose that $X_i \in \pi_{j+1} - \pi_j (j \geqq 0)$ and $a \in \sigma_i$. Hence for some term t in the equation for X_i, $a \in t(\sigma_1, \cdots, \sigma_n)$. Now if t is constant, then clearly $a \in \sigma_i'$. Otherwise t is a product of factors, say $t = f_1 f_2 \cdots f_p$, and as $X_i \in \pi_{j+1}$ there are two possibilities:

(1) Some factor, say $f_q (1 \leqq q \leqq p)$, is a constant and $\lambda \notin f_q$. But then since $a \in t(\sigma_1, \cdots, \sigma_n)$ we must have $a \in f_q$ and $\lambda \in f_k(\sigma_1, \cdots, \sigma_n)$ for $k \neq q (1 \leqq k \leqq p)$ and so $a \in \sigma_i'$.

(2) Each $f_q (1 \leqq q \leqq p)$ is either a constant set, or $X_k \in \pi_j$ or $\neg X_k$ with $X_k \in \pi_j$. Again we must have some $f_q (1 \leqq q \leqq p)$ with $a \in f_q(\sigma_1, \cdots, \sigma_n)$ while $\lambda \in f_r(\sigma_1, \cdots, \sigma_n)$ for all $r \neq q (1 \leqq r \leqq p)$. If f_q is constant, the argument is the same as in case (1) above. If $f_q = X_k \in \pi_j$, then $a \in f_q(\sigma_1, \cdots, \sigma_n) = \sigma_k$ and so by induction hypothesis $a \in \sigma_k'$ and hence $a \in f_q(\sigma_1', \cdots, \sigma_n')$. Also since $\mathcal{S}$ is λ-determined, $\lambda \in f_r(\sigma_1', \cdots, \sigma_n')$ for $r \neq q (1 \leqq r \leqq p)$ and hence $a \in t(\sigma_1', \cdots, \sigma_n') \subseteq \sigma_i'$. Similarly the argument follows if $f_q = \neg X_k$ where $X_k \in \pi_j$.

Thus we have established the anchor for our main induction on $|z|$.

Main induction step (note the need for the anchor for $|z| = 0, 1$). We now assume that $z \in \sigma_i$ if and only if $z \in \sigma_i' (1 \leqq i \leqq n)$ for all z with $|z| \leqq m$ (and $m \geqq 1$) and let $y \in \sigma_i$ with $|y| = m + 1$. We show that $y \in \sigma_i'$ by another induction on j, the smallest index so that $X_i \in \pi_j$.

Anchor step ($j = 0$). Since $y \in \sigma_i$, there is some term t in the equation for X_i so that $y \in t(\sigma_1, \cdots, \sigma_n)$. Now if t is constant, then clearly $y \in \sigma_i'$. Otherwise t is a product of factors, say $t = f_1 f_2 \cdots f_p$, and as $X_i \in \pi_0$ there are two possibilities:

(1) Some factor, say $f_q(1 \leqq q \leqq p)$, is a constant and $\lambda \notin f_q$. Then $y = y_1 y_2 \cdots y_p$, where $y_k \in f_k(\sigma_1, \cdots, \sigma_n)(1 \leqq k \leqq p)$ and $|y_q| > 0$. Hence $|y_k| \leqq m$ for $k \neq q(1 \leqq k \leqq p)$. Now each of the factors f_k for $k \neq q(1 \leqq k \leqq p)$ is either a constant, a variable X_r or a complemented variable $\neg X_r(1 \leqq r \leqq n)$. Hence by the main induction hypothesis for all k, $1 \leqq k \leqq p$, $y_k \in f_k(\sigma'_1, \cdots, \sigma'_n)$ and so $y \in t(\sigma'_1, \cdots, \sigma'_n) \subseteq \sigma'_i$.

(2) Two factors, say f_q and f_r, are either variables in $\bar{\Lambda}_n$ or complemented variables from Λ_n. Thus $y = y_1 y_2 \cdots y_p$, where $y_k \in f_k(\sigma_1, \cdots, \sigma_n)$ and $|y_q|, |y_r| > 0$. Hence $|y_k| \leqq m$ for all $1 \leqq k \leqq p$ and so, as with case (1), we can apply the main induction hypothesis to conclude that $y \in \sigma'_i$.

Induction step $(j = j+1)$. Finally assume that $y \in \sigma_i$ if and only if $y \in \sigma'_i$, for all i such that $X_i \in \pi_j$ and let $X_i \in \pi_{j+1} - \pi_j (j \geqq 0)$. Details here are similar to those given in cases above and so are omitted.

This then completes the main induction argument and establishes the uniqueness (but not the existence) of solutions to reductive systems. □

The other major concern is that a solution does indeed exist. To present this argument we will use the

Notation. For $\alpha \subseteq C^*$ and each $m \geqq 0$ we let $\alpha^m = \{w \in \alpha \mid |w| \leqq m\}$, and if $\mathscr{E}$ is an expression of a set equation system over variables $X_1, \cdots, X_n$ and $\alpha_1, \cdots, \alpha_n \subseteq C^*$, then $\mathscr{E}(\alpha_1, \cdots, \alpha_n)$ will denote the set obtained by replacing each occurrence of variable X_i by set α_i $(1 \leqq i \leqq n)$ and performing the indicated set operations.

LEMMA. *Let $\mathscr{S} = \{X_i = \mathscr{E}_i \mid 1 \leqq i \leqq n\}$ be a reductive set equation system and suppose that $\alpha_i \subseteq C^*(1 \leqq i \leqq n)$ are sets such that α_i includes all constant terms of $\mathscr{E}_i$ and $\lambda \in \alpha_i$ if and only if $X_i \in \Lambda_n$. Then for each $w \in \mathscr{E}_i(\alpha_1, \cdots, \alpha_n)$ with $|w| = m > 0 (1 \leqq i \leqq n)$ either:*

a) *w belongs to a constant term of $\mathscr{E}_i$, or*

b) *$X_i \in \pi_k (k \geqq 0)$ and w is formed by the concatenation of strictly shorter subwords which belong to constant sets, α_j or $\neg\alpha_j (1 \leqq j \leqq n)$; that is, $w \in \mathscr{E}_i(\alpha_1^{m-1}, \cdots, \alpha_n^{m-1})$, or*

c) *$X_i \in \pi_{k+1} (k \geqq 0)$ and $w \in \alpha_j$ or $w \in \neg\alpha_j$ for some $X_j \in \pi_k (1 \leqq j \leqq n)$.*

THEOREM 7. *Each reductive set equation system has a (unique) solution. Furthermore, if the system is finitary, then each component of the solution is a recursive set.*

Proof. Suppose that $\mathscr{S} = \{X_i = \mathscr{E}_i \mid 1 \leqq i \leqq n\}$. We construct a solution inductively by defining σ_i^j for $1 \leqq i \leqq n$ and $j = 0, 1, 2, \cdots$ where σ_i^j contains only strings of length at most j and the solution $(\sigma_1, \cdots, \sigma_n)$ is given by $\sigma_i = \bigcup_{j=0}^{\infty} \sigma_i^j$, for $1 \leqq i \leqq n$.

We first of all define

$$\sigma_i^0 = \begin{cases} \varnothing & \text{if } X_i \notin \Lambda_n, \\ \{\lambda\} & \text{if } X_i \in \Lambda_n. \end{cases}$$

Then assuming that $\sigma_i^j (j \geqq 0)$ is defined for all $1 \leqq i \leqq n$, we define σ_i^{j+1} by induction on k such that $X_i \in \pi_k$. For $X_i \in \pi_0$, $\sigma_i^{j+1} = (\mathscr{E}_i(\sigma_1^j, \cdots, \sigma_n^j))^{j+1}$ and for $X_i \in \pi_{k+1} (k \geqq 0)$, σ_i^{j+1} is defined to be all strings of length at most $j+1$ in the set obtained by the evaluation of $\mathscr{E}_i$ with respect to the sets: σ_m^{j+1} if $X_m \in \pi_k$ and σ_m^j if $X_m \notin \pi_k (1 \leqq m \leqq n)$.

The argument that $(\sigma_1, \cdots, \sigma_n)$ is in fact a solution is by induction on the length of strings. For strings of length zero, it is clear by Theorem 5 that all equations are satisfied. The induction step follows by a direct application of the lemma above.

To see that the component sets of the solution are recursive, just note that each of the steps in the construction of σ_i^j can be carried out algorithmically provided that the constant sets are finite (or even if they are just computable). □

Hence we have established, in terms of the readily determined sets Λ_n, $\bar{\Lambda}_n$ and $\pi_i (1 \leqq i \leqq n)$, a sufficient condition for the existence and uniqueness of a solution.

While this condition does not characterize those systems which possess a unique solution, we would like to point out its rather considerable generality. For the context-free case (i.e., no use of complementation) quite a number of special forms which result in unique solutions have been investigated. The uniqueness claims in [28, Thms. 2.1 and 2.3] can be deduced since it is easily seen that the reductive property is implied by the hypotheses. Similar comment holds for a number of the uniqueness results of [27, (e.g., Thms. 6.1, 9.1; Eqs. 7.1, 7.1', 7.4, 7.4', 7.5, 7.8]. Also the rather general condition in [24] of the absence of the so-called "empty word property" can be shown to imply the reductive property for the systems considered and hence the uniqueness result there [24, Thm. III.2, p. 121] can be deduced as well. It might also be noted for the context-free languages, if one takes the equations in Greibach normal form, one has a reductive system. The only work known to us which uses the complement operator in a system of equations is [4] and [18]. However, the restrictions assumed there lead to solutions which are always regular; moreover, it is easily verified that these same restrictions imply the reductive property, and so the uniqueness result presented there may also be viewed as following from our general result.

Having established some of the generality of our reductive property and having already made the point of the necessity of dealing with systems with unique solutions, we henceforth restrict attention to such systems. Another important aspect of our interests are those relating to computability. For the most part we will concern ourselves with the finitary systems. This could be relaxed somewhat by providing some constructive finite description for coefficient sets, but this becomes cumbersome and as we shall shortly see, adds no generality unless sets considerably more complex than context-free were to be admitted as coefficients.

Our next result is motivated by the connection between equation systems without complement and context-free grammars and the fact that the one-sided linear grammars characterize the regular sets. Since the regular sets are closed under complement, we might anticipate that the one-sided linear equation systems with complementation have solutions which are always regular. Recall that this same line of thought failed when applied to cursor reversal in the previous section. The problem is that the operation (complementation in this case) can be used recursively in the definitions. However, in this case the result holds and the proof provides the means to deal with several of the examples given earlier in this section.

Many of the transformation techniques for formal grammars apply without essential change to the systems we consider here. For instance, if in some system $\mathcal{S} = \mathcal{S}(C, I)$ a term of the form $X_j\{sa\}$ occurs, where $X_j \in I$, $s \in C^*$ and $a \in C$, then we can consider the system $\mathcal{S}' = \mathcal{S}'(C, I \cup \{Z\})$ which is obtained from $\mathcal{S}$ by replacing this term by $Z\{a\}$ and adding the equation $Z = X_j\{s\}$. Clearly, the solution sets of $\mathcal{S}'$ have not been perturbed from those of $\mathcal{S}$ (though of course the solutions for $\mathcal{S}'$ have an additional component corresponding to the new variable Z which does not occur in solutions to $\mathcal{S}$). We make implicit use of such techniques in proving the following:

THEOREM 8. *The solution sets for each one-sided linear, reductive set equation system (with complement) whose constant sets are finite are regular sets.*

Proof. We will assume that the system is given in left-linear form (the approach for right-linear systems is comparable), taking $I = \{X_1, \cdots, X_n\}$. We suppose $\mathcal{S}(C, I)$ consists of

$$X_i = f_0^i \cup X_1 f_1^i \cup \cdots \cup X_n f_n^i \cup \neg X_1 f_{n+1}^i \cup \cdots \cup \neg X_n f_{2n}^i$$

for $1 \leqq i \leqq n$, where each of the sets $f_j^i (0 \leqq j \leqq 2n)$ is finite. By the comments preceding the theorem we may assume without loss of generality that the sets $f_j^i (1 \leqq i \leqq n,$

$0 \leqq j \leqq 2n$) contain strings of length at most 1. For each solution set $\sigma_i (1 \leqq i \leqq n)$ we define a finite state acceptor A_i. The intuitive idea for the construction is that an accepting run for a string w should end in a state which consists of exactly the variables X_i for which w is in the solution σ_i. Let $A_i = (S, C, \delta, s_0, F_i)$ for $1 \leqq i \leqq n$, where

S (the states) $= \{s_0\} \cup \mathscr{P}(I)$ ($\mathscr{P}(I)$ is the collection of all subsets of I and s_0 is a new abstract element not in I);

$$F_i = \begin{cases} \{s \in \mathscr{P}(I) | X_i \in s\} & \text{if } X_i \in \bar{\Lambda}_n, \\ \{s_0\} \cup \{s \in \mathscr{P}(I) | X_i \in s\} & \text{if } X_i \in \Lambda_n, \end{cases}$$

and we define the next-state function δ, $\delta : S \times C \to S$ as follows: for $a \in C$ let

$$t(s_0, a) = \{X_p | a \in f_0^p\} \cup \{X_p | X_p \in \Lambda_n \text{ and } a \in f_q^p\} \cup \{X_p | X_p \in \bar{\Lambda}_n \text{ and } a \in f_{n+q}^p\}.$$

Then let $\delta(s_0, a)$ be the smallest set τ which contains $t(s_0, a)$ with the property that $X_p \in \tau$ and $\lambda \in f_q^p$ implies $X_q \in \tau$ and $X_p \notin \tau$ and $\lambda \in f_{n+p}^q$ implies $X_q \in \tau$.

For $s \in \mathscr{P}(I)$ and each $a \in C$ define $t(s, a) = \{X_p | \exists X_q \in s \text{ and } a \in f_q^p\} \cup \{X_p | \exists X_q \notin s \text{ and } a \in f_{n+q}^p\}$. Then let $\delta(s, a)$ be the smallest set τ which contains $t(s, a)$ and has the property that $X_p \in \tau$ and $\lambda \in f_p^q$ implies $X_q \in \tau$ and $X_p \notin \tau$ and $\lambda \in f_{n+p}^q$ implies $X_q \in \tau$. We omit the rather tedious argument of the correctness of A_i. It is another double induction with the main induction on the length of string and a subinduction on the smallest index of the containing π set, similar in many respects to the proof of the uniqueness theorem. □

COROLLARY. *Each one-sided linear, reductive set equation system (with complement) whose constant sets are regular has regular solution sets.*

Proof. Regard the system as a set equation system over an extended alphabet that includes the constant sets. By the preceding result, the solution sets to the system over this extended alphabet are regular. But the solution sets to the original system are clearly obtained by substitution of elements from the constant sets in place of those extended alphabet symbols, and the regular sets are closed under substitution. □

Our primary interest has been in developing a useful and general condition to guarantee unique solutions in systems utilizing complementation. Clearly this must be in the context of some restrictions on the constant sets allowed (e.g., for arbitrary $L \subseteq C^*$, $X = L$ uniquely "defines" L in a trivial way). The preceding result examines a certain combination of restrictions on the constant sets and the equation forms. As was suggested earlier, if one is to restrict only the nature of the constant sets, the finitary systems seem most natural to consider. We present a few results to relate the languages which can be expressed with finitary reductive systems to the usual formal languages hierarchy.

THEOREM 9. *The finitary reductive languages properly contain the context-free languages and are closed under union, product, complement and intersection.*

Proof. For a context-free language $K \subseteq C^*$, one takes a Greibach normal form grammar G with $K = L(G)$. Then as we have already mentioned the naturally corresponding set equation system $\mathscr{S}(G)$ is finitary and reductive. In fact it can be verified that $\Lambda_0 \cup \bar{\Lambda}_0 = I = \pi_0$.

The closure properties follow by the usual constructions, though there are some additional facts to be verified. We illustrate with just one of these results.

Let $\mathscr{S}_1$ and $\mathscr{S}_2$ be finitary reductive systems over disjoint sets of variables, say n_i variables in $\mathscr{S}_i (i = 1, 2)$. Also let X_i be a variable of $\mathscr{S}_i (i = 1, 2)$. A finitary reductive

system $\mathcal{S}_0$ so that $L_{\mathcal{S}_0}(X_0) = L_{\mathcal{S}_1}(X_1) \cup L_{\mathcal{S}_2}(X_2)$ is as follows:

the variables and equations of $\mathcal{S}_0$ are those of $\mathcal{S}_1$ together with those of $\mathcal{S}_2$ plus the new variables X_0 and the new equation $X_0 = X_1 \cup X_2$; it may then be verified that $X_0 \in \Lambda_{\max(n_1 n_2)+1}$ if $\lambda \in L_{\mathcal{S}_1}(X_1) \cup L_{\mathcal{S}_2}(X_2)$, and $X_0 \in \bar{\Lambda}_{\max(n_1,n_2)+1}$ otherwise, so $\mathcal{S}_0$ is λ-determined; also $X_0 \in \pi_{\max(n_1,n_2)+1}$ so $\mathcal{S}_0$ is reductive.

The other operations may be treated similarly and the details are omitted. □

This result also provides the justification for a very nice pattern calculus for the reductive context-free patterns augmented with complementation. The underlying set oriented semantics naturally provides for the usual commutative, associative, distributive and DeMorgan's laws. These are sufficiently familiar that they do not require restatement here. The fact that such constructions do not violate the reductive property makes this property a relatively workable restriction.

Also with respect to decision problems, most results are negative. Since the context-free grammars are included, all the problems which are undecidable for the context-free case are undecidable for the finitary reductive set equation systems. Moreover, the undecidability of equality with C^* for the context-free case and the closure of the finitary reductive systems under complementation implies the undecidability of the emptiness problem for these systems. Of course, as we pointed out previously, membership is decidable.

Conclusions. We have introduced two formal semantic models for string patterns. Each of these models may be adopted for the context-free patterns. Each model leads to the exploration of a significant extension to the context-free case. In both cases the nature of the extension is to provide much greater expressive power and moreover to add succinctness in many cases that could be treated without the extension. Each model must be regarded as successful in treating certain aspects which do not seem natural to the other model. It would seem desirable to explore both of the extensions (i.e., cursor reversal and complementation) in the context of a single model. This would allow the investigation of the interactions of the extensions and a determination of the effect of these interactions on expressive power and complexity. It is not apparent that either of the models presented here can be adapted to serve in this unifying role. Nonetheless we feel that these results demonstrate the potential for considerable benefit to be derived from the alternative approach of extending the context-free patterns by a few very general operations instead of by a great many rather special purpose ones. The results on expressive power are encouraging, and in the case of the procedure model, while an actual implementation would still require a mechanism for the efficient elimination of nondeterminism, Theorem 4 together with the results of [1] suggest that this is indeed feasible.

REFERENCES

[1] A. V. AHO, J. H. HOPCROFT AND J. D. ULLMAN, *Time and tape complexity of pushdown automaton languages*, Inform. and Control, 13 (1968), pp. 186–206.

[2] L. ALLISON, *Phrase structures, non-determinism and backtracking*, Inform. Process. Lett., 7 (1978), pp. 139–143.

[3] B. H. BARNES, *A two-way automaton with fewer states than any equivalent one-way automaton*, IEEE Trans. Comput., C-20 (1971), pp. 474–475.

[4] J. A. BRZOZOWSKI AND E. LEISS, *On equations for regular languages, finite automata and sequential networks*, Theoret. Comput. Sci., 10 (1980), pp. 19–35.

[5] N. CHOMSKY AND M. P. SCHÜTZENBERGER, *The algebraic theory of context-free languages*, in Computer Programming and Formal Systems, P. Braffort and D. Hirschberg, eds., North-Holland, Amsterdam, 1963.

[6] J. H. CONWAY, *Regular Algebra and Finite Machines*, Chapman and Hall, London, 1971.
[7] J. W. DE BAKKER, *Semantics and termination of nondeterministic recursive programs*, in Automata, Languages, Programming, Edinburgh Univ. Press, 1976, pp. 435–477.
[8] J. N. DOYLE, *A generalized facility for the analysis and synthesis of strings, and a procedure-based model of an implementation*, S4D48, Dept. of Computer Science, Univ. of Arizona, Tucson, AZ, 1975.
[9] A. C. FLECK, *Towards a theory of data stuctures*, J. Comput. System Sci., 5 (1971), pp. 475–488.
[10] ———, *Formal models for string patterns*, in Current Trends in Programming Methodology, Vol. IV: Data Structuring, R. Yeh, ed., Prentice-Hall, Englewood Cliffs, NJ, 1978.
[11] R. W. FLOYD, *Nondeterministic algorithms*, J. Assoc. Comput. Mach., 14 (1967), pp. 636–644.
[12] J. F. GIMPEL, *A theory of discrete patterns and their implementation in* SNOBOL4, Comm. ACM, 16 (1973), pp. 91–100.
[13] ———, *Algorithms in* SNOBOL4, John Wiley, New York, 1976.
[14] J. N. GRAY, M. A. HARRISON AND O. H. IBARRA, *Two-way pushdown automata*, Inform. and Control, 11 (1967), pp. 30–70.
[15] R. E. GRISWOLD, *Extensible pattern matching in* SNOBOL4, Proc. ACM Annual Conf., Minneapolis, MN, 1975, pp. 248–252.
[16] R. E. GRISWOLD, J. F. POAGE AND I. P. POLONSKY, *The* SNOBOL4 *Programming Language*, Prentice-Hall, Englewood Cliffs, NJ, 1971.
[17] J. HARTMANIS, *On the succinctness of different representations of languages*, in Automata, Languages and Programming, Lecture Notes in Computer Science, 71, Springer-Verlag, New York, 1979.
[18] E. LEISS, *On generalized language equations*, Theoret. Comput. Sci., 14 (1981), pp. 63–77.
[19] R. S. LIMAYE, *Analysis of string patterns using a procedure-type model and formal languages*, Doctoral dissertation, Univ. of Iowa, Iowa City, 1978.
[20] K. C. LIU, *On string pattern matching: A new model with a polynomial time algorithm*, this Journal, 10 (1981), pp. 118–140.
[21] K. C. LIU AND A. C. FLECK, *String pattern matching in polynomial time*, 6th Annual ACM Symposium on Principles of Programming Languages, San Antonio, TX, 1979, pp. 222–225.
[22] M. O. RABIN AND D. SCOTT, *Finite automata and their decision problems*, IBM J. Res. Develop., 3 (1959), pp. 114–125.
[23] D. G. RIPLEY AND R. E. GRISWOLD, *The measurement of* SNOBOL4 *programs*, SIGPLAN Notices, 10 (1975), pp. 36–53.
[24] A. SALOMAA, *Theory of Automata*, Pergamon Press, New York, 1969.
[25] G. F. STEWART, *An algebraic model for string patterns*, in Second ACM Symposium on Principles of Programming Languages, Palo Alto, CA, 1975.
[26] L. J. STOCKMEYER, *The complexity of decision problems in automata theory and logic*, Tech. Rep. MAC TR-133, Project MAC, Massachusetts Institute of Technology, Cambridge, MA, 1974.
[27] T. URPONEN, *On axiom systems for regular expressions and on equations involving languages*, Ann. Univ. Turku, Ser. A I, 145 (1971), pp. 1–51.
[28] ———, *Equations with a Dyck language solution*, Inform. and Control, 30 (1976), pp. 21–37.

SIAM J. COMPUT.
Vol. 12, No. 1, February 1983

0097-5397/83/1201-0012 $01.25

ON PROVING UNIFORM TERMINATION AND RESTRICTED TERMINATION OF REWRITING SYSTEMS*

J.V. GUTTAG,† D. KAPUR,‡ AND D.R. MUSSER‡

Abstract. In mechanical theorem proving, particularly in proving properties of algebraically specified data types, we frequently need a decision procedure for the theory of a given finite set of equations (axioms). A general approach to this problem is to try to derive from the axioms a set of rewrite rules that are "canonical," i.e., they rewrite to a canonical form all terms that are equal (according the axioms and the equivalence and substitution properties of equality). Rewrite rules are canonical if and only if they determine a relation that is both confluent and uniformly terminating. The difficulty of proving uniform termination has been the major drawback of the rewrite rule approach to deciding equations.

A new method of proving uniform termination is proposed. Assuming that the rewriting relation is globally finite (for any term there are only finitely many terms to which it can be rewritten), nontermination can occur only if there are cycles. Uniform termination is proved by showing that no cycles can occur. A method related to the Knuth and Bendix method of proving confluence is developed and used as the basis of such proof. In most cases, the proposed method will only prove termination for terms up to a certain size; this kind of "restricted termination" has a number of applications.

Key words. uniform termination, restricted termination, global finiteness, rewrite rules, confluence, Knuth-Bendix algorithm, overlap closure, canonical, rewrite dominoes, equational axioms, theorem proving

1. Introduction.

Term rewriting systems, also called (sets of) rewrite rules, are a model of computation that has the interesting and useful property of being directly applicable to obtaining decision procedures for equational theories. A term rewriting system is said to be *uniformly terminating*[1] if for every term, every sequence of rewrites starting from that term is of finite length. This property corresponds to the uniform halting property of Turing machines, a fact which Huet and Lankford (1977) used to demonstrate the undecidability of uniform termination.

In one of the main applications of term rewriting systems, the Knuth-Bendix (1970) approach to obtaining equational decision procedures, we start with a set of equations and attempt to derive from them a set of rewrite rules with the uniform termination property, as well as the property of confluence (for any term all sequences of rewrites emanating from it are extendable to a common term). If this can be done, the rules compute a canonical form for each class of terms that are equivalent under the original equations. Having such a canonical form yields an efficient decision procedure for the theory of the original set of equations.

Another application of rewrite rules is to produce "direct implementations" of abstract data types (Guttag, Horowitz, and Musser, (1978)). Such implementations are generally not efficient enough to be used in production programs, but can be helpful during the design of new data types and of programs that use the data types. Direct implementations are guaranteed to terminate if and only if the rewrite rules have the uniform termination property.

*Received by the editors December 18, 1981, and in revised form June 11, 1982. This paper was typeset at the General Electric Research and Development Center using the *Troff* software developed for the Unix operating system.

†Laboratory for Computer Science, Masachusetts Institute of Technology, Cambridge, Massachusetts 02139.

‡General Electric Company, Corporate Research and Development, Schenectady, New York 12301.

[1]More commonly called "finitely terminating" or "Noetherian."

Although undecidable in general, the uniform termination property can be proved for particular term rewriting systems in a variety of ways, mostly based on the mapping of terms into a well founded partially ordered set (see Huet and Oppen (1980) for a survey). An unfortunate problem with attempting to prove uniform termination this way is that, for rule sets of any significant size, an appropriate mapping is often very difficult to construct — it may require a great deal of ingenuity. In this paper, we present a new approach to proving termination, which, though not as general as the previous methods, is more algorithmic. This method will usually not yield a complete proof of uniform termination by itself. However, it can be used for proving restricted termination (i.e., termination of a finite set of terms), which has applications as discussed later in the paper. The method can possibly be used to simplify the application of other methods of proving uniform termination.

When the original rules are not uniformly terminating, one would often like to be able to detect this situation quickly, e.g., in order to avoid wasting time attempting to construct a proof of uniform termination. Under some reasonable restrictions on the form of rewrite rules, our approach provides such a test. That is, we show that if the rules are globally finite (that is to say, the number of different terms to which any term can be rewritten is finite) and every rule is right-linear or every rule is left-linear, our method can be used to effectively search for cycles in the rewriting relation.

In an effort to make this paper self-contained, we devote § 2 to a short tutorial on term rewriting systems. The reader familiar with the literature in this area will need only to skim this section. We describe the basic mechanism of term rewriting systems, and state and prove a well known result relative canonicity to uniform termination and confluence. We also relate the property of global finiteness to uniform termination; in particular, we show that if a reduction relation is globally finite and acyclic, it is uniformly terminating.

In § 3 we address the problem of showing that a rewriting relation is globally finite. We first prove that global finiteness is undecidable, and then develop some syntactic conditions sufficient to ensure global finiteness.

In §§ 4 and 5, we address the problem of showing that a rewriting relation is acyclic. The main problem is the size of the space of terms that must be traversed in searching for cycles. Our main results show that the search space can be cut down significantly. We show how to construct, from a set of rules **R**, another set of rules we call the *overlap closure* of **R**, with the property that if a reflexive rule is contained in it, then the rewriting relation of **R** has a cycle. The overlap closure corresponds to a subset of the transitive closure of the rewriting relation.

Our main theorem here states that a partial converse holds, so that generation of the overlap closure and looking for reflexive rules is sufficient to detect a cycle if one exists. In proving this theorem in § 5, we develop several lemmas of independent interest and a new way representing rewrite rules and sequences of rewrites using what we call *rewrite dominoes* and "rewrite domino layouts." We will introduce this representation and use it in presenting the proofs of our main results about the overlap closure. We believe that this representation also will be useful in the study of other areas of rewrite rule theory.

The generation of the overlap closure is very similar to the way the Knuth-Bendix process generates rules in attempting to produce a confluent rewriting relation. Its construction is based on the use of *derived pairs* of terms obtained from superpositions of the right hand side of one rule with the left hand side of another.

This is in contrast to the Knuth-Bendix process, which uses *critical pairs* obtained from superpositions of the left hand sides of the rules.

Like the Knuth-Bendix process, the overlap closure process may fail to terminate (that is, it may continue to generate new rules indefinitely). In fact, when the original rules are uniformly terminating, it will usually happen that overlap closure generation is nonterminating. In this case, the overlap closure process does not by itself yield a proof of uniform termination (see Huet and Oppen (1980)). As we show in § 6, apart from showing nontermination, partial generation of the overlap closure is useful in some applications where it suffices to have a proof of "restricted termination" — in this case termination for all "small" terms.

The overlap closure construction is more general than the forward chain construction discussed by Dershowitz (1981). As discussed in § 5, the overlap closure can be used in proofs of termination of both right-linear and left-linear rewriting systems, whereas forward chains require an additional strong assumption in the case of left-linear rewriting systems.

2. Definitions, notation, and basic theory.

2.1 Term rewriting systems. We begin with a definition of "terms." We assume a denumerably infinite set of distinguishable symbols called *variable symbols,* and a disjoint finite set of distinguishable symbols called *function symbols.* A *term* is defined inductively as either (1) a variable symbol, or (2) a function symbol followed by a finite sequence of terms. In the latter case, if f is the function symbol and $t_1, \cdots, t_n$ is the sequence of terms, the term is denoted $f(t_1, \cdots, t_n)$ and the t_i are called the *arguments* of the term. The number of arguments, n, is called the *arity* (nullary, unary, binary, etc.) of the function symbol. A constant term is written as $f()$ and binary terms will sometimes be written in infix notation, e.g., $(x+y) \bullet z$ for $\bullet(+(x,y),z)$.

The *subterms* of a term are the term itself and the subterms of its arguments. Like a variable, a term of the form $f()$ has no subterms other than itself. A *subterm position* and *corresponding subterm* within a term is a finite sequence of nonnegative integers separated by "." and a related term determined as follows: to the null sequence (denoted $<>$) corresponds the entire term. If $f(t_1, \cdots, t_n)$ is the subterm at position i then the subterm at position $i.j$ is t_j.

For example, the subterm positions and corresponding subterms within $f(x,g(y,k(z)),h())$ are:

$<>$	$f(x,g(y,k(z)),h())$
1	x
2	$g(y,k(z))$
2.1	y
2.2	$k(z)$
2.2.1	z
3	$h()$

We write $t[i]$ for the subterm at position i within term t.

A *rewrite rule* is an ordered pair of terms (l,r) such that every variable that occurs in r occurs also in l. We usually denote the rule as $l \rightarrow r$. A *term rewriting system* is a set (usually finite) of such rules. This is all there is to the syntax of term rewriting systems; they also have a very simple semantics, to which we now turn.

Two terms are *identical* if (1) they are identical variables, or (2) they have identical function symbols and their sequences of arguments are identical. We write $t_1 == t_2$ for this relation.

A *substitution* is a mapping θ from variable names to terms such that $\theta(v) == v$ for all but a finite number of variable symbols. It is denoted by an expression of the form $[t_1/v_1, \cdots, t_n/v_k]$, where the $k \geqslant 0$ variable symbols $v_1, \cdots, v_k$ are distinct. (The case $k=0$ is the identity substitution.) The domain of a substitution θ is extended to the set of all terms by inductively defining $\theta(f(t_1, \cdots, t_n))$ to be $f(\theta(t_1), \cdots, \theta(t_n))$.

A set of rewrite rules, **R**, generates a binary relation "$\rightarrow$" on the set of all terms, called the *rewriting* relation, as follows:

1. For every rule (l, r) in **R**, and every substitution θ, $\theta(l) \rightarrow \theta(r)$ holds.
2. If $t \rightarrow u$, then for every function symbol f and sequence of terms $t_1, \cdots, t_i, \cdots, t_n$ with $t_i = t$ for some i, $f(t_1, \cdots, t, \cdots, t_n) \rightarrow f(t_1, \cdots, u, \cdots, t_n)$ also holds.

Whenever $t \rightarrow u$, we say that "t rewrites directly to u (using **R**)." An equivalent way of defining the direct rewriting relation is as follows:

We say that t_2 *has the form of* t_1 if there is a substitution θ such that $\theta(t_1) == t_2$; also we say that t_2 is *matched* by t_1. Then t rewrites directly to u (using **R**) if and only if there is a subterm position i such that $t[i]$ is matched by the left-hand side l of some rule $l \rightarrow r$ of **R**, say $\theta(l) == t[i]$, and u can be obtained from t by replacing $t[i]$ by $\theta(r)$ at position i. We write this as $[t$ with $\theta(r)$ at $i]$.

2.2. Reduction relations. As in Huet (1980), we now develop a number of concepts at a more abstract level than term rewriting systems: we assume a set E and a binary relation "$\rightarrow$" on E, called "reduction." When definitions or lemmas depend on the set E being a set of terms and the reduction relation being the rewriting relation generated by a set of rewrite rules, we make explicit note of this fact.

The reflexive and irreflexive transitive closures of $\rightarrow$ are denoted $\rightarrow^*$ and $\rightarrow^+$, respectively. Thus $A \rightarrow^* B$ if and only if there are elements $A_0, \cdots, A_n$, $n \geqslant 0$, such that $A = A_0$, $A_n = B$ and $A_0 \rightarrow A_1 \rightarrow \cdots \rightarrow A_n$; for $A \rightarrow^+ B$ we require $n \geqslant 1$. The sequence $A_0 \rightarrow \cdots \rightarrow A_n$ is called a reduction sequence from A_0 to A_n.

Let $\leftrightarrow$ be the relation defined by $A \leftrightarrow B$ if and only if $A \rightarrow B$ or $B \rightarrow A$. The reflexive transitive closure of $\leftrightarrow$, denoted $\leftrightarrow^*$, is the abstract version of "equality" (it is an equivalence relation on E; and in the case of a term rewriting system, it is moreover an equality relation since the substitution property holds). In order to deal with $\leftrightarrow^*$ in terms of $\rightarrow^*$, we will define and use canonical forms in E.

First, we define an element P of E to be *terminal* if there is no element Q such that $P \rightarrow Q$. If $A \rightarrow^* B$ and B is terminal, then we call B a *terminal form of* A. An element can have many distinct terminal forms. A *nonterminating reduction sequence from* A is an infinite sequence $A_0 \rightarrow A_1 \rightarrow \cdots$ with $A_0 = A$. The reduction relation $\rightarrow$ is said to be *terminating for* A if there is no nonterminating reduction sequence from A, and $\rightarrow$ is said to be *uniformly terminating* if it is terminating for every element of E.

We say any B and C are *joinable* if there exists a D such that $B \rightarrow^* D$ and $C \rightarrow^* D$. The relation $\rightarrow$ is *confluent from* A if for every B and C such that $A \rightarrow^* B$ and $A \rightarrow^* C$, B and C are joinable. We say $\rightarrow$ is *uniformly confluent* if it is confluent from every element of E.

LEMMA 2.2.1 (well known). *If* $\rightarrow$ *is uniformly confluent, the terminal form of any element, if it exists, is unique.*

Proof. Let $A \rightarrow^* B$ and $A \rightarrow^* C$, with both B and C terminal. By confluence from A, there must be a D such that $B \rightarrow^* D$ and $C \rightarrow^* D$, but since B and C are terminal, we must have $B = D = C$. □

Let $\equiv$ be an equivalence relation on E. We say that $\rightarrow$ is *canonical with respect to* $\equiv$ if it is uniformly terminating and any two elements have the same terminal form if and only if they belong to the same $\equiv$ — equivalence class. If $\rightarrow$ is canonical with respect to $\leftrightarrow^*$, we simply say that it is *canonical.*

THEOREM 2.2.2 (well known). *A reduction relation $\rightarrow$ is canonical if and only if it is both uniformly terminating and uniformly confluent.*

Proof. First, suppose $\rightarrow$ is canonical; then by definition it is uniformly terminating; we have to show that it is uniformly confluent. Let A be any element and suppose $A \rightarrow^* B$ and $A \rightarrow^* C$; then $B \leftrightarrow^* C$ and, since $\rightarrow$ is canonical, B and C have the same terminal form. Thus $\rightarrow$ is confluent from A, for all A.

In the other direction, suppose $\rightarrow$ is uniformly terminating and uniformly confluent. Let A and B be any elements such that $A \leftrightarrow^* B$; we have to show that A and B have the same terminal form. By definition of $\leftrightarrow^*$, there are elements $A_0, \cdots, A_n$, $n \geqslant 0$, such that $A = A_0$, $A_n = B$ and for $0 \leqslant i \leqslant n-1$ either $A_i \rightarrow A_{i+1}$ or $A_{i+i} \rightarrow A_i$. The proof is by induction on n. If $n = 0$, then $A = B$ and, by confluence and Lemma 2.2.1, they have the same terminal form. If $n > 0$, then by the induction hypothesis, A_0 and A_{n-1} have the same terminal form, T, say. If $A_{n-1} \rightarrow A_n$, then by confluence from A_{n-1}, T is also the terminal form of A_n. The other case is $A_n \rightarrow A_{n-1}$: by confluence from A_n and Lemma 2.2.1, T is also the terminal form of A_n. □

In the case of a term rewriting relation, Knuth and Bendix (1970) showed how to perform the test for uniform confluence by examining how the left-hand sides of the rules "overlap" each other, producing "critical pairs" of terms to be checked for confluence. Out approach to proving uniform termination relies on a generalized notion of overlapping, yielding "derived pairs." This will be discussed in § 4.1.

2.3. Relating uniform termination to global finiteness and acyclicity. A reduction relation $\rightarrow$ is *locally finite* if for every A in E, the set of elements B such that $A \rightarrow B$ is finite; and $\rightarrow$ is *globally finite* if the set of B such that $A \rightarrow^* B$ is finite.

LEMMA 2.3.1. *If a reduction relation is globally finite and acyclic, it is uniformly terminating.*

Proof. Any nonterminating reduction sequence would either have to repeat some element (hence be a cycle) or contain infinitely many distinct elements (hence be globally infinite). Thus all reduction sequences have to terminate. □

The converse does not hold in general but does hold for locally finite reduction relations.

THEOREM 2.3.2. *A locally finite reduction relation is uniformly terminating if and only if it is both globally finite and acyclic.*

Proof. The if part is immediate from the lemma. Suppose the relation is not globally finite or is cyclic. In the latter case, it is obviously not uniformly terminating. In the former case, since the the relation is assumed to be locally finite, we can apply Koenig's lemma to some element which has infinitely many decendants, concluding that there must be an infinite path of reductions from that element; thus the relation is not uniformly terminating. □

Since it is easily shown that the rewriting relation of a finite set of rules is locally finite, we have the following.

COROLLARY 2.3.3. *The rewriting relation of a finite set of rules is uniformly terminating if and only if it is both globally finite and acyclic.* □

In the next section we investigate methods of proving global finiteness, and in §§ 4 and 5 methods for proving that a rewriting relation is acyclic.

3. Proving global finiteness. In general, we cannot decide uniform termination algorithmically, as the following result of Huet and Lankford (Huet and Lankford (1977, Thm. 1)) shows.

THEOREM 3.1. *The uniform termination problem for rewrite rule systems is undecidable, even for terms restricted to unary and nullary function symbols.*

From this result we can show that the question of global finiteness for rewrite rule systems is also undecidable.

THEOREM 3.2. *There is no decision procedure for global finiteness of rewrite rule systems.*

Proof. Let $\mathbf{R} = \{l_i \rightarrow r_i\}$ be a finite set of rewrite rules in which all terms l_i and r_i are built from a finite set $\{f_1, \cdots, f_k\} \cup \{g_1, \cdots, g_m\}$ of function symbols, where each f_i is unary and each g_i is nullary, and a variable symbol x. Let h be a function symbol distinct from any of the f_i or g_i.

Construct a new set of rules

$$\mathbf{R}' = \{l_i \rightarrow h(r_i)\} \cup \{f_i(h(x)) \rightarrow h(f_i(x)) : 1 \leqslant i \leqslant k\}.$$

Then for any $t_0, t_1, \cdots, t_n$,

$$t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n \text{ (using } \mathbf{R}),$$

if and only if

$$t_0 \rightarrow h(t_1) \rightarrow \cdots \rightarrow h^n(t_n) \text{ (using } \mathbf{R}').$$

where h^n denotes the n-fold composition of h. The set $\mathbf{R}$ is uniformly terminating if and only if $\mathbf{R}'$ is globally finite (because if $\mathbf{R}$ is not globally finite, then $\mathbf{R}$ is not uniformly terminating). By the Huet and Lankford result, this means there can be no decision procedure for global finiteness. □

Thus we must be satisfied with finding conditions that are sufficient to guarantee global finiteness. We begin with a simple sufficient condition that we can show is syntactically checkable.

Define the *size* of a term t to be the number of function and variable symbols it contains. Denoting this $\text{Size}(t)$, we have $\text{Size}(f(g(x, f(x)))) = 5$, for example.

A rewrite rule, $t \rightarrow u$, is *nonexpanding* if for every substitution θ, $\text{Size}(\theta(t)) \geqslant \text{Size}(\theta(u))$. Otherwise, a rewrite rule is called *expanding.*

A rewrite rule system $\mathbf{R}$ is said to be nonexpanding if and only if every rule in $\mathbf{R}$ is nonexpanding; we also say that the rewriting relation of $\mathbf{R}$ is nonexpanding.

LEMMA 3.3. *If a rewriting relation $\rightarrow$ is nonexpanding, it is globally finite.*

Proof. Since the relation is nonexpanding, if $t \rightarrow^* u$ then $\text{Size}(u) \leqslant \text{Size}(t)$. Also the only variables that can occur in u are those in t. Thus there are only finitely many possibilities for u, and $\rightarrow$ must be globally finite. □

The next lemma and theorem justify our claim that the nonexpanding condition is syntactically checkable. We define $\text{Num}(v, t)$ to be the number of occurrences of the variable v in the term t.

LEMMA 3.4. *Let θ be the substitution $[t_1/v_1, \cdots, t_k/v_k]$. Then for any term t,*

$$\text{Size}(\theta(t)) = \text{Size}(t) + \sum_{j=1}^{k} \text{Num}(v_j, t)(\text{Size}(t_j) - 1).$$

Proof. With suitable inductive definitions of Size and Num, the lemma follows easily by induction on the structure of terms. □

THEOREM 3.5. *A rewrite rule, $t \rightarrow u$, is nonexpanding if and only if:*

1) $\text{Size}(t) \geqslant \text{Size}(u)$, *and*

2) *for every variable* v *in* t, $\text{Num}(v,t) \geqslant \text{Num}(v,u)$.

Proof. Let θ be any substitution $[t_1/v_1, \cdots, t_k/v_k]$. By Lemma 3.3,

(*) $\text{Size}(\theta(t)) - \text{Size}(\theta(u))$

$$= \text{Size}(t) - \text{Size}(u) + \sum_{j=1}^{k} (\text{Num}(v_j,t) - \text{Num}(v_j,u))(\text{Size}(t_j) - 1),$$

from which it is obvious that if 1) and 2) hold, then (*) is always nonnegative, implying that $t \to u$ is nonexpanding.

Now suppose 1) or 2) fails to hold. If 1) fails, the identity substitution would make (*) negative. Suppose 2) fails, i.e., there is a v for which $\text{Num}(v,t) < \text{Num}(v,u)$. Then again (*) can be made negative: taking $\theta = [s/v]$ for some term s, (*) simplifies to

$$\text{Size}(t) - \text{Size}(u) + (\text{Num}(v,t) - \text{Num}(v,u))(\text{Size}(s) - 1),$$

which becomes negative when $\text{Size}(s)$ is sufficiently large. Thus $t \to u$ fails to be nonexpanding. □

Requiring a rewriting relation to be nonexpanding is restrictive since there are useful expanding but globally finite relations. For example, a set of rules for canonicalizing propositional formulas to their disjunctive normal form would include the following distributive law which is expanding though the rewriting relation is globally finite.

$$x \cdot (y + z) \to x \cdot y + x \cdot z.$$

This restriction does have the advantage of requiring only strictly local, rule at a time, syntactic analysis. We have looked at relaxations of this restriction that preserve this locality property. An approach to relaxing this restriction so that rewrite rules using the if-then-else operator, which often arise in equational specifications of abstract data types, can be handled is discussed in Appendix A. The following two equations taken from a specification of the data type set specify one of its operations, *has?,* which tests for set membership.

$$\text{has?}(i,\text{null}) = \text{false},$$

$$\text{has?}(i,\text{insert}(i',s)) \to \text{if } i = i' \text{ then true else has?}(i,s).$$

The rule, expressed using the if-then-else operator, corresponding to the second equation is

$$\text{has?}(i,\text{insert}(i',s)) \to \text{if-then-else}(i = i',\text{true},\text{has?}(i,s)).$$

This approach can handle most rewriting systems that we have come across in specifying abstract data types (Musser (1980)).

4. Searching for cycles. We remind the reader of our basic approach to proving uniform termination: proving global finiteness, and proving there are no cycles in the rewriting relation. We have dealt with global finiteness in the previous section, and we now turn to the question of cycles. We assume we are given a term rewriting system $\mathbf{R} = \{l_i \to r_i\}$ whose rewriting relation is globally finite, and we wish to determine whether or not there is any cycle of terms

(*) $t_0 \to t_1 \to \cdots \to t_n \to t_0$

The following development of a method of searching for cycles is based on some generalizations of notions that Knuth and Bendix (1970) used in testing for uniform confluence.

4.1. Superpositions and derived pairs. The defintions of superpositions and derived pairs depends on the important concept of "unification."

Two terms t and u are said to be *unifiable* if there is a substitution θ such that $\theta(t) \equiv \theta(u)$. θ is called a *unifier* of t and u, and whenever θ is chosen so that it is a factor of any other unifier θ_1 (i.e., θ_1 can be written as a composition $\theta_2 \bullet \theta$ for some θ_2), then it is called a *most general unifier* (m.g.u.) of t and u. It can be shown that the m.g.u. of two terms, if it exists, is unique up to variable renaming.

Consider, for example, the terms: $t = f(x,g,(y))$ and $u = f(h(z),w)$ and the substitutions:

$\theta_1 = [h(g(u))/x, g(u)/z, g(y)/w]$ and

$\theta_2 = [h(z)/x, g(y)/w]$.

θ_1 unifies t and u to the term $f(h(g(u)),g(y))$, and

θ_2 unifies t and u to the term $f(h(z),g(y))$.

θ_1 is clearly not the most general unifier of t and u since it is not a factor of θ_2. θ_2, on the other hand, is the most general unifier of t and u.

Two terms are said to *overlap* if one is unifiable with a nonvariable subterm of the other. In determining whether an overlap exists, the variables of one term are renamed, if necessary, so as not to conflict with those of the other term. Two terms could overlap in many ways resulting in many superpositions as discussed below.

Let s and t overlap. Their *superposition* is defined as either

a) s unifies with a nonvariable subterm t' of t, by m.g.u. θ, in which case $\theta(t)$ is called a superposition of s and t; or

b) t unifies with a nonvariable subterm s' of s, by m.g.u. θ, in which case $\theta(s)$ is called a superposition of s and t.

Consider, for example, the terms $s = x^{-1} \bullet x$ and $t = (x' \bullet y') \bullet z'$. The substitution $\theta = [x^{-1}/x', x/y']$ unifies $x^{-1} \bullet x$ with $x' \bullet y'$. The resultant superposition of s and t is $(x^{-1} \bullet x) \bullet z'$.

Now consider ordered pairs of terms (r,s) and (t,u) such that s and t overlap, as above. (If the variables of t must be renamed, the same renaming must be applied to u.) Then along with the superposition $\theta(t)$ or $\theta(s)$ we obtain the *derived pair* of terms $<p,q>$, where

a) if s unifies with a nonvariable subterm $t[i]$ by m.g.u. θ,

$p = [\theta(t)$ with $\theta(r)$ at $i]$,

$q = \theta(u)$;

b) if t unifies with a nonvariable subterm $s[i]$ by m.g.u. θ,

$p = \theta(r)$,

$q = [\theta(s)$ with $\theta(u)$ at $i]$.

In the case of a rewriting system $\mathbf{R} = \{(l_i, r_i)\}$, the derived pairs obtained from the pairs (r_i, l_i) and (l_j, r_j) are called *critical pairs.*

Consider, for example, obtaining a critical pair from the rewrite rules:

$x^{-1} \bullet x \rightarrow e$,

$(x' \bullet y') \bullet z' \rightarrow x' \bullet (y' \bullet z')$.

We begin by constructing the ordered pairs $(e, x^{-1}\bullet x)$ and $((x'\bullet y')\bullet z', x'\bullet(y'\bullet z'))$. As we saw earlier $x^{-1}\bullet x$ can be unified with $x'\bullet y'$ using the substitution $\theta = [x^{-1}/x', x/y']$. This leads to the derived pair $< e\bullet z', x^{-1}\bullet(x\bullet z')>$ which is a critical pair of the rules.

The computation of critical pairs is central to the Knuth-Bendix test for confluence (Knuth and Bendix (1970)). In this application, it is always the left-hand sides that are superposed with each other. In the method of searching for cycles to be described, we consider superpositions of the right- and left-hand sides as well.

4.2. Overlap closure. For a term rewriting system $\mathbf{R}$, the *overlap closure of* $\mathbf{R}$, written $\mathrm{OC}(\mathbf{R})$, is the term rewriting system defined inductively as follows:

a. Every rule $r \to s$ in $\mathbf{R}$ is also in $\mathrm{OC}(\mathbf{R})$.
b. Whenever $r \to s$ and $t \to u$ are in $\mathrm{OC}(\mathbf{R})$, every derived pair $<p,q>$ of (r,s) and (t,u) is in $\mathrm{OC}(\mathbf{R})$ (as $p \to q$).
c. No other rules are in $\mathrm{OC}(\mathbf{R})$.

That each derived pair is in fact a rewrite rule is shown by the following:

LEMMA 4.2.1. *If r,s,t,u are terms such that (r,s) and (t,u) are rewrite rules, then every derived pair $<p,q>$ of (r,s) and (t,u) is also a rewrite rule.*

Proof. One just has to verify that for each case in the definition of derived pair that every variable that occurs in q occurs also in p. □

Examples of overlap closures:

i. Let $\mathbf{R} = \{f(x) \to g(x)\}$, then $\mathrm{OC}(\mathbf{R}) = \mathbf{R}$.
ii. Let $\mathbf{R} = \{f(x) \to g(h(x)), h(x) \to k(x)\}$, then $\mathrm{OC}(\mathbf{R}) = \mathbf{R} \cup \{f(x) \to g(k(x))\}$.
iii. Let $\mathbf{R} = \{x\bullet(y\bullet z) \to (x\bullet y)\bullet z\}$, then from the superposition $(x\bullet(x'\bullet y'))\bullet z'$ we obtain the rule

$$x\bullet((x'\bullet y')\bullet z') \to ((x\bullet x')\bullet y')\bullet z'$$

and from the superposition $(x\bullet((x'\bullet y')\bullet z')$ we obtain

$$x\bullet(x'\bullet(y'\bullet z')) \to (x\bullet(x'\bullet y'))\bullet z'.$$

These rules then lead to further rules, and $\mathrm{OC}(\mathbf{R})$ is infinite.
iv. Let $\mathbf{R} = \{f(x) \to g(x), g(h(x)) \to f(h(x))\}$. Then $\mathrm{OC}(\mathbf{R})$ consists of $\mathbf{R}$ and the reflexive rules $f(h(x)) \to f(h(x))$ and $g(h(x)) \to g(h(x))$.

The name "overlap closure" comes from the fact that the rules of $\mathrm{OC}(\mathbf{R})$ are a subset of the transitive closure of the rewriting relation of $\mathbf{R}$:

LEMMA 4.2.2. *If $p \to q$ is in $\mathrm{OC}(\mathbf{R})$ then $p \to^+ q$ (using $\mathbf{R}$).*

Proof. By induction on the construction of $p \to q$ in $\mathrm{OC}(\mathbf{R})$. The basis of the induction is the case that $p \to q$ is included in $\mathrm{OC}(\mathbf{R})$ by virtue of being a rule of $\mathbf{R}$. Then obviously $p \to^+ q$ holds. If $(p \to q)$ is included in $\mathrm{OC}(\mathbf{R})$ by being a derived pair of (r,s) and (t,u) then by the induction hypothesis for the two rules (r,s) and (t,u) we have $r \to^+ s$ and $t \to^+ u$. By the definition of derived pair and the transitivity of $\to^+$, we then have $p \to^+ q$. □

COROLLARY 4.2.3. *If $\mathrm{OC}(\mathbf{R})$ contains a reflexive rule, $t \to t$, then the rewriting relation of $\mathbf{R}$ has a cycle.*

Proof. Immediate from the Lemma. □

We would like to have the converse of this corollary, that if the rewriting relation of $\mathbf{R}$ has a cycle, then $\mathrm{OC}(\mathbf{R})$ contains a reflexive rule. This would permit searching for cycles by incrementally computing $\mathrm{OC}(\mathbf{R})$, looking for a reflexive rule.

While we have not been able to prove this in full generality, we will present in the next section a restricted version and its proof. The proof is not easy, because the overlap closure of **R** is in general much smaller than the full transitive closure of **R**. It is this small size, relative to the transitive closure, however, that makes it feasible to use the overlap closure as the basis of an approach to proving uniform termination or at least, a useful notion of "restricted termination." These ideas will be discussed in § 6.

5. Rewrite dominoes and overlap closure theorems. In order to be able to develop and prove useful results about the overlap closure, we need to be able to deal precisely with the various cases of overlap between successive applications of rewrite rules in a rewrite sequence. We have found it useful to introduce a new representation of rewriting that helps to make such cases clear.

The *domino representation* (or *rewrite domino*) of a rewrite rule is a rectangle divided into left and right halves in which are inscribed tree representations of the left and right terms of the rule. Function symbols in the terms are represented by labelled circles in the trees. Variable symbols are represented by labeled rectangles, called "variable boxes." For examples of some rules and their corresponding rewrite dominoes, see Fig. 1.

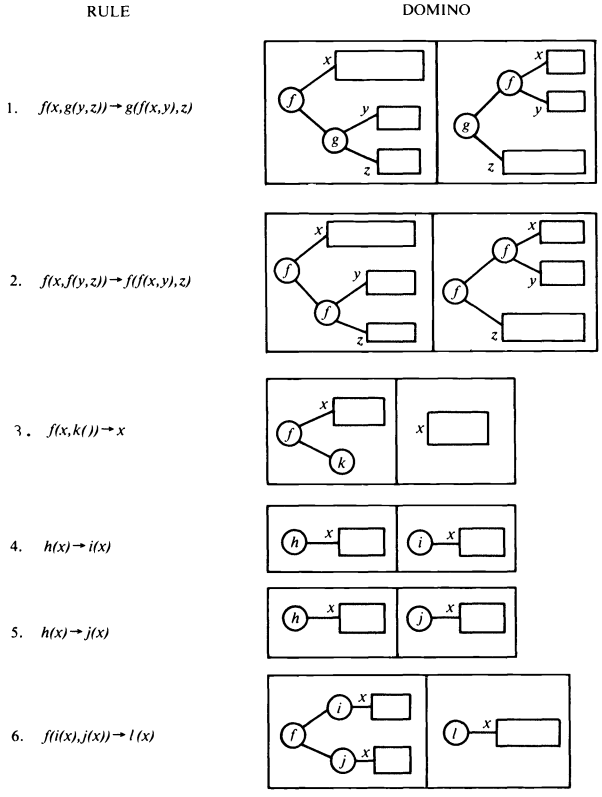


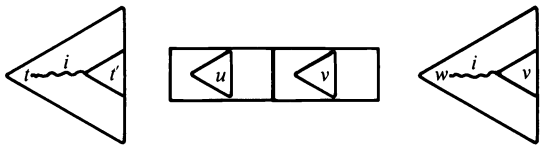
FIG. 1. *A set of rewrite rules and their corresponding rewrite dominoes.*

For each kind of domino (that is, each domino corresponding to a specific rule), we assume there is an infinite stock of dominoes of that kind with their variable rectangles filled in with all possible terms. For each such domino, we also assume an infinite number of copies are available in the stock.

A sequence of rewrites can be represented by a *domino layout,* which is a two-dimensional arrangement of dominoes that obeys the rules of matching corresponding to those of term rewriting (§ 2). Before giving the formal definition of a layout, we refer the reader to an example of a rewrite sequence using the rules given in Fig. 1 and its corresponding domino layout as shown in Fig. 2. Another example is in Fig. 3, and the two layouts in Figs. 2 and 3 could be concatenated to give a single longer layout.

We draw trees oriented sideways with the root at the left, and we will use nested triangles to represent trees schematically. We will give a simultaneous inductive definition of the terms *layout, position of a domino in a layout, adjacency* of dominoes in a layout, and *right-end dominoes* of a layout. (These definitions may seem tedious, but are necessary to avoid dependence in proofs on intuitive geometric notions.)

The basis of the definitions is a *unit layout from t to w:* a horizontal arrangement of a tree t, a domino, D, with trees u and v, and another tree w,



in which at some position, i, in t (as defined in § 2) there is a subtree t' that is identical to u, ignoring the variable boxes that appear in u; and w is the tree [t with v at i].

The position of D in this layout is i, and D is a right-end domino. A *layout* is defined as follows: a unit layout is a layout; and if L is a layout from t_0 to t_1 and L_1 is a unit layout from t_1 to t_2, then the concatenation L^+ of L and L_1, with t_1 deleted, is a layout from t_0 to t_2. The position of the domino D_1 of L_1 in the new layout L^+ is the position i of D_1 in L_1. In L^+, D_1 is said to be *adjacent to* just those dominoes of D and L that satisfy: D is a right-end domino of L and the position j of D is a prefix of i or i is a prefix of j (recall that positions are sequences of natural numbers). These dominoes are also said to be adjacent to D_1, so that adjacency is a symmetric relation. Finally, we say that D_1 is a right-end domino of L^+, and each domino of L is a right-end domino of L^+ if and only if it is a right-end domino of L and it is not adjacent to D_1.

The examples in Figs. 2 and 3 illustrate a number of observations we can make about this representation of rewriting:

1. Two dominoes that are not in the transitive closure of the adjacency relation represent rewrites of disjoint subterms. Thus in drawing layouts we allow one of such a pair of dominoes to be placed above the other. In other words, in a domino layout there is no distinction between different orders of rewriting when the rules are being applied to disjoint subterms; e.g., the layout in Fig. 3 would not be different if rule 5 had been applied before rule 4 or before rule 3. One can think of these rules being applied in parallel, since the order of application is always immaterial in this case. The layout representation just makes this property especially evident.

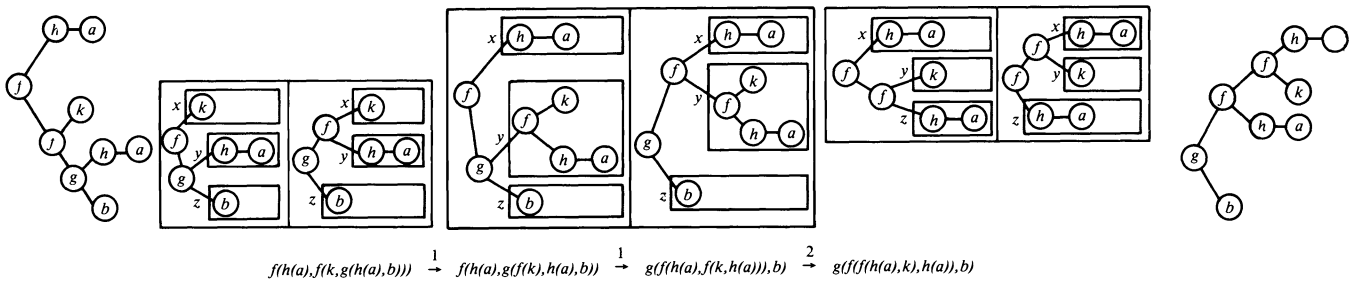


FIG. 2. *A rewrite domino layout and the corresponding rewriting sequence (using dominoes of Fig. 1).*

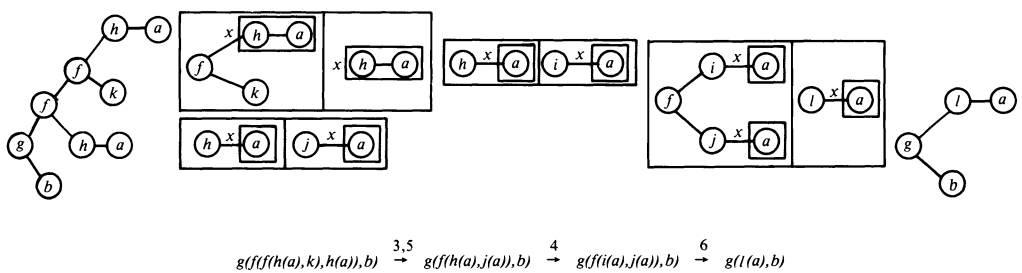


FIG. 3. *Another layout (a continuation of the layout in Fig. 2).*

2. To the property that "the rightmost term of a rewrite sequence is terminal" corresponds the property that "there is no way to play a domino on the layout" (formally, there is no way to concatenate a unit layout onto the layout). The layout is said to be *blocked.* (The layout in Fig. 3 is blocked.)
3. Thus the rules have the uniform termination property if and only if every possible layout eventually is blocked. Equivalently, there are no infinite layouts.

Our purpose with this representation of rewriting is to provide a conceptual tool for finding and presenting proofs of new results about term rewriting systems. The first result we will prove with the aid of rewrite dominoes is one that will allow us to speed up the search for cycles by considering only those sequences of rewrites in which a "major rewrite" occurs.

A rewrite $t_0 \rightarrow t_1$ is called a *major rewrite* if it is by application of a rule, $t \rightarrow u$, to the entire term t_0; i.e., for some substitution θ, $\theta(t) = t_0$ and $\theta(u) = t_1$. When only a proper subterm of t_0 is matched, $t_0 \rightarrow t_1$ is called a *minor rewrite.*

In a layout, a domino is called a *major domino* (of the layout) if it represents a major rewrite, and a *minor domino* otherwise.

A *major cycle* is a cycle in which at least one of the rewrites is major.

THEOREM 5.1. *If a rewriting relation has a cycle, it has a major cycle.*

Proof. Let us define the *corridor* of a domino D in a layout to be the horizontal rectangle containing all dominoes whose positions in the layout are equal to or extensions of the position of D, using the definition of position as a sequence of natural numbers. (For example, in Fig. 3, the central domino has position 1.1 and its corridor contains it and the top leftmost domino, which has the same position, but does not contain the bottom leftmost domino, which has position 1.2, or the rightmost domino, which has position 1. The corridor of the rightmost domino contains all of the dominoes of the layout.) If we add to a corridor of a domino D the appropriate trees at either end, it forms a sublayout of the original layout, and D is a major domino of this sublayout.

Any two corridors in a layout are either disjoint or one is contained in the other. Therefore, we can determine a sublayout which has a domino that is major in it, as follows: start with any leftmost domino and follow its corridor to the right; whenever a domino is encountered that does not lie in the corridor, adopt its corridor. When we reach the right end, we have a corridor containing a layout. If the whole layout is cyclic, the sublayout corresponding to this corridor will be also, and will represent a major cycle. □

"Major cycle" is a weaker notion than "prime cycle" introduced by Klop (1980), i.e., every prime cycle is a major cycle but not vice versa. Consider for example, the rewriting system, $\{f(x,y) \rightarrow f(y,x), a() \rightarrow b(), b() \rightarrow a()\}$. The cycle

$$f(a(),b()) \rightarrow f(b(),a()) \rightarrow f(b(),b()) \rightarrow f(b(),a()) \rightarrow f(a(),a()) \rightarrow f(a(),b())$$

is a major cycle but is not a prime cycle because another cycle

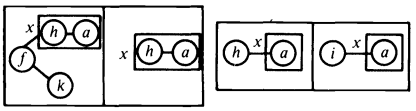
$$a() \rightarrow b() \rightarrow a()$$

is contained in it. Klop also proves a result about prime cycles similar to Theorem 5.1.

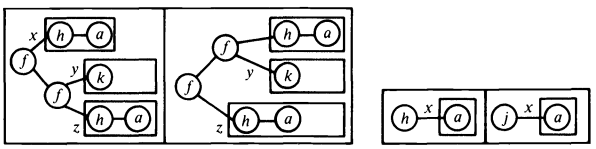
We now want to define some terminology and some manipulations of layouts that will be useful in proving theorems about the overlap closure of a set of rules. Consider an adjacent pair of dominoes in a layout. Let t and u be the trees on the adjacent halves, where a subtree t' of t is identical to u (possibly $t' = t$):

If either of t' or u is contained entirely within a variable box, i.e., the match is not between two nonvariable subterms, we say that the pair of dominoes is *weakly matched,* and otherwise that it is *strongly matched.*

Examples. In Fig. 3, the domino pair



is weakly matched. Similarly the pair



that appears in the concatenation of the layouts of Figs. 2 and 3 is weakly matched, while all the other adjacent pairs are strongly matched.

A term is said to be *linear* if no variable occurs in it more than once. A rewrite rule is *left-linear* if its left term is linear and *right-linear* if its right term is linear.

Now suppose we have two weakly matched dominoes, as in Fig. 4a, where t' is contained in the x variable box. If the (s,t) domino is right-linear (i.e., t is linear), then the pair of dominoes can be *transposed* as follows: remove the (u,v) domino from the layout and move the (s,t) domino to the right, so that copies of the (u,v) domino can be inserted to the left of the (s,t) domino, one adjacent to each x box in s (see Fig. 4b). Then *the resulting configuration is still a layout,* (the dominoes all match, using the same set of rules) with the same end trees. This is the case also when a symmetric kind of transposition is performed on a layout in Fig. 5a, producing the layout in Fig. 5b, where we assume that the (u,v) domino is left-linear.

Such transpositions cannot necessarily be performed on strongly matched dominoes, but we will define a different kind of manipulation for this case. Strong matching corresponds to the concept of overlapping in the definition of derived pairs: if

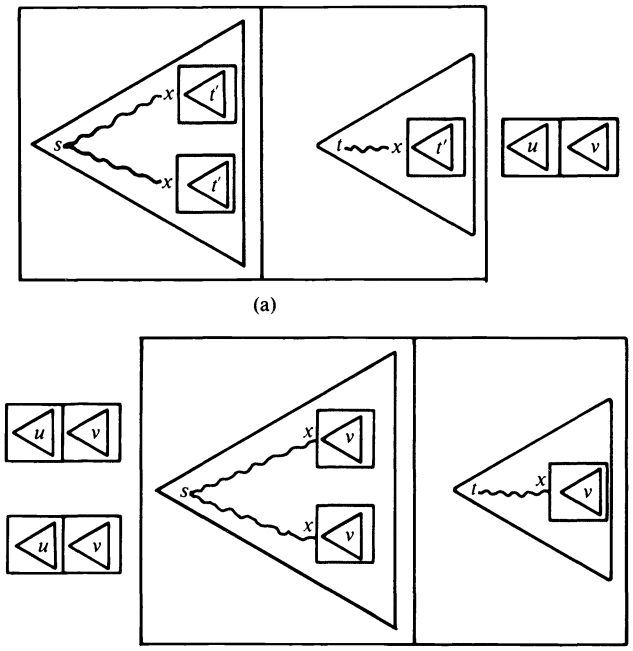


FIG. 4. *Transposition of weakly matched dominoes, where left domino is right-linear.*

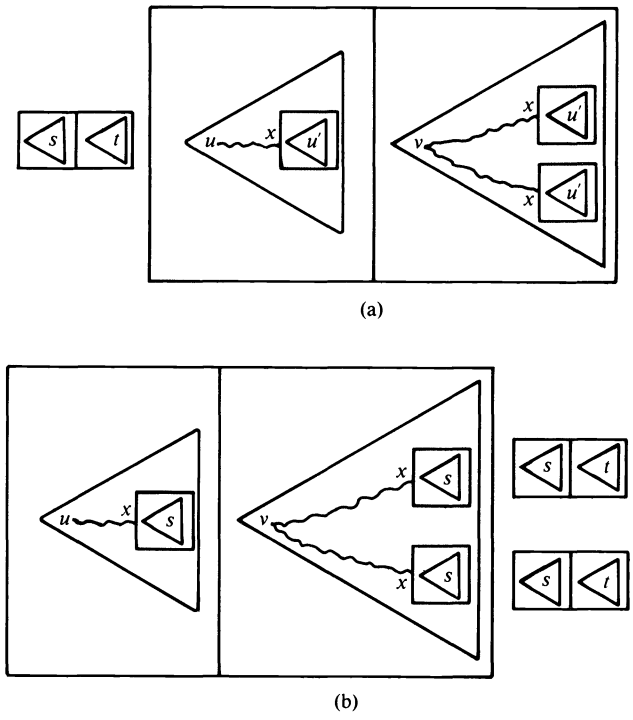


FIG. 5. *Transposition of weakly matched dominoes, where right domino is left-linear.*

(r,s) and (t,u) are rules that have a derived pair $<p,q>$, then the dominoes corresponding to (r,s) and (t,u) can be placed in a layout so that they are strongly matched. The layout configuration shows just where the strong match occurs and identifies a potential derived pair.

Suppose now that instead of our stock of dominoes corresponding to a given rule set **R**, we have a stock corresponding to OC(**R**), the overlap closure of **R**. Then for any strongly matched pair of dominoes in a layout there is a domino in our stock which corresponds to a derived pair generated by the matching pair. By a technical lemma proved in Appendix B, we can replace the strongly matched pair in the layout by the "derived pair domino" thus identified, and the result will still be a layout with the same end trees.

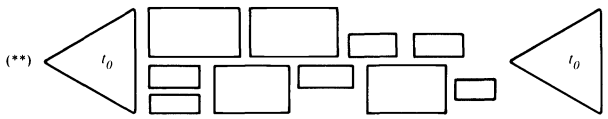
We are now in a position to prove:

THEOREM 5.2. *Suppose the rewriting relation of* **R** *is globally finite and every rule in* **R** *is right-linear. If the rewriting relation of* **R** *has a cycle,* OC(**R**) *contains a reflexive rule.*

Proof. (By construction.) Let

(*) $t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n \rightarrow t_0$

be a given cycle. Corresponding to (*) is a cyclic domino layout

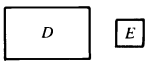


where the dominoes correspond to rules of **R**. In fact since each of these rules is also in OC(**R**), we may take this layout as a layout of dominoes corresponding to rules of OC(**R**). We will show how to manipulate this layout to a form that shows there is a reflexive rule $t \rightarrow t$ in OC(**R**).

We describe the manipulations as an algorithm operating on the cyclic layout (**).

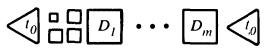
Step 1. [Extract major cycle.] As in the proof of Theorem 5.1, extract from (**) a sublayout representing a major cycle, making it the layout subject to the following steps. Also replace t_0 with its subterm matched by the layout.

Step 2. [Push major dominos to right end.] Manipulate the layout to a form in which all of the major dominoes are together at the right end, by means of transpositions or replacements by derived pair dominoes: whenever D is a major domino and E is a minor domino adjacent to D on the right



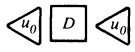
either D and E are weakly matched, in which case they can be transposed, or they are strongly matched, in which case they can be replaced by the derived pair domino they define — which is a major domino. This derived pair domino is also right linear, as the lemma in Appendix C shows.

Step 3. [Look for cycle among major dominoes.] There is now a nonempty sequence of major dominoes $D_1, \cdots, D_m$ at the right end of the layout:



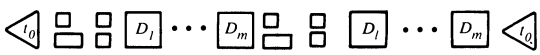
These dominoes can only be strongly matched — except for the case where the right-hand side of D_i is just a variable, but shortly we will show that such a possibility can be ruled out. If there is some contiguous subsequence $D_i, \cdots, D_j$ that forms a cyclic layout

then, since there can only be strong matches, these dominoes can be combined by $j-i+1$ replacements into a single domino D that forms a cyclic layout:



Let D represent (p,q). Then there is a substitution θ such that $u_0=\theta(p)$ and $\theta(q)=u_0$, i.e., θ unifies p and q. Furthermore, a derived pair of (p,q) and (p,q) is the reflexive rule $(\theta(p),\theta(q))$. Since this is in $\mathrm{OC}(\mathbf{R})$, we terminate the algorithm.

Step 4. [Duplicate.] If no such subsequence exists, concatenate to the layout a copy of the layout. Return to Step 2 with the resulting layout:



That concludes the statement of the algorithm. Before considering the question of termination of the algorithm, we dispense with the detail mentioned in Step 3: the case of adjacent major dominoes D and E where the right term u of D is just a variable. We can assume the left term t of D is not just a variable (if it were then it would have to be the same variable as u and we would already have a reflexive rule). Since the layout is cyclic, if we drop D from the layout, we obtain a layout that has as its right end term a proper subterm identical to the left end term. From this we conclude that the term rewriting relation is not globally finite, contrary to assumption. This contradiction rules out the case under discussion.

It is obvious that each step of this algorithm is effective and terminating. Overall termination is guaranteed by the following facts:

a. At the k th execution of Step 2, the number of major dominoes, m, at the right end is at least 2^k.

b. Let $t'^{(k)}_0$ denote the term to the left of D_1 in the layout at the k th execution of Step 3. Since each $t'^{(k)}_0$ is derived from t_0 and the rewriting relation is globally finite, there are only finitely many distinct possibilities for $t'^{(k)}_0$. By a), then, there is one such term for which arbitrarily long layouts of major

dominoes exist. Again by global finiteness, these layouts cannot all continue without producing a term, u_0, that is a duplicate of some term previously obtained in the layout.

Since the algorithm always terminates, and does so with a reflexive rule in OC(**R**), this proves the theorem. □

The corresponding theorem obtained by replacing "right-linear" by "left-linear" can also be proved in a similar manner. Combining these theorems with Corollary 4.3, we have:

THEOREM 5.3. *Suppose the rewriting relation of* **R** *is globally finite and every rule in* **R** *is right-linear or every rule in* **R** *is left-linear. Then the rewriting relation of* **R** *is uniformly terminating if and only if* OC(**R**) *contains no reflexive rule.*

In the next section we explore some applications of this theorem.

Recently, Dershowitz (1981) has proposed a "forward chain" construction for rewriting systems and proved that a right-linear rewriting system is uniformly terminating if and only if it has no infinite forward chains. However, for left-linear systems the analogous result requires that the left-hand sides of the rules be nonoverlapping, a problem that we had independently encountered when considering the forward chain construction and a similar backward chain construction. We were thus led to invent the overlap closure construction. The following example from Dershowitz (1981) illustrates the advantage of the overlap closure construction over forward chains. Using the forward chain construction, it is not possible to determine the nontermination of this left-linear rewrite system, as pointed out by Dershowitz. The rewriting system is

$$f(a(),b(),x) \to f(x,x,b()),$$

$$b() \to a().$$

These rules have only two forward chains, both finite:

$$f(a(),b(),x) \to f(x,x,b()) \to f(x,x,a()), \quad \text{and} \quad b() \to a(),$$

but we cannot conclude anything about the termination of the rules because they are not right-linear and, although they are left-linear, the left-hand sides are overlapping. But in the overlap closure construction, the rules have a derived pair rule

$$f(b(),b(),x) \to f(x,x,b()),$$

which, when overlapped with itself, gives the reflexive rule

$$f(b(),b(),b()) \to f(b(),b(),b()),$$

as a derived pair, proving that the rules are nonterminating.

An open question about the power of the overlap closure construction is whether the assumption of left-linearity or right-linearity is necessary. Although we have not been able to find proofs of our results without this assumption, we have also been unable to construct a counterexample.

6. Using the overlap closure. Theorem 5.3 implies that the uniform termination property for globally finite, right-linear (left-linear) rewrite systems is decidable when the overlap closure is finite. Unfortunately, OC(**R**) will usually be infinite.

When OC(**R**) is infinite, Theorem 5.3 implies that nontermination is semidecidable. In our experience, if **R** is nonterminating, then there is a cycle involving terms of size comparable to the terms in the rules. Since rules in OC(**R**) get very big soon, we are likely to generate a reflexive rule very quickly.

Theorem 5.3 has utility for proving "restricted termination," i.e., termination for all terms in some set S. We describe this as *termination over S*. A particularly interesting set to look at is the set of all terms of limited size.

LEMMA 6.1. *If a rewriting relation arrow is globally finite and S is finite, termination over S is decidable.*

Proof. For each element of S, we just have to follow all possible rewriting paths until we either reach a terminal form or a cycle. By global finiteness, there can be only finitely many such paths and each is of finite length. □

Let $OC_n(\mathbf{R}) = \{(l,r) \text{ in } OC(\mathbf{R}): \text{Size}(l) \leqslant n\}$. If $\mathbf{R}$ contains only nonexpanding rules, then $OC_n(\mathbf{R})$ can be computed by starting with the empty set, inserting all rules (l,r) of $\mathbf{R}$ such that $\text{Size}(l) \leqslant n$, and continuing to compute and insert derived pairs until a set S is reached such that for any rules (r,s) and (t,u) from S any derived pair (p,q) is either in S or has $\text{Size}(p) > n$. As for $OC(\mathbf{R})$, checking whether $OC_n(\mathbf{R})$ has a reflexive rule is better for deciding termination over set of terms of size $\leqslant n$ than the brute force approach suggested in the proof of the above lemma, because the search space in the former case is reduced considerably.

LEMMA 6.2. *Suppose* $\mathbf{R}$ *contains only right-linear (left-linear) and nonexpanding rules. If* $OC_n(\mathbf{R})$ *contains no reflexive rule,* $\mathbf{R}$ *is terminating over the set of all terms of size n or less.*

Proof. By contradiction. Suppose there is a cycle with terms of size less than or equal to n; from the cycle, we will construct a reflexive rule in $OC_n(\mathbf{R})$ in the same way as in the proof of Theorem 5.2.

Let the cycle be

$$t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_m == t_0.$$

The size of t_i is less than or equal to n; note that the size of each term in the cycle is the same, as rules are restricted to be nonexpanding. Using the construction suggested in the proof of Theorem 5.2, we can extract a major cycle. In Step 2, if the rules are right-linear (left-linear), weak matches between adjacent dominoes are transposed to the left (right). At the end of Step 2, there is a nonempty sequence of major dominoes having terms of size k at the right (left) end of the domino layout. If there is some contiguous subsequence that forms a cyclic layout, then there is a reflexive rule in $OC_k(\mathbf{R})$. Otherwise, we do Step 4. Since all dominoes have terms of size k, their derived pairs will be of size less than or equal to k. So, we will eventually get a reflexive rule in $OC_k(\mathbf{R})$. □

A reduction relation arrow is said to be *canonical over S* if it is terminating over S and any two elements in S have the same terminal form if and only if they belong to the same $\leftrightarrow^*$ equivalence class. Thus for a term rewriting relation that is canonical over S, equations expressed using terms drawn from S are decidable by rewriting to terminal forms and checking for identity. We have the following partial generalization of Theorem 2.2.2.

THEOREM 6.3. *If a reduction relation arrow is both terminating over S and uniformly confluent, it is canonical over S.*

Remark. Uniform confluence, not just "confluence over S," is necessary here. The following rewrite system illustrates this point.

$$a(x) \rightarrow h(x),$$

$$a(x) \rightarrow k(x).$$

If $S=\{h(x),k(x)\}$, then clearly $\rightarrow$ is both terminating over S and confluent over S, but is not canonical over S, as $h(x)=k(x)$.

Proof. By contradiction. Let t_1 and t_2 be in S for which arrow is not canonical; i.e., t_1 and t_2 belong to the same $\leftrightarrow^*$ equivalence class but they have different terminal forms t_1^* and t_2^*.

$$t_1 \leftrightarrow t'_1 \leftrightarrow \cdots \leftrightarrow t'_k \leftrightarrow t_2$$

where $\leftrightarrow$ stands for either $\rightarrow$ or $\leftarrow$. If $t_1 \rightarrow t'_1$, then t'_1 also has the terminal form t_1^*. Otherwise, if $t_1 \leftarrow t'_1$, then t'_1 has the terminal form t_1^* by uniform confluence. Similarly, for any $t'_i \leftrightarrow t'_j$, t'_i and t'_j have the same terminal form t_1^*. In particular, t_2 has the terminal form t_1^*, but t_2^* is also the terminal form of t_2 implying that $t_1^* == t_2^*$. Hence the contradiction. □

The above theorem about restricted canonicity is useful only if it is possible to prove uniform confluence (or more importantly to transform a given set of rules into a uniformly confluent set of rules having the same equational theory as the original set) in the absence of uniform termination requirement on the rule set. We have attempted to modify the Knuth-Bendix algorithm to do this, but without success. Peterson and Stickel (1981) remark that such an approach for establishing uniform confluence is not likely to be successful, as it would allow one to prove the decidability of the word equation problem over free semigroups in a simple way as follows:

Given the rule set for semigroups

$$x \bullet (y \bullet z) \rightarrow (x \bullet y) \bullet z,$$

$$(x \bullet y) \bullet z \rightarrow x \bullet (y \bullet z);$$

to solve a word equation problem (i.e., for a pair of terms, t_1 and t_2, whether there exists a substitution of variables which make the resulting instances of t_1 and t_2 equal in the theory of free semigroups), we add the following rules

$$h(t_1) \rightarrow 0,$$

$$h(t_2) \rightarrow 1,$$

where h is a new function symbol distinct from all other function symbols appearing in words. (0 and 1 are assumed to be distinct.) If the above set of rules is uniformly confluent, then there does not exist any substitution θ such that $\theta(t_1)=\theta(t_2)$ in the theory of free semigroups; otherwise, such a substitution exists. The word equation problem over free semigroups is a difficult problem which has been worked on for over 20 years and only recently solved by Makanin who gives a very lengthy and complex proof.

Proving restricted termination is useful in analyzing automatic implementation of a data type generated from its algebraic specification as discussed in Guttag, Horowitz, and Musser (1978). An implementation of an operation is essentially taking the expression corresponding to the operation invocation and simplifying it using the rewrite rules specifying the operation behavior. It is possible to run such an implementation and evaluate expressions whose termination can be proved a priori. Analysis of the operation behavior is helpful in designing abstract data types.

Proving restricted termination is also useful in an OBJ-like system (Goguen and Tardo (1979)) for designing specification of abstract data types and algorithms using equational axioms. Axioms are viewed as unidirectional rewrite rules and specifications are analyzed by interpreting expressions. OBJ has a memory mode in

which during the simplification of expressions, rewriting of expressions in the presence of cycles is allowed. While rewriting a term, when a term from the set S whose termination is proved is hit, there is no need to store the intermediate terms generated in the rewriting from that point onwards.

7. Conclusion. In rewrite rule theorem proving, as elsewhere, almost all of the interesting questions are undecidable. A particularly interesting undecidable question, with a number of practical ramifications, is whether or not a given set of rewrite rules has the uniform termination property. The motivation behind the work presented in this paper is the circumvention of this undecidability.

Our rather distinctive approach to this began with dividing the problem of proving uniform termination into the separable component problems of proving global finiteness and proving acyclicity. We then dealt with the former by developing sufficient conditions for establishing global finiteness. The conditions developed in § 3 and Appendix A seem reasonably general, and, moreover, are computationally easy to check. They are based on showing that every instance of each rule is nonexpanding, a property for which we presented syntactically checkable necessary and sufficient conditions.

Proving acyclicity, on the other hand, is not amenable to any kind of local, rule at a time, analysis. Our approach to dealing with this problem started with the development of a procedure that finds a cycle if one exists but may not terminate otherwise. The obvious procedure for doing this, which is tremendously inefficient, is based on the enumeration of all terms over the alphabet of the rewriting system. Our procedure, which is closely related to the Knuth-Bendix procedure for constructing confluent sets of rewrite rules, is based on the computation of what we called the overlap closure of the set of rewrite rules. In order to develop this procedure, we introduced in § 5 a new model of term rewriting called rewrite dominoes. The primary application of this model was to prove that a set of rewrite rules, provided they are all right-linear (or all left-linear) and globally finite, has a cycle if and only if its overlap closure contains a reflexive rule.

This result implies that in the cases where the overlap closure computation terminates we can decide uniform termination of the original set of rewrite rules. Unfortunately the overlap closure is rarely finite. However, our procedure generates the overlap closure in such a way that for any integer n we can generate a subset of the overlap closure that is sufficient to decide whether or not there is a cycle in which every term is of size n or less. This gives us a decision procedure for what we called restricted termination. We conjecture that for certain classes of term rewriting systems, it should be possible to compute a bound, n, such that if a cycle exists, there exists a cycle in which every term is of size n or less. For such classes, the overlap closure would provide a decision procedure for uniform termination.

We have explored the utility of restricted termination in the application of term rewriting with which we are most familiar, the algebraic specification of abstract data types, and see a number of interesting ways to make use of it there. We suspect the notion of restricted termination will be useful in other application areas, but we have not investigated this. Investigations should also be made into the usefulness of the overlap closure computation and the domino model of term rewriting systems for the study of properties other than uniform or restricted termination.

Appendix A. Proving uniform termination of rules using if-then-else operator. In specifying abstract data types using equations, it is often useful to introduce an auxiliary function *if-then-else* with the following semantics:

$$\text{if-then-else}(\text{true},x,y) = x,$$
$$\text{if-then-else}(\text{false},x,y) = y.$$

But the rewrite rules corresponding to the equational axioms using *if-then-else* are often expanding; for example, consider the following rule:

$$\text{has?}(i,\text{insert}(s,i')) \rightarrow \text{if-then-else}(i = i',\text{true},\text{has?}(i,s)).$$

We will only consider rewrite rules whose right-hand side may use *if-then-else* operator; their left-hand sides are assumed not to use *if-then-else,* except for the rules

$$\text{if-then-else}(\text{true},x,y) \rightarrow x,$$
$$\text{if-then-else}(\text{false},x,y) \rightarrow y.$$

The uniform termination of such rules can be proved by considering the termination of a corresponding set of rules that do not use *if-then-else* operator, as the following theorem shows. For a rule (l,r) in $\mathbf{R}$, the corresponding set $D(\{(l,r)\})$ of rules is defined as:

$$D(\underset{i}{U}R_i) = \underset{i}{U}D(R_i)$$

$$\{(l,r)\} \quad \text{if } r \text{ does not use } \textit{if-then-else}.$$

$$D(\{(l,r)\}) = D(\{\ (l,[r \text{ with } b \text{ at } i]), \quad \text{where } \textit{if-then-else}(b,e_1,e_2) \text{ is a subterm of } r \text{ at position } i.$$
$$(l,[r \text{ with } e_1 \text{ at } i]),$$
$$(l,[r \text{ with } e_2 \text{ at } i])\})$$

A rule (l,r) expressed using if-then-else is called *componentwise-nonexpanding* if the corresponding derived set of rules, $D(\{(l,r)\})$, is nonexpanding. Similarly, the rule set $\mathbf{R}$ is *componentwise-nonexpanding* if $D(\mathbf{R})$ is nonexpanding.

6 THEOREM A.1. *If* $\mathbf{R}$ *is componentwise-nonexpanding, then* $\mathbf{R}$ *is uniformly terminating if and only if* $D(\mathbf{R})$ *is uniformly terminating.*

Proof. Since $\mathbf{R}$ is componentwise-nonexpanding, $D(\mathbf{R})$ is nonexpanding and hence globally finite.

(i) ==> (by contradiction)

Assume that $D(\mathbf{R})$ is not uniformly terminating and $\mathbf{R}$ is. Since $D(\mathbf{R})$ is globally finite, there is a cycle $t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n == t_0$, $n > 0$. This cycle must use a rule in $D(\{l \rightarrow r\})$ such that r contains the if-then-else operator, otherwise $\mathbf{R}$ would have a cycle. From the above cycle, we can construct a sequence of rewrites in $\mathbf{R}$ such that wherever a rule from the set corresponding to a rule in $\mathbf{R}$ using if-then-else is applied in the cycle, we apply the rule in $\mathbf{R}$. Let the sequence of rewrites be

$$t_0 \rightarrow t'_1 \rightarrow \cdots \rightarrow t'_n.$$

t'_n in that case contains t_0, which can be further rewritten. Thus we get an infinite sequence of rewrites using $\mathbf{R}$, implying $\mathbf{R}$ is not terminating, which is a contradiction.

(ii) <= (by contradiction)

Assume that **R** is not uniformly terminating and $D(\mathbf{R})$ is. There are two possibilities:

(a) **R** leads to an infinite sequence of rewrites.

$$t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_n \rightarrow \cdots$$

Since **R** is componentwise-nonexpanding, there must exist i, j such that $j > i$ and the subterm, t'_i, in t_i being rewritten at the $i+1$st step is a subterm in t_j. From the rewriting sequence

$$t_i \rightarrow \cdots \rightarrow t_j, \quad (*)$$

we can construct a finite set S of rewriting sequences in $D(\mathbf{R})$ by following the rewriting sequence (*) as follows:

(i) Initialize S to be $\{t_i\}$.

(ii) At a rewriting step $t_k \rightarrow t_{k+1}$ in which the subterm t'_k at position m is rewritten using the rule (l,r) of **R**,

(a) if r does not use *if-then-else* operator, discard sequences in S whose last term does not have t'_k at position m and extend other sequences in S by rewriting t'_k at position m using (l,r),

(b) otherwise, discard sequences in S whose last term does not have t'_k at position m and extend other sequences in S by rewriting t'_k at positon m in all possible ways using every rule in $D(\{(l,r)\})$.

Thus S will have a sequence with its last term having t'_i as a subterm. This implies that $D(\mathbf{R})$ is nonterminating.

(b) **R** has a finite cycle:

Using the same argument as in (a), it can be shown that $D(\mathbf{R})$ also has a finite cycle. □

The above theorem can be generalized to handle rules having the following property: the rules use function symbols that have the following semantics and that do not appear in the left-hand side of any rules other than the rules giving their semantics;

$$f(i,x_1, \cdots, x_n) \rightarrow x_i, \qquad \text{for } 1 \leqslant i \leqslant n.$$

Note that if-then-else operator is a particular case of the above function symbol f when $n = 2$. So, for proving uniform termination of such rules, we derive from the rules using these function symbols on their right-hand side, a corresponding set of rules which do not use these function symbols and prove uniform termination of the new set.

Appendix B. Proof of lemma used in Theorem 5.2.

LEMMA B.1. *Suppose* $t_0 \rightarrow t_1$ *using* $r \rightarrow s$ *applied at position* i, $t_1 \rightarrow t_2$ *using* $t \rightarrow u$ *applied at* $i.j$, *and* $s[j]$ *and* t *overlap determining the derived pair* $<p,q> = <\theta(r), [\theta(s)$ *with* $\theta(u)$ *at* $j]>$. *Then* $t_0 \rightarrow t_2$ *using* $p \rightarrow q$ *applied at* i. *A similar result holds for the case in which* s *unifies with a subterm of* t.

Proof. Rename the variables of t and u, if necessary, so that s and t have no variable in common. There is some subterm $t_0[i]$ and a substitution θ_1 such that $\theta_1(r) = t_0[i]$ and $t_1 = [t_0$ with $\theta_1(s)$ at $i]$.

Again, there is some subterm $t_1[(i.j)]$ and a substitution θ_2 such that $\theta_2(t) = t_1[(i.j)]$ and $t_2 = [t_1$ with $\theta_2(u)$ at $i.j]$.

Since the variables of s and t are disjoint, we have $(\theta_1 \cup \theta_2)(s[j]) = \theta_1(s[j]) = \theta_2(t) = (\theta_1 \cup \theta_2)(t)$. That is, $\theta_1 \cup \theta_2$ is a unifier of $s[j]$ and t and therefore has θ as a factor:

$\theta_1 \cup \theta_2 = \theta_3 \bullet \theta$, for some substitution θ_3.

Thus $t_0[i] = \theta_1(r) = (\theta_1 \cup \theta_2)(r) = (\theta_3 \bullet \theta)(r) = \theta_3(\theta(r)) = \theta_3(p)$. That is, t_0 is matched by p at i. Now consider $\theta_3(q)$; it is

θ_3 ([$\theta(s)$ with $\theta(u)$ at j])

= [$\theta_3(\theta(s))$ with $\theta_3(\theta(u))$ at j]

= [$\theta_1(s)$ with $\theta_2(u)$ at j].

Thus $t_2 =$ [t_1 with $\theta_2(u)$ at $i \bullet j$]

= [[t_0 with $\theta_1(s)$ at i] with $\theta_2(u)$ at $i \bullet j$]

= [t_0 with [$\theta_1(s)$ with $\theta_2(u)$ at j] at i]

= [t_0 with $\theta_3(q)$ at i], showing that $t_0 \rightarrow t_2$ using $p \rightarrow q$ applied at i.

We omit the proof of the case in which s unifies with a subterm of t. □

Appendix C. Derived pair construction is (left, right) linearity preserving.

THEOREM C.1. *If $r \rightarrow s$ and $t \rightarrow u$ are two right linear rules with disjoint variable sets, then each of their derived pairs, $< p, q >$ is also right linear.*

Proof. There are three cases:

(i) s unifies with the subterm $t[i]$ of t by their m.g.u. θ.

The corresponding derived pair $< p, q >$ has

$p =$ [$\theta(t)$ with $\theta(r)$ at i],

$g = \theta(u)$.

Since s is linear, by the following lemma, substitutions for any two distinct variables in $t[i]$ in θ do not have a common variable. The variables in t other than the ones in $t[i]$ do not play any role. So $\theta(u)$ is linear.

(ii) the subterm $s[i]$ of s unifies with t by their m.g.u. θ.

The corresponding derived pair $< p, q >$ has

$p = \theta(r)$,

$q =$ [$\theta(s)$ with $\theta(u)$ at i].

Since $s[i]$ is linear, by the lemma, substitutions for any two distinct variables in t in θ do not have a common variable. So, $\theta(s)$ and $\theta(u)$ are linear. And, q is thus linear.

(iii) if subterms of s do not unify with t or s does not unify with subterms of t, then there are no derived pairs of $r \rightarrow s$ and $t \rightarrow u$. □

By a similar argument, it can also be proved that every derived pair of two left linear rules is left linear.

LEMMA C.2. *Let t and u be unifiable terms with disjoint variable sets, and θ be their most general unifier. Let θ^* be the restriction of θ to the variables of u, say $\theta^* = [e_1/v_1, \cdots, e_n/v_n]$. If t is linear, then all variables in $e_1, \cdots, e_n$ are distinct.*

Proof. For every variable x having $k (>1)$ occurrences in u, replace different occurrences of x by distinct variables $x_1, \cdots, x_k$ that do not appear in t and u. Let u' be the resulting term which is linear.

By the following lemma, in the m.g.u. θ' of t and u', substitutions for distinct variables in t and u' do not have a common variable. Let σ_x be the m.g.u. for the set of terms $\theta'(x_i)$, $1 \leqslant i \leqslant k$, the substitutions for the variables used to replace multiple occurrences of x in u. If these σ_x for every variable x having multiple occurrences in u are composed with θ', we get a unifier of t and u.

In this unifier, substitutions for variables in u do not have a common variable. From this, it is evident that the m.g.u. θ of t and u cannot have substitutions for variables in u that share common variables. □

LEMMA C.3. *For two unifiable terms t and u, if t and u are linear, then the substitutions in their m.g.u. θ for any two distinct variables of t or u do not have common variables.*

Proof. By induction on the structure in term t.

Basis. t is a variable.

Then $\theta(t) = u$ and the statement trivially holds.

Inductive step. $t == f(t_1, \cdots, t_n)$

For t and u to be unifiable, either u is a variable or $u == f(u_1, \cdots, u_n)$. The case of u being a variable is handled as in the basis step.

For the case $u == f(u_1, \cdots, u_n)$, for each i, $1 \leqslant i \leqslant n$, t_i must unify with u_i by their m.g.u. θ_i, say. By the inductive hypothesis, the statement holds for each of θ_i. Since t and u are linear, the disjoint union of θ_i, $1 \leqslant i \leqslant n$, is the m.g.u. θ of t and u. It follows that the statement of the lemma holds for θ also. □

Acknowledgments. The basic idea of conducting a search for repeated terms (cycles) or subterms sprang from discussions in 1977 between one of the authors (Musser) and Dallas Lankford. We thank P. Gloess, G. Huet, and J. Levy for their interest and assistance in refuting some of our earlier conjectures, thus helping us arrive at the notion of the overlap closure and the theorems of § 5. We also thank P. Narendran for assistance in constructing the proof of the theorems in Appendix C, J. Goguen for discussions of the approach to term rewriting used in OBJ, and M. O'Donnell for helpful comments and pointing out the relationship between some of the definitions in § 5 to Klop's work.

REFERENCES

N. DERSHOWITZ, *Termination of linear rewriting systems — preliminary version,* in Automata, Languages, and Programming, S. Even and O. Kariv, eds., Eighth Colloquium, Israel, Lecture Notes in Computer Science 115, Springer-Verlag, New York, 1981.

J.A. GOGUEN AND J. TARDO, *An introduction to OBJ-T,* Proc. of Conference on Specification of Reliable Software, 1979.

J.V. GUTTAG, E. HOROWITZ, AND D.R. MUSSER, *Abstract data types and software validation,* Comm. A.C.M., 21 (1978), pp. 1048-1064.

G. HUET, *Confluent reductions: abstract properties and applications to term rewriting systems,* J. ACM, 27 (1980), pp. 797-821.

G. HUET AND D.S. LANKFORD, *On the uniform halting problem for term rewriting systems,* Rapport Laboria 283, INRIA, Paris, March 1978.

G. HUET AND D.C. OPPEN, *Equations and rewrite rules: a survey,* in Formal Languages Theory: Perspectives and Open Problems, R. Book, ed., Academic Press, New York, 1980.

J.W. KLOP, *Reduction cycles in combinatory logic,* in To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism, J.P. Seldin and R. Hendley, eds., Academic Press, New York, 1980, pp. 193-214.

D.E. KNUTH AND P. BENDIX, *Simple word problems in universal algebra,* in Computational Problems in Abstract Algebra, J. Leech, ed., Pergamon Press, New York, 1970, pp. 263-297.

D. MUSSER, *Abstract data type specification in the AFFIRM system,* IEEE Trans. Software Engineering, 6 (1980), pp. 24-31.

G.E. PETERSON AND M.E. STICKEL, *Complete sets of reductions for equational theories,* J. ACM, 28 (1981), pp. 233-264.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0001 $01.25/0

CONCURRENCY CONTROL BY LOCKING*

CHRISTOS H. PAPADIMITRIOU†

Abstract. We present a geometric method for studying concurrency control by locking. When there are only two transactions, our method yields an exact characterization of safe locking policies and also of deadlock-free locking policies. Our results can be extended to more than two transactions, but in that case the problem becomes NP-complete.

Key words. database, concurrency control, transaction, locking, two-phase locking, geometry of locking, serializability, safety, NP-complete problems

1. Introduction.

A database consists of a set of named data objects called *entities*. The values of these entities must at any time be related in some ways, prescribed by the *integrity constraints* of the database. When a user accesses or updates a database, she/he may have to violate temporarily these integrity constraints, in order to restore them at some later time, with the specific data changed. For example, in a banking system, there may be no way to transfer funds from an account to another in a single atomic step, without temporarily violating the integrity constraint stating that, say, the sum of all balances equals the total liability of the bank. For this reason, several steps of the interaction of the same user with the database are grouped into a *transaction*. Transactions are assumed to be *correct*, that is, they are guaranteed to preserve consistency when run in isolation from other transactions.

When many transactions access and update the same database concurrently, the consistency of the database may fail to be restored after all transactions have been completed. If, for example, transaction 1 consists of the two steps

$$x := x+1, \qquad x := x-1$$

and transaction 2 of the single step

$$x := 2*x,$$

and the consistency requirement is simply "$x = 0$", then executing transaction 2 between the two steps of transaction 1 turns a consistent database into an inconsistent one. This is despite the fact that both transactions are *individually* correct; that is, each preserves database consistency when run alone. We must therefore find ways to prevent such undesirable interleaving without excessively harming the parallelism and overall efficiency of the system. This is the *database concurrency control* problem, already discussed extensively in the literature (see [Pa2], [EGLT1], [EGLT2], [SLR], [BGRP], [Pa1], [KP], [Ya], [SK]).

Since we assume that each transaction is by itself correct, a reasonable goal for the database concurrency control mechanism would be to rearrange the steps of the transactions so that the resulting sequence of steps is *serializable*. This means that its effect is as though the transactions executed without any interleaving, one after the other in some order. Serializability has been widely recognized as the right notion of correctness (e.g. [EGLT1], [SLR], [Pa1]). In fact, in [KP] we show that it is the most liberal notion of correctness possible, when only syntactic information (i.e., the names of the entities accessed at each step) is available. We denote the set of all serializable schedules by SR.

* Received by the editors December 15, 1981, and in revised form June 7, 1982.

† Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, and National Technical University of Athens, Greece.

A very common way for implementing concurrency control is *locking*. In this method each entity is equipped with a binary semaphore—its *lock*—and transactions synchronize their operation by locking and unlocking the entities that they access. The lock-unlock steps are inserted in a transaction according to some *locking policy*. A locking policy may have the property that, if all transactions are locked according to it, then any interaction is guaranteed to be serializable. Such a locking policy is called *safe*. A classical example of safe locking policies is the *two-phase locking* (2PL) policy proposed in [EGLT1]. In 2PL a transaction must lock an entity before accessing it, and must not unlock it until after the last access of the entity, *and after the last lock of the transaction is granted*. Thus a transaction has two phases: the *locking phase*, during which the transaction requests (but does not release) locks, and the *unlocking phase*, during which locks are released but not requested. 2PL is a safe locking policy [EGLT1]. Another safe locking policy is the *tree policy* of [SK], in which the entities are arranged in a tree, typically one reflecting some logical or physical structure, and transactions access whole subtrees. An entity may be locked only if its father entity is currently locked. This tree policy can be further generalized to the *digraph policy* and the *hypergraph policy* [Ya]; the latter is shown in [Ya] to be the most general safe locking policy possible, in the sense that *all* safe policies can be considered as special cases of the hypergraph policy.

In [Pa1], [KP] we proposed a measure whereby the performance of concurrency control mechanisms in general—and locking policies in particular—can be evaluated in a uniform setting. This measure is expressed in terms of the class of all sequences of transaction steps that can be the response of the concurrency controller to a stream of execution requests. The richer this class, the fewer unnecessary delays and rearrangements of steps will occur, and the greater the *parallelism* supported by the system.

In this paper we study safe locking policies in a theoretical, unifying way. In § 2 we describe our model and define our terminology.

In §§ 3 and 4 we characterize safe locking policies. We first give a characterization of safety for the case of two transactions. To do so, we employ a geometric methodology reminiscent of that used by Dijkstra for studying *deadlocks* [CES], [CD]. Here we use it in a very different way to study *incorrect completions*. (We ignore deadlocks, as seems reasonable to do in view of their limited significance in this context; see [GLPT].)

Besides its independent interest and elegance, the two-transaction solution is the building block for resolving the general case (§ 4). It turns out that a locking policy defined on $d > 2$ transactions is safe if and only if all of its two-transaction subsystems are safe, plus a combinatorial condition. This combinatorial condition turns out to be NP-complete, but it is simple enough to have some interesting corollaries. For example, all specific locking policies mentioned above can be shown to be safe as immediate consequences of the condition.

Versions of several of the results in this paper—in particular those in § 4—were independently obtained by Mihalis Yannakakis—see the first part of [Ya]. Both our results and those in [Ya] were announced in a joint extended abstract [YPK].

2. Definitions. A *transaction system* $\tau = \{T_1, \cdots, T_d\}$ is a set of *transactions*. A transaction $T_i = (T_{i1}, \cdots, T_{im_i})$ is a sequence of *actions* or *transaction steps*. Each action T_{ij} has associated with it an *entity*, $x_{ij} \in E$, where E is a set of entities. The x_{ij}'s need not be distinct.

Each action T_{ij} is thought of as the indivisible execution of the following:

$$T_{ij}: t_{ij} := x_{ij}$$

$$x_{ij} := f_{ij}(t_{i1}, \cdots, t_{ij}).$$

The first instruction stores the current value of x_{ij} to a local variable t_{ij}, not in E, and the second changes x_{ij} in the most general possible way based on all available local (to the transaction) information. The t_{ij}'s are all distinct. A *schedule* (or *history*) s of τ is a permutation of all steps of τ such that $j < k \leqq m_i$ implies $s(T_{ij}) < s(T_{ik})$. The set of all schedules is denoted by S. s is called *serial* if, for all i and $j < m_i$, $s(T_{ij}) + 1 = s(T_{i,j+1})$. Two schedules are *equivalent* if they are equivalent as parallel program schemata with uninterpreted f_{ij}'s; s is *serializable* (notation: $s \in SR$) if it is equivalent to some serial schedule.

Deciding serializability of a schedule s is known to be an NP-complete problem if we distinguish between reading and writing steps [Pa], [PBR], but can be easily done in our model of actions as follows [EGLT1]: Construct a digraph $D(s)$ by associating a node with each transaction T_i and drawing an arc (T_i, T_j) whenever, in the schedule s, T_i updates an entity before T_j does. Then s is serializable if and only if $D(s)$ is acyclic.

A *locked transaction system* $L(\tau)$ is a special augmented version of the (ordinary) transaction system τ. The operator L performing this augmentation is called a *locking policy*. A locking policy transforms each transaction of τ by inserting *lock* x and *unlock* x steps, $x \in E$, according to the following rules:

(1) For each entity x there is at most one *lock* x step in each transaction. If it exists, then there is also a unique subsequent *unlock* x step.

(2) Every access to an entity x is surrounded by a *lock-x–unlock-x* pair.

Let s be a schedule of $L(\tau)$. s is said to be *legal*—notation: $s \in G(L(\tau))$—if and only if between any two occurrences of *lock* x in s there is an occurrence of *unlock* x. Let L^{-1} be the operator that removes all lock-unlock steps; the set $L^{-1}(G(L(\tau))) - O(L)$ for short—is the *output set* of the locking policy L, and it captures the essential amount of parallelism supported by L.

We say that a locked transaction system $L(\tau)$ is *safe* if and only if any schedule in $O(L)$ is serializable. We say that it is *deadlock-free* if and only if for any legal *prefix* p of a schedule of $L(\tau)$, there is a *suffix* s such that $p \cdot s \in G(L(\tau))$.

Given a transaction system τ, there are certain well-known locking policies that can be applied to it. One is the *two-phase locking* (2PL) policy [EGLT1]. In it we insert locks surrounding the accesses of all entities in each transaction, subject to the following rule: The last entity to be locked is locked before the first entity is unlocked. Thus, a transaction is divided into two phases: the *locking phase*, during which locks are acquired but not released, and the *unlocking phase*, in which locks are released but not requested. Notice that this does not uniquely define a locking policy; it is in fact a *family* of locking policies. In an extremely conservative interpretation, we could lock all entities before the first step, and unlock them after the last. More reasonably, we could request for entities at the first step that they are accessed, and release locks at the end of the transaction. In fact, it is shown in [KP] that the latter interpretation of 2PL is the best possible concurrency control, when syntactic information is acquired in an incremental, dynamic manner (best in terms of the parallelism allowed). It was first shown in [EGLT1] that 2PL is safe (though not deadlock-free).

If the entities are *unstructured* (that is, transactions access them in all possible permutations of orders and patterns) then 2PL is the best possible locking policy. Suppose, however, that the entities form a tree, and are accessed by transactions as follows:

(a) A transaction accesses a subtree, whose root is the first entity to be accessed (after, of course, it is locked).

(b) After this, when an entity is locked, its parent must be locked and not yet unlocked.

Then this family of locking policies, called the *tree policy*, is shown in [SK] to be both safe and deadlock-free. This holds for the more general *digraph policy* of [Ya]. In fact, the latter is generalized in [Ya] to the *hypergraph policy* which, it is proved, is the most general possible safe policy.

In this paper we are only discussing what is known as the *exclusive* version of locking. There are, however, variants of locking in which locks of different kinds are defined (e.g., shared locks or read-locks, intention locks, etc.). Certain kinds of locks may coexist with others, whereas certain other kinds cannot. In [Pa3] we show that in some respects these more general kinds of locking behave in a way very similar to ordinary exclusive locks. It appears that the results of this paper can be quite easily generalized to these kinds of locks.

3. The geometry of locking. Consider a transaction system τ consisting of two transactions, T_1 and T_2. In the coordinate plane (Fig. 1) take the two axes to correspond to T_1 and T_2, and the integer points 1,2, etc., on these axes to correspond to the steps T_{11}, T_{12}, etc. (respectively T_{21}, T_{22}, etc.) of the transactions. A point p may present

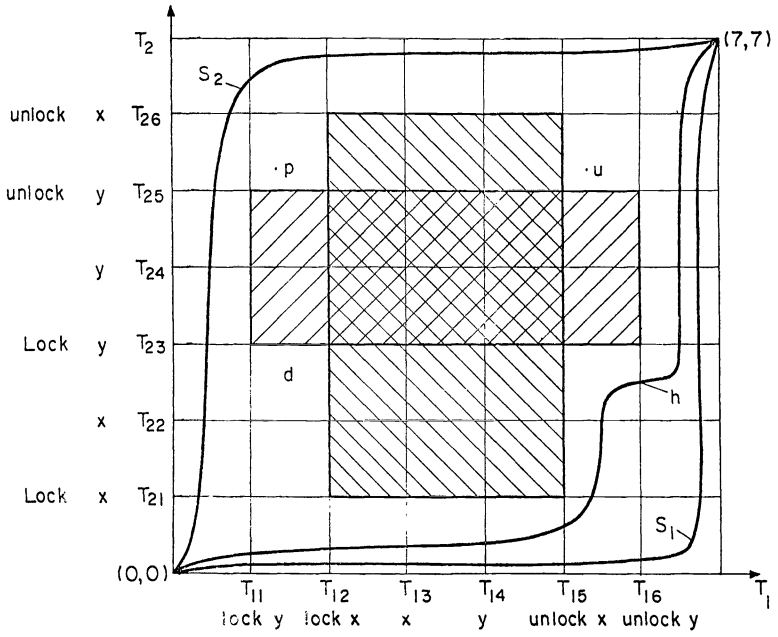


FIG. 1

a possible state of progress made toward the completion of T_1 and T_2. T_1 and T_2 will in general be *locked* transactions and will therefore contain properly nested lock-unlock steps. Each entity x such that both T_1 and T_2 contain a *lock-x–unlock-x* pair, has the effect of creating a *forbidden region* (a rectangle delimited by the grid lines corresponding to the four *lock x* or *unlock x* steps), the points of which represent unreachable states (see Fig. 1). Adding such rectangles to the plane has some consequences. For example, the point u is now unreachable, yet not in any rectangle; in contrast, point d is a state of *deadlock*.

What is the geometric image of a schedule? A schedule is a nondecreasing curve from the point (0, 0) to the point (m_1+1, m_2+1), not passing through any other grid point, nor through any rectangle (e.g., h in Fig. 1). (In fact, a schedule is more precisely

a *class* of such curves, all of which cross the same line segments of the grid.) To read the schedule off any such curve, we simply enumerate the grid lines that it intersects. For example, in Fig. 1 $h = T_{11}T_{12}T_{13}T_{14}T_{15}T_{21}T_{22}T_{16}T_{23}T_{24}T_{25}T_{26}$. The two serial schedules are represented by the curves s_1 and s_2 in Fig. 1.

Let h and h' be schedules such that $h = h_1S_1S_2h_2$, $h' = h_1S_2S_1h_2$, and S_1, S_2 are steps *not* updating the same variable. We then write $h \sim h'$. Let $\overset{*}{\sim}$ be the transitive-reflexive closure of $\sim$. The following fact has appeared many times in the literature [EGLT1], [PBR], [Pa1].

LEMMA 1. *Let h, h' be schedules. Then $h \equiv h'$ if and only if $h \overset{*}{\sim} h'$.*

Consider the point set $S = [0, m_1+1]\times[0, m_2+1]-R$, where R is the set of all forbidden rectangles. In other words, S is the relevant nonforbidden region of the plane. Let $\mathscr{C}$ be the set of all nondecreasing curves from $(0, 0)$ to (m_1+1, m_2+1) in S. We can partition these curves into *homotopy classes*. Two curves in $\mathscr{C}$ are *homotopic* if their union can be shrunk into a single point. In Figure 2, for example, h_1 and h_2 are homotopic, whereas h_1 and h_3 are not (since they "enclose" a forbidden rectangle). Homotopy is an equivalence relation in $\mathscr{C}$. Also, if two curves represent the same schedule, then they are homotopic.

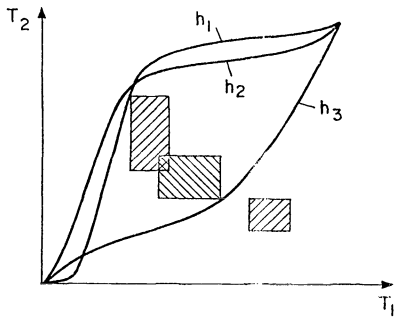


FIG. 2

LEMMA 2. *Two schedules are equivalent if and only if any two corresponding curves are homotopic.*

Proof. We first notice that condition (a) in our definition of locking has as a consequence that every forbidden rectangle contains a grid point (T_{1i}, T_{2j}) such that T_{1i} and T_{2j} update the same entity x; and conversely, any such point is surrounded by a forbidden rectangle. It follows that two curves in $\mathscr{C}$ are homotopic in $[0, m_1+1]\times[0, m_2+1]-R$ if and only if they are homotopic in $[0, m_1+1]\times[0, m_2+1]-P$, where P is the set of all points (T_{1i}, T_{2j}) such that T_{1i} and T_{2j} update the same entity. Now any two curves $h, h' \in \mathscr{C}$ are homotopic in $[0, m_1+1]\times[0, m_2+1]-P$ if and only if h can be transformed into h' via a continuous transformation, avoiding all points in P. Such a continuous transformation can be broken down into finitely many transformations of one of the following two types:

(a) transformations that do not change the schedule represented by the curve (Fig. 3a),

(b) transformations in which the curve crosses a grid point *not in P*. (Fig. 3b.)

Type (a) leaves the corresponding schedule unchanged. Also, type (b) changes the schedule h into a h' such that $h \sim h'$. Therefore, we have from the above discussion that two curves are homotopic if and only if the corresponding schedules satisfy $h'^* \sim h$; or, by Lemma 1, if and only if $h \equiv h'$. □

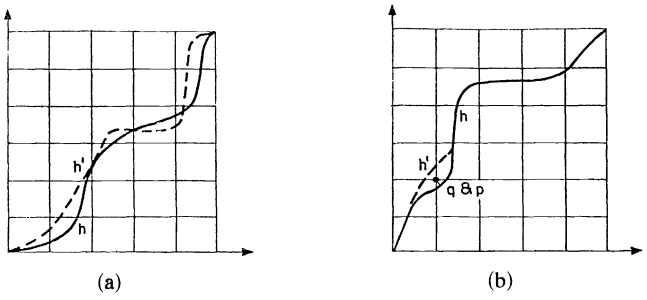


FIG. 3

Since the two serial schedules are s_1 and s_2 of Fig. 1, we conclude from Lemma 2 that nonserializable schedules are exactly those that are homotopic to neither.

THEOREM 1. *A schedule is not serializable if and only if the corresponding curve separates two rectangles.*

Thus h_1 and h_2 in Fig. 2 are serializable schedules, whereas h_3 is not. Hence τ is unsafe.

Certain further geometric concepts help illuminate the concept of *safety* of two-transaction systems. Let R be any subset of the plane, possibly disconnected. We call two points (x_1, y_1) and (x_2, y_2) in the plane *incomparable* if $(x_1 - x_2) \cdot (y_1 - y_2) < 0$ (points p and q in Fig. 4). Then R is said to be *closed* if, for any two incomparable points (x_1, y_1) and (x_2, y_2) that are connected in R, the points (x_1, y_2) and (x_2, y_1) are also in R. The *closure* of R is the smallest closed region that contains R. Notice that closure (R) is always well-defined (see Fig. 4).

LEMMA 3. *If a connected rectilinear*[1] *region R is closed, then R is the region contained between two increasing curves that intersect only at their endpoints.*

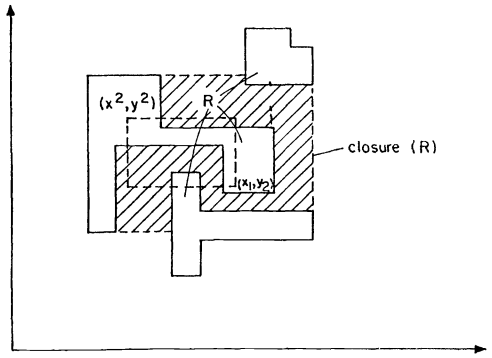
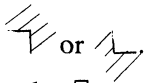


FIG. 4

Proof. The boundary of a rectilinear region R is *not* as in the lemma only if it contains a fragment like



In both cases closedness is contradicted. □

We make here the following helpful observation: The forbidden rectangles corresponding to two locked transactions have the property that each of their edges has a unique ordinate or abscissa. As a consequence, the same is true for the rectangles

[1] A region is called *rectilinear* if it is the union of rectangles with edges parallel to the axes.

comprising the closure of the forbidden region, as the closure of a rectilinear region has rectangles with ordinates and abscissas among those of the original rectangles. A helpful corollary is that components of the closure may not accidentally "touch" at a line, or just a point. If they are disconnected, they are clearly divided by corridors of width one. Therefore, a nondecreasing curve avoiding all components can always be turned to one that avoids all grid points (except $(0, 0)$ and (m_1+1, m_2+1)).

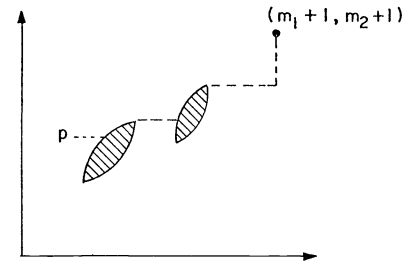


FIG. 5

LEMMA 4. *Let R be a closed (possibly disconnected) rectilinear subset of $[0, m_1+1]\times[0, m_2+1]$, and let p be a point in $[0, m_1+1]\times[0, m_2+1]-R$. Then there is a nondecreasing curve from $(0, 0)$ to p and from p to (m_1+1, m_2+1) in $[0, m_1+1]\times[0, m_2+1]-R$.*

Proof. We shall only prove the second claim, as the first is completely symmetric. From p we first go parallel to the x-axis to increasing x's, see Fig. 5. If we do not "hit" a component of R, we are done; otherwise, we follow the nondecreasing curve in the boundary of the component (Lemma 3) and when it ends we again proceed parallel to the x-axis. It is clear that we shall eventually reach the $x = m_1+1$ line. □

THEOREM 2. *τ is safe if and only if the closure of the union of the forbidden rectangles is connected.*

Proof. Suppose that τ is unsafe. Then, by Theorem 1, there is an increasing curve h from $(0, 0)$ to (m_1+1, m_2+1) which separates two rectangles r_1 and r_2. Consider the region that consists of all grid squares not touched by h. This region is closed, contains all rectangles and is disconnected, and each of its components contains a rectangle. It follows that the closure of the forbidden region is the union of two nonempty subsets of the two components, and thus it is disconnected.

For the "if" direction, suppose that the closure is disconnected. There are two cases. Either there is a vertical line which hits two components of the closure, or there is not. In the second case (Fig. 6a) there is a vertical line which *separates* two components—and thus a nondecreasing curve, consisting of two horizontal and one vertical

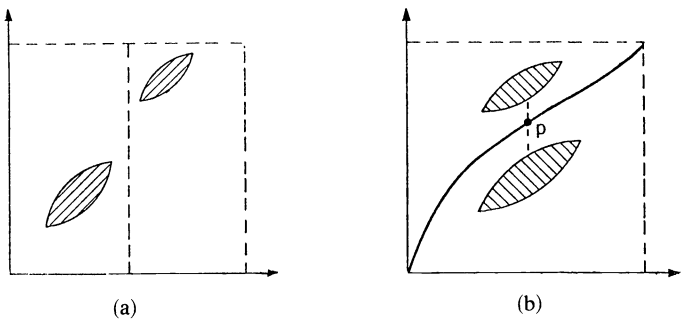


FIG. 6

line segments, which also separates two components. In the first case, (Fig. 6b) consider this line, and a point p on it that is in between the two components. By Lemma 4, there is a nondecreasing curve from $(0, 0)$ to p and one from p to (m_1+1, m_2+1). Their union is a nondecreasing curve that separates the two components. □

In a rectilinear region R we can define the *lower-left boundary*, LLB (R), to be the union of all horizontal segments that are lower boundaries of R, together with all vertical segments that are left boundaries of R (that is, LLB (R) is the set of all points (x, y) such that $(x+\delta, y+\varepsilon) \in R$ whereas $(x-\delta, y-\varepsilon) \notin R$, for all $\delta, \varepsilon > 0$). The proof of the following result is now straight forward.

THEOREM 3. *τ is deadlock-free if and only if* LLB (R) $\supseteq$ LLB (*closure* (R)), *where R is the union of the forbidden rectangles.*

Theorems 2 and 3 lead to fast algorithms for the solution of the safety and deadlock problems for locked transaction systems [LP]. There are other insights that are offered by this geometric viewpoint. For example, the correctness of 2PL has now become very intuitive. 2PL says that all rectangles must contain the point p whose projections p_1 and p_2 mark the phase-shift points of the two transactions (see Fig. 7). Thus the rectangles are certainly connected, so is their closure, and the correctness of 2PL follows immediately from Theorem 2.

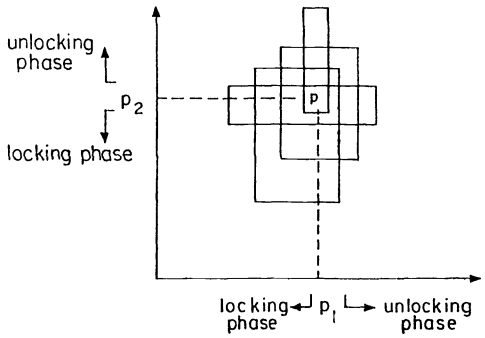


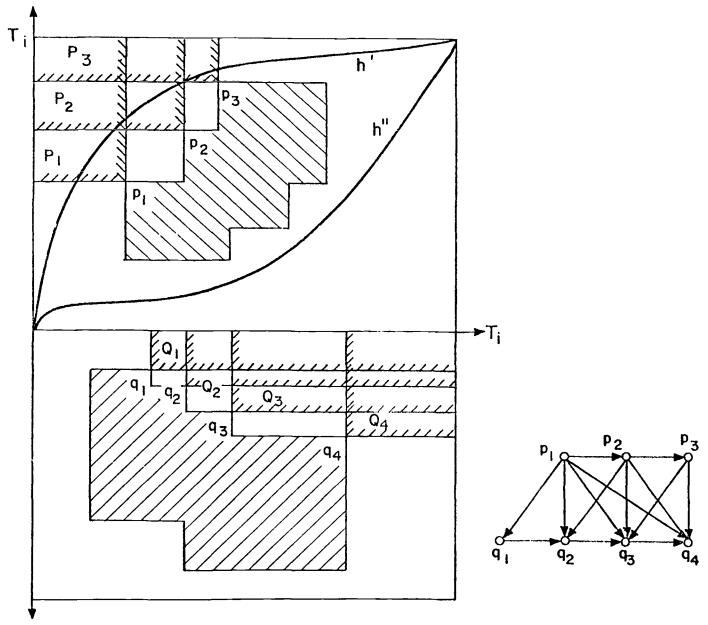
FIG. 7

4. More than two transactions. Consider now a set of $d > 2$ locked transactions $\tau = \{T_1, \cdots, T_d\}$. We wish to study the problem of safety of τ. If any subset of τ with two transactions is unsafe, then clearly so is τ. Define the graph $G(\tau) = (\tau, A)$ with transactions as nodes, and with $[T_i, T_j] \in A$ if and only if T_i and T_j update a common entity. From our assumption above it follows that, for any $[T_i, T_j] \in A$, the closure of the forbidden region on the (T_i, T_j)-plane is connected. Therefore, any schedule h of τ will have a projection on the (T_i, T_j)-plane that is either equivalent to T_iT_j or equivalent to T_jT_i (see Fig. 8). In the first case we write $T_i <_h T_j$, and in the latter $T_j <_h T_i$.

LEMMA 5. *h is serializable if and only if $(\tau, <_h)$ is acyclic.*

Proof. The lemma follows by observing that $(\tau, <_h)$ is exactly the digraph constructed for testing the serializability of h in § 2. □.

Therefore, for τ to be safe, for each directed cycle that corresponds to an undirected cycle in $G(\tau)$ there must be a reason why no h exists that has this same cycle in the graph of $<_h$. Intuitively, the reason is that there is a contradiction in the order in which the curve of h intersects the prisms with bases marked $P_1, P_2, \cdots, Q_1, Q_2$ in Fig. 8. This is captured as follows: With each pair $([T_i, T_j], [T_j, T_k])$ of edges in A we associated a digraph B_{ijk}. The vertices of this



(a) (b)

FIG. 8

digraph are the vertices p_m, q_n of the P_m and Q_n regions (see Fig. 8). There is an arc from u to v if and only if (a) either u is a p_m or v is a q_n (or both), and (b) the T_j-coordinate of u is smaller than the T_j-coordinate of v. The construction is illustrated in Fig. 8b. Finally, if C is a directed cycle corresponding to a simple undirected cycle in $G(\tau)$, we let B_C be the union of all B_{ijk} digraphs for all consecutive triples (T_i, T_j, T_k) of C. The result is the following:

THEOREM 4. *τ is safe if and only if*

(a) *the restriction of τ to any two transactions is safe, and*

(b) *for all directed cycles C corresponding to undirected simple cycles of $G(\tau)$, the digraph B_C has a cycle.*

Proof. We shall assume throughout the proof that condition (a) holds, as otherwise both directions are easy. Suppose that τ is unsafe, and yet (b) holds. Then there is a schedule h such that $<_h$ has a simple cycle C. By condition (b), B_C has a cycle. Consider, however, the order whereby h enters (equivalently, leaves) the different prisms P_i, Q_j, etc., appearing in B_C. This order must be, by the construction of B_C, consistent with the arcs in B_C. This, however, is a contradiction, since B_C was assumed to have a cycle, and therefore to be inconsistent with any linear order.

Conversely, suppose that (b) does not hold; suppose, that is, that there is a directed cycle C such that B_C is acyclic. Let $(p_1, q_1, \cdots)$ be a linear order consistent with B_C. We construct a nonserializable schedule h of τ as follows: We first arrange all transactions not appearing in C in some serial order. We then execute the transactions in C by starting with all program counters to 0, and by successively considering the next point in the order. If this point lies on the (T_i, T_j)-plane, where $(T_i, T_j) \in C$, then we proceed by arranging the steps of T_j and then the steps of T_i that bring the state to the point considered. Since the order is consistent with B_C, this will always be possible; after the last point we somehow schedule all remaining steps. It

is easy to see that, if h is the resulting schedule, then $(\tau, <_h)$ contains the cycle C, and therefore h is not serializable. □

Based on Theorem 4, we obtain extremely easy proofs of correctness of several locking policies:

COROLLARY 1. *Any transaction system obeying* 2PL *is safe.*

Proof. That the two-transaction subsystems of any transaction system that obeys 2PL is safe follows trivially from Theorem 2, as discussed in the end of the previous section. Property (b) of Theorem 4 follows from the fact in 2PL the graphs B_{ijk} are complete bipartite. □

COROLLARY 2. *Any transaction system obeying TP is safe.*

Proof. Since the common variables of any two transactions form a rooted tree in *TP*, condition (a) of Theorem 4 is trivial. Condition (b) also follows easily from the tree structure. □

Consider now the special case in which any two transactions have at most one variable in common—or, equivalently, the closure of the forbidden region on all planes is either empty or rectangular.

COROLLARY 3. *Under the above assumption τ is safe if and only if each of the restrictions of τ to every biconnected component of $G(\tau)$ obeys* 2PL.

Another consequence of Theorem 4 is an algorithm for checking a transaction system for safety.

COROLLARY 4. *Checking a transaction system τ for safety can be done in time polynomial in the number of minimal cycles of $G(\tau)$.*

In general, of course, $G(\tau)$ will have an exponential number of minimal cycles, and thus Corollary 7 does not imply a genuine polynomial-time algorithm. In fact, such an algorithm is quite unlikely in view of the next result:

THEOREM 5. *Testing a transaction system for nonsafety is* NP*-complete.*

Proof. Recently Fortune, Hopcroft and Wyllie [FHW] showed that almost all digraph homeomorphism problems—specifically, those whose pattern is not a rooted tree of depth one—are NP-complete. From their results one immediately deduces that the following problem is NP-complete:

"Given a digraph $D = (V, A)$ and four arcs (v_1, v_2), (v_2, v_3), (u_1, u_2), $(u_2, u_3) \in A$, is there a simple cycle C of D such that $C = (v_1, v_2, v_3, \cdots, u_1, u_2, u_3, \cdots, v_1)$?"

We shall reduce this problem to the nonsafety problem. Given a digraph $D = (V, A)$ and four arcs (v_1, v_2), (v_2, v_3), (u_1, u_2), (u_2, u_3) we shall construct a locked transaction system τ such that C exists in D if and only if τ is unsafe. τ has one transaction T_v for each node v of D. Let us assume that for no nodes $u, v \in V$, does A contain both (u, v) and (v, u)—that is, $A \cap A^{-1} = \varnothing$ (it is easily seen that this is not a loss of generality, since a new node can always be placed in the middle of an arc). We shall next describe the graphs B_{uvw} for (u, v), $(v, w) \in A$. Unless $(u, v, w) = (u_1, u_2, u_3)$ or (v_1, v_2, v_3), B_{uvw} is the graph shown in Fig. 9a, realized as shown in Fig. 9b. For these two triples, B_{uvw} is as shown in Fig. 9c, realized as in Fig. 9d.

Consider now any simple cycle in D. The corresponding union of the B_{uvw}'s will consist of concatenations of graphs such as in Figs. 9a and 9c. It is easy to see that such a concatenation is acyclic only if it contains at least two copies of Fig. 9c, that is, only if C passes through (v_1, v_2, v_3) and (u_1, u_2, u_3). Conversely, consider any directed cycle C, whose undirected version is a simple cycle in $G(\tau)$, such that B_C is acyclic. If, however, any arc (u, v) of C is not in A—that is, $(v, u) \in A$—then B_C will contain two complete bipartite graphs corresponding to (u, v), its predecessors and its successor edge in C (by the construction in Fig. 9b, d). It follows easily that B_C could not be acyclic. Thus all arcs in C are also in A, and in fact, since B_C was acyclic, all four (v_1, v_2), (v_2, v_3), (u_1, u_2), and (u_2, u_3) are in C. The theorem follows. □

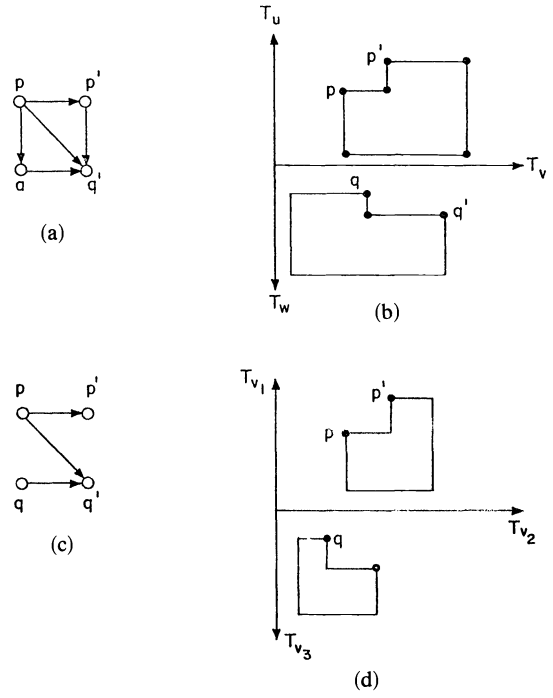


FIG. 9

Consequently, we can view Corollary 3 as suggesting an efficient algorithm for a special case of the NP-complete problem of nonsafety—namely, the case in which any two transactions interact by at most one entity. The result is tight, since the proof of Theorem 5 (see Figs. 9b, d) suggests that the problem remains NP-complete when two entities per pair of transactions are allowed.

REFERENCES

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

[BGRP] P. A. BERNSTEIN, M. GOODMAN, J. B. ROTHNIE AND C. H. PAPADIMITRIOU, *Analysis of Serializability of SSD-1: A system of distributed databases (The fully redundant case)*, IEEE Trans. Software Engng, SE-4 (1978), pp. 154–168.

[CD] E. G. COFFMAN, JR. AND P. J. DENNING, *Operating Systems Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

[CES] E. G. COFFMAN, JR., M. J. ELPHICK AND A. SHOSHANI, *Systems deadlock*, Comput. Surveys, 3 (1971), pp. 67–68.

[EGLT1] K. P. ESWARAN, J. N. GRAY, R. A. LORIE AND I. L. TRAIGER, *The notions of consistency and predicate locks in a database system*, Comm. ACM, 19 (1976), pp. 624–633.

[EGLT2] ———, *On the notions of consistency and predicate locks*, IBM Research Report RJ 1487, 1974.

[FHW] S. FORTUNE, J. E. HOPCROFT AND J. WYLLIE, *The directed subgraph homeomorphism problem*, Theoret. Comput. Sci. 10 (1980), pp. 111–121.

[GJ] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Fransisco, 1978.

[GLPT] J. N. GRAY, R. A. LORIE, G. R. PUTZOLU AND I. L. TRAIGER, *Granularity of locks and degrees of consistency in a shared data base*, IBM Research Report RJ 1654, 1975.

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

[KM] R. M. KARP AND R. E. MILLER, *Properties of a model for parallel computations: Determinacy, termination and queuing*, SIAM J. Appl. Math., 14 (1966), pp. 1390–1410.

[KP] H. T. KUNG AND C. H. PAPADIMITRIOU, *An optimality theory of concurrency control for databases*, Proc. ACM-SIGMOD Conference, 1979,

[LW] Y. E. LIEN AND P. H. WEINBERGER *Consistency, concurrency, and crash recovery*, Proc. ACM-SIGMOD Conference, 1978,

[LP] W. LIPSKI, JR. AND C. H. PAPADIMITRIOU, *A fast algorithm for testing for safety and deadlocks in locked transaction systems*, J. Algorithms, 2 (1981), pp. 211–226.

[Pa1] C. H. PAPADIMITRIOU, *Serializability of concurrent updates*, J. Assoc. Comput. Mach., 26 (1979), pp. 631–653.

[Pa2] ———, *The Theory of Database Concurrency Control*, monograph, in preparation.

[Pa3] ———, *On the power of locking*, Proc. 1981 SIGMOD Conference

[PBR] C. H. PAPADIMITRIOU, P. A. BERNSTEIN AND J. B. ROTHNIE, *Computational problems related to database concurrency control*, Conference on Theoretical Computer Science, University of Waterloo, Ontario, 1977, pp. 275–282.

[SK] A. SILBERCHATZ AND Z. KEDEM, *Consistency in hierarchical database systems*, J. Assoc. Comput. Mach., 27 (1980), pp. 72–80.

[SLR] R. E. STEARNS, P. M. LEWIS AND D. J. ROSENKRANTZ, *Concurrency control for database systems*, in Proc. 17th Annual Symposium on Foundations of Computer Science, 1976, pp. 19–32.

[Ya] M. YANNAKAKIS, *A theory of safe locking policies in database systems*, J. Assoc. Comput. Mach., to appear.

[YPK] M. YANNAKAKIS, C. H. PAPADIMITRIOU AND H. T. KUNG, *Locking policies: Safety and freedom from deadlock*, Proc. 20th ACM Conference on Foundations of Computer Science, 1979, pp. 283–287.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0002 $01.25/0

DYNAMIC BIN PACKING*

E. G. COFFMAN, JR.†, M. R. GAREY† AND D. S. JOHNSON†

Abstract. Motivated by potential applications to computer storage allocation, we generalize the classical one-dimensional bin packing model to include dynamic arrivals and departures of items over time. Within this setting, we prove close upper and lower bounds on the worst-case performance of the commonly used First Fit packing algorithm, and, using adversary-type arguments, we show that *no* on-line packing algorithm can satisfy a substantially better performance bound than that for First Fit.

Key words. approximation algorithms, memory management, performance evaluation, storage allocation, worst-case bounds

1. Introduction.

One-dimensional bin packing plays an important role in a large variety of combinatorial problems arising in operations research and computer science. The classical model [4] assumes an unbounded collection of equal-capacity bins and a given set of items, each of which has a size no larger than the common bin capacity. The optimization problem, which is easily seen to be NP-hard [2], is to pack the given items into as few bins as possible, subject to the requirement that the items in each bin sum to no more than the bin capacity. Most work on this problem and its many variants (e.g., see [3]) has concentrated on proving close bounds on the worst-case performance of simple "approximation algorithms" that might be expected to construct near-optimal packings.

In certain potential applications, such as those relating to computer storage allocation, the classical model fails to be realistic in that dynamic arrivals and departures of items are not considered; that is, a model is needed in which items arrive over time, reside for varying amounts of time in the bins to which they are assigned, and then depart from the packing. The contribution of this paper is the formulation of such a "dynamic" bin packing model and the analysis of some approximation algorithms within this context. A summary of our results can be found at the end of § 2.

As mentioned, a principal motivation for our model is dynamic storage allocation in computer systems. Here the items are records or files, and the bins are storage units, such as disk cylinders, limited by the property that records cannot effectively be "overlapped" from one unit to the next. The problem we address through our model is that of how to distribute records among storage units so that at all times each unit will have sufficient space to hold all the records assigned to it. We explicitly assume that a record can always be packed in any storage unit that has sufficient total space available for it, and hence we do not consider the related problem of how one manages space *within* a storage unit in order to avoid fragmentation or "checkerboarding" [5] that might prevent a record from fitting even though the total space available would be sufficient for it.

2. The model.

Our model is a natural extension of the classical static one-dimensional bin packing model. The items to be packed will be described by a finite sequence or *list* $L = (p_1, p_2, \cdots, p_n)$. Each *item* (or piece) p_i in L corresponds to a triple (a_i, d_i, s_i), where a_i is the arrival time for p_i, d_i is its departure time, and s_i is its size. The item p_i resides in the packing for the time interval $[a_i, d_i)$, and we assume

* Received by the editors July 29, 1981, and in revised form May 27, 1982.
† Bell Laboratories, Murray Hill, New Jersey 07974.

that $d_i - a_i > 0$ for all i. Without loss of generality, the common bin capacity will be taken always to be 1, so we also assume that each s_i satisfies $0 < s_i \leqq 1$. Finally, we shall assume that the items in L are ordered so that $a_1 \leqq a_2 \leqq \cdots \leqq a_n$.

Motivated by applications such as dynamic storage allocation, we shall restrict our attention to packing rules that do not move items from one bin to another once they have been packed, and that operate *on-line*, i.e., that pack items as they arrive without any knowledge of future arrivals. Thus such an algorithm will assign the items in L to bins in order of increasing index, under the single constraint that at each time t there be no bin that contains "currently active" items whose sizes sum to more than 1. We use L_t to denote the sublist of items in L that are active at time t (i.e., for which $a_i \leqq t < d_i$).

Given this setting, it is natural that a First Fit (FF) packing rule be of central interest. The FF rule to be analyzed maintains two lists of bins: a list of currently empty bins and a list of currently occupied bins, with the latter ordered so that the times of most recent transition from empty to nonempty are nondecreasing. We use $B_1, B_2, \cdots$ to denote the bins in the occupied-bin list, in this order, and l_i to denote the *level* (sum of item sizes) for B_i, taking care always to ensure that the time of definition is clear from context.

For each i, the FF rule attempts to pack p_i at time a_i into the first occupied bin (one with lowest index) that has sufficient available space for p_i (i.e., whose level l_j satisfies $1 - l_j \geqq s_i$). If no such occupied bin exists, p_i is placed into an empty bin and that bin is appended to the list of occupied bins. The departure of p_i at time d_i simply causes an increase by the amount s_i in the available space in the bin in which it was packed, and, if the bin becomes empty at that time, it is moved to the list of empty bins. It is readily seen that the ordering for occupied bins described above is maintained by this procedure.

Let FF (L, t) denote the number of occupied bins in the FF packing of L at time t. Our measure of performance for the FF rule applied to L is defined by

$$\mathrm{FF}(L) = \max_{0 \leqq t \leqq a_n} \mathrm{FF}(L, t).$$

In words, FF (L) is the maximum number of occupied bins ever required by FF in processing L. This definition and notation can be extended to an arbitrary packing algorithm A simply by replacing FF with A.

An example illustrating the application of FF is shown in Fig. 1. Let m be an integer divisible by 6, and let $\varepsilon = 1/2m$. The list L_m is described as follows: First, m items of size $\frac{2}{3} - \varepsilon$, followed by m items of size $\frac{1}{3} - \varepsilon$, followed by m items of size 2ε arrive. These are packed by FF as shown in Fig. 1(a). Then, all the items of size $\frac{1}{3} - \varepsilon$ depart, and a sequence of m items of sizes $\frac{1}{3}, \frac{1}{3} + \varepsilon, \frac{1}{3}, \frac{1}{3} + \varepsilon, \cdots, \frac{1}{3}, \frac{1}{3} + \varepsilon$ arrives. Figure 1(b) shows the FF packing at this point, with the number of nonempty bins having increased from m to $3m/2$. Finally, all the items of size $\frac{1}{3} + \varepsilon$ and $\frac{2}{3} - \varepsilon$ depart, and $5m/6$ items of size 1 arrive. The final FF packing, now with $7m/3$ nonempty bins, is shown in Fig. 1(c). Thus FF $(L_m) = 7m/3$, with the maximum being achieved when the last item arrives.

We shall be comparing FF (L) (or, in general, $A(L)$) with two different measures of how tightly the items in L can be packed. The first, $\mathrm{OPT}_R(L)$, is the maximum number of bins ever required in dynamically packing L when the current set of items is *repacked* into the minimum possible number of bins each time a new item arrives. The second, $\mathrm{OPT}_{NR}(L)$, is the maximum number of bins ever required in dynamically packing L when *no rearrangement* of items is allowed but the items are otherwise

packed optimally, i.e., so as to achieve the least possible value of this maximum over all such packings of L, with L assumed to be fixed and known in advance. (For $\mathrm{OPT}_R(L)$ it is irrelevant whether or not L is known in advance.) Clearly $\mathrm{OPT}_R(L) \leqq \mathrm{OPT}_{NR}(L)$ for all lists L. For the example given above, it is not difficult to see that $\mathrm{OPT}_R(L_m) = m+1$ and for m divisible by 9, $\mathrm{OPT}_{NR}(L_m) = (10m/9)+1$.

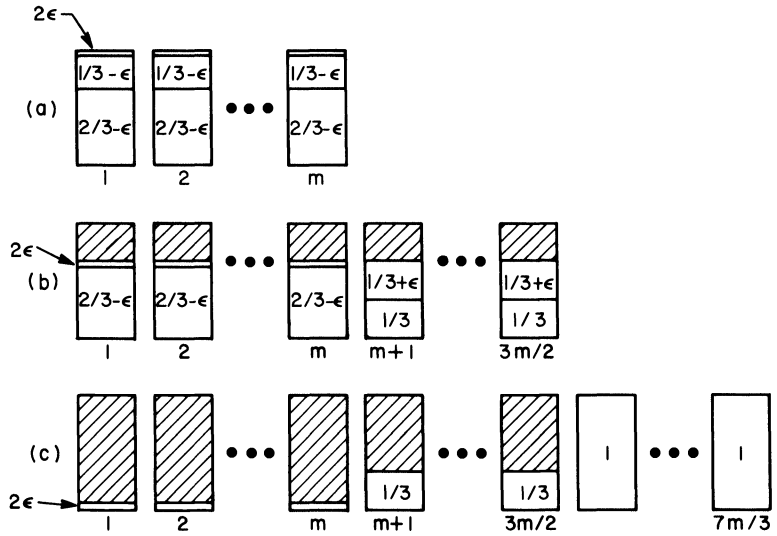


FIG. 1. *An illustration of the application of* FF *for dynamic bin packing.*

As in earlier bin packing studies, our specific comparisons will focus on asymptotic values of certain "performance ratios." For an arbitrary packing algorithm A, let

$$W_{R,n}(A) = \sup_{|L|=n} \frac{A(L)}{\mathrm{OPT}_R(L)}$$

where the supremum is taken over all sequences of n triples (a_i, d_i, s_i) satisfying $a_i \geqq 0$, $d_i > a_i$, $0 < s_i \leqq 1$, for $1 \leqq i \leqq n$, and $a_1 \leqq a_2 \leqq \cdots \leqq a_n$. We then define

$$W_R(A) = \limsup_{n\to\infty} W_{R,n}(A).$$

Similar definitions apply to $W_{NR}(A)$. Note that the lists L_m in our example show that $W_R(\mathrm{FF}) \geqq 7/3$ and that $W_{NR}(\mathrm{FF}) \geqq 21/10$.

The above notation will be extended in a natural way to account for situations in which there is an upper bound on maximum item size. In particular, we use $W_R(A, k)$ to denote the asymptotic bound for algorithm A in the case that all item sizes s_i satisfy $s_i \leqq 1/k$, k an integer. Clearly $W_R(A) = W_R(A, 1)$.

A standard approach to obtaining bounds on the quantity W_R has been to prove the somewhat stronger result that for all L

$$A(L) \leqq \alpha \cdot \mathrm{OPT}_R(L) + \beta$$

for constants α and β. Then $W_R(A)$ is bounded above by the multiplicative constant α. This will also be our approach, although we shall omit the rather tedious derivations of the less interesting additive constants β, primarily in order to keep the paper down to manageable size.

The organization of the remainder of the paper and an outline of the main results are as follows:

In § 3 we derive upper bounds for $W_R(\mathrm{FF}, k)$. Specifically, we show that

$$W_R(\mathrm{FF}, k) \leqq \frac{k+1}{k} + \frac{1}{k-1} \log \frac{k^2}{k^2-k+1}, \qquad k \geqq 2,$$

$$W_R(\mathrm{FF}, 1) \leqq \frac{5}{2} + \frac{3}{2} \log \frac{\sqrt{13}-1}{2} \approx 2.897.$$

(All logarithms in this paper are to base e.) We also describe a slightly modified version of FF for which W_R is bounded above by $2.788\cdots$.

In § 4, we derive the following lower bounds for $W_R(\mathrm{FF}, k)$:

$$W_R(\mathrm{FF}, k) \geqq \frac{k+1}{k} + \frac{1}{k^2}, \qquad k \geqq 2,$$

$$W_R(\mathrm{FF}, 1) \geqq \frac{11}{4} = 2.750.$$

The lower bound of 2.750 holds also for our modified version of FF.

In § 5 we go beyond the analysis of the single algorithm FF and prove lower bounds on W_R that hold for *arbitrary* on-line packing algorithms. In particular, we show that for any on-line algorithm A

$$W_R(A, k) \geqq W_{NR}(A, k) \geqq \frac{k+1}{k} + \frac{1}{k(k+1)}, \qquad k \geqq 2,$$

$$W_R(A, 1) \geqq \frac{5}{2},$$

$$W_{NR}(A, 1) \geqq \frac{43}{18} \sim 2.388\cdots,$$

showing that the corresponding bounds for FF and our modified version of FF are not too far from the best that one could hope for (even if comparison to the *nonrearranged* optimum is more appropriate).

Section 6 then concludes the paper with a brief discussion of our results and some directions for further research.

3. Upper bounds for First Fit. Since the proofs of the result for $k = 1$ (the general case) and $k \geqq 2$ differ substantially, we shall work out the results separately.

THEOREM 1. *For* $k \geqq 2$,

$$W_R(\mathrm{FF}, k) \leqq \frac{k+1}{k} + \frac{1}{k-1} \log \frac{k^2}{k^2-k+1}.$$

Proof. We shall show for any list $L = (p_1, p_2, \cdots, p_n)$ satisfying $s_i \leqq 1/k$, $1 \leqq i \leqq n$, that

$$\mathrm{FF}(L) \leqq \alpha \cdot \mathrm{OPT}_R(L) + O(1)$$

where α is the upper bound given in the theorem statement and the constant represented by $O(1)$ is independent of L and n.

We begin by observing that we can restrict our attention to lists L satisfying two special properties. The first property is that

$$\mathrm{FF}(L) = \mathrm{FF}(L, a_n) > \mathrm{FF}(L, t), \qquad 0 \leqq t < a_n,$$

i.e., that the maximum number of bins is used when the last item is packed and not before. This is justified by noting that, if i is the least index such that $\mathrm{FF}(L, a_i) = \mathrm{FF}(L)$, then the truncated list L' consisting of only the first i items in L satisfies $\mathrm{FF}(L') = \mathrm{FF}(L)$ and $\mathrm{OPT}_R(L') \leqq \mathrm{OPT}_R(L)$, and hence must violate the desired bound whenever L does.

The second property is that no occupied bin ever becomes empty during the process of applying FF to L. This is justified by noting that, if some occupied bin becomes empty during the FF packing of L, say at time t, then the list L' formed from L by deleting all items that were ever assigned to that bin prior to time t will yield exactly the same final packing as L, which implies by our first property that $\mathrm{FF}(L') = \mathrm{FF}(L)$, and will satisfy $\mathrm{OPT}_R(L') \leqq \mathrm{OPT}_R(L)$. Thus again it is the case that L' will violate the desired bound whenever L does.

Therefore, without loss of generality, let $L = (p_1, p_2, \cdots, p_n)$ be any list satisfying the above two properties and having $s_i \leqq 1/k$ for all i, $1 \leqq i \leqq n$. Let $m = \mathrm{OPT}_R(L)$. Notice that the second of our two properties implies that the index of an occupied bin, determined from its position on the occupied-bin list, never changes during the FF packing of L, so there will be no confusion when we refer to bin i (or B_i) without reference to a specific time. We shall proceed by deriving lower bounds on the size of the smallest item that FF can possibly place in bin B_i and on the level l_i required in B_i whenever an item of size s is packed into some bin B_j with $j > i$. These bounds will then be used in conjunction with the requirement that no item size exceed $1/k$ and the fact that the sum of all item sizes is at most $\mathrm{OPT}_R(L) = m$ to obtain an upper bound on $\mathrm{FF}(L)$.

Let h_j denote the size of the smallest item that FF ever places in bin B_j, and suppose that such an item p is placed in B_j at time t. Then at that time the levels of all lower-indexed bins must exceed $1 - h_j$, or FF would have placed p in one of them instead. Furthermore, since no item size exceeds $1/k$, every such preceding bin B_i must contain at least k items (or p would fit there as a kth item) and hence the level of B_i must also be at least $k \cdot h_i$. Thus, at the time p is packed, we must have

$$l_i \geqq \max\{1 - h_j, k \cdot h_i\}, \qquad i < j. \tag{1}$$

In particular, using just the fact that $l_i > 1 - h_j$ and the fact that the sum of all item sizes cannot exceed m, we obtain

$$(j-1)(1-h_j) + h_j < m,$$

which implies, for $j > 2$,

$$h_j > 1 - \frac{m-1}{j-2}.$$

Setting j equal to i in this inequality and substituting into (1), we conclude that when p is packed into B_j,

$$l_i \geqq \max\left\{1 - h_j, k \cdot \left\{1 - \left(\frac{m-1}{i-2}\right)\right\}\right\}, \qquad i < j. \tag{2}$$

We now consider the situation in more detail when $j > m(k+1)/k$ (the theorem holds trivially when there is no such j). In this case, at the time a piece of size h_j is

placed in B_j, we must have

$$\sum_{i=1}^{m(k+1)/k} l_i + \sum_{i=1+m(k+1)/k}^{j-1} l_i + h_j \leqq m. \tag{3}$$

Using (2) to bound the first summation, we obtain

$$\sum_{i=1}^{m(k+1)/k} l_i \geqq \sum_{i=1}^{\lambda} (1-h_i) + k \sum_{i=\lambda+1}^{m(k+1)/k} \left(1-\frac{m-1}{i-2}\right), \tag{4}$$

where

$$\lambda = \left\lfloor \frac{k(m-1)}{k-1+h_j} + 2 \right\rfloor \tag{5}$$

is the dividing point between those values of i for which $1-h_i$ gives the maximum and those values of i for which $k \cdot (1-(m-1)/(i-2))$ gives the maximum. Notice that $\lambda < m(k+1)/k$, since $j \geqq 1+m(k+1)/k$ implies $h_j > 1/(k+1)$. Simplifying (4) and using the conventional logarithmic approximation for the harmonic numbers, we have

$$\sum_{i=1}^{m(k+1)/k} l_i \geqq \lambda(1-h_j) + k\left(\frac{m(k+1)}{k} - \lambda\right) - k \cdot (m-1) \cdot \log \frac{(k+1)m}{k(\lambda-2)} + O(1).$$

Substituting (5) into the above and applying routine manipulations, we eventually arrive at

$$\sum_{i=1}^{m(k+1)/k} l_i \geqq m - km \log\left(\frac{k+1}{k^2}h_j + \frac{k^2-1}{k^2}\right) + O(1).$$

Substituting this expression back into (3) and replacing l_i, $i > m(k+1)/k$, by the lower bound $k \cdot h_i$, we finally have

$$\sum_{i=1+m(k+1)/k}^{j-1} h_i \leqq m \log\left(\frac{k+1}{k^2}h_j + \frac{k^2-1}{k^2}\right) + O(1). \tag{6}$$

Inequality (6) is the key to completing the proof. It implicitly provides a lower bound for every h_j, $j > m(k+1)/k$. We shall use these lower bounds to obtain a lower bound on the sum of the h_i's for $i < j$, which in turn will be used to determine an upper bound on the largest j for which h_j can be less than or equal to $1/k$, the maximum piece size allowed. That bound on j will then be our desired bound for FF (L).

To do this, let the function $h(j)$ be defined only at integer points $j \geqq m(k+1)/k$ by $h(m(k+1)/k) = 1/(k+1)$ and for $j > m(k+1)/k$

$$\sum_{i=m(k+1)/k}^{j-1} h(i) = m \log\left(\frac{k+1}{k^2}h(j) + \frac{k^2-1}{k^2}\right). \tag{7}$$

It is not difficult to see that

$$\sum_{i=m(k+1)/k}^{j-1} h(i) \leqq \sum_{i=1+m(k+1)/k}^{j-1} h_i + O(1). \tag{8}$$

Now, replacing (7) by the continuous version

$$F(x) = m \log\left(\frac{k+1}{k^2}f(x) + \frac{k^2-1}{k^2}\right) \tag{9}$$

where

$$F(x)=\int_{(k+1)m/k}^{x} f(y)\,dy \quad \text{and} \quad f(m(k+1)/k)=\frac{1}{k+1},$$

it is routine to verify that

$$F(j)=\sum_{i=m(k+1)/k}^{j-1} h(i)+O(1)\leqq \sum_{i=1+m(k+1)/k}^{j-1} h_i+O(1). \tag{10}$$

The solution to the differential equation (9) for $F(x)$ is readily found to be

$$F(x)=m\cdot\log\left\{\frac{k^2}{k^2-1}-\frac{1}{k^2-1}e^{(k-1)(x/m-(k+1)/k)}\right\}^{-1}. \tag{11}$$

Hence, from (10) and (11), we have

$$\sum_{i=1+m(k+1)/k}^{j-1} h_i\geqq m\cdot\log\left\{\frac{k^2}{k^2-1}-\frac{1}{k^2-1}e^{(k-1)(j/m-(k+1)/k)}\right\}^{-1}+O(1). \tag{12}$$

To complete the proof, we observe that the right-hand side of (6) is an increasing function of h_j, and thus we can use the bound $h_j\leqq 1/k$ to obtain from (6) that

$$\sum_{i=1+m(k+1)/k}^{j-1} h_i\leqq m\cdot\log\frac{k^3+1}{k^3}+O(1).$$

Combining this expression with (12) evaluated at $j=\mathrm{FF}(L)$, we have

$$m\cdot\log\frac{k^3+1}{k^3}\geqq m\cdot\log\left\{\frac{k^2}{k^2-1}-\frac{1}{k^2-1}e^{(k-1)(\mathrm{FF}(L)/m-(k+1)/k)}\right\}^{-1}+O(1),$$

from which

$$\mathrm{FF}(L)\leqq m\left(\frac{k+1}{k}+\frac{1}{k-1}\log\frac{k^2}{k^2-k+1}\right)+O(1)$$

follows by straightforward algebra. □

In § 4 we will construct examples that come rather close to the upper bounds given by Theorem 1. For example, we will show that

$$1.75\leqq W_R(\mathrm{FF},2)\leqq\tfrac{3}{2}+\log\tfrac{4}{3}\approx 1.788.$$

We now examine the unrestricted ($k=1$) case. It is easy to prove that $W_R(\mathrm{FF})\leqq 3$. Specifically, the following two observations suffice. First, any item packed in a bin B_j with $j>2\cdot\mathrm{OPT}_R(L)$ must have size larger than $\frac{1}{2}$, since a piece of size $s<\frac{1}{2}$ could not be packed in such a bin unless each of the first $2\cdot\mathrm{OPT}_R(L)$ bins were filled to a level of at least $1-s>\frac{1}{2}$, which would imply that the cumulative size of all items in the packing was greater than $\mathrm{OPT}_R(L)$. The bound of 3 then follows from the observation that there can be at most $\mathrm{OPT}_R(L)$ pieces larger than $\frac{1}{2}$, since more than that number could not possibly fit in $\mathrm{OPT}_R(L)$ bins.

One might expect that a similar argument, using the fact that there can be at most $\mathrm{OPT}_R(L)$ pieces larger than $\frac{1}{2}$, would allow us to conclude that $W_R(\mathrm{FF})$ is bounded above by $1+W_R(\mathrm{FF},2)$. Indeed, such an argument is easily seen to work for the modified version of FF that segregates pieces larger than $\frac{1}{2}$ from the other pieces, always placing such an item in a bin by itself and never placing other items with it. In this case, we know from Theorem 1 that the items that do not exceed $\frac{1}{2}$

will never occupy more than W_R (FF, 2) · OPT_R $(L)+O(1)$ bins and that the items larger than $\frac{1}{2}$ will never occupy more than OPT_R (L) bins, so we have immediately that

$$W_R \text{ (FFM)} \leqq \tfrac{5}{2}+\log \tfrac{4}{3} \approx 2.788,$$

where FFM denotes this modified version of FF.

Unfortunately, the situation with FF itself is not so simple. The difficulty arises from the possibility of constructing lists that end up with no active items larger than $\frac{1}{2}$, but that use items larger than $\frac{1}{2}$ temporarily in order to force an FF packing with more than W_R (FF, 2) · OPT_R (L) bins. In particular, it is not difficult to construct such lists that require a number of bins arbitrarily close to 2 · OPT_R (L). Since the type of upper bound argument used above would still allow for an additional OPT_R (L) pieces larger than $\frac{1}{2}$ being placed beyond this point, we would have no improvement whatsoever over the trivial bound of 3 for W_R (FF). Using considerably more subtle arguments, we shall reduce this somewhat to approximately 2.897.

THEOREM 2. *Define the constant* $\psi = \frac{3}{2}\log((\sqrt{13}-1)/2) \approx .397$. *We have*

$$W_R \text{ (FF)} \leqq \tfrac{5}{2}+\psi \approx 2.897.$$

Proof. Let W denote the set of bins with indices larger than $3m/2$, where $m = \mathrm{OPT}\,(L)$. We begin with the following preliminary result.

CLAIM 2.1. *Suppose a total of l items has been packed by* FF *in the bins of W. Let $A(l)$ denote their cumulative size, and let g_l denote the size of the l*th *such item. Then there exists a constant c such that*

$$A(l) \geqq \frac{m}{2}[e^{2(l-c)/3m}-1] \tag{13}$$

and

$$g_l \geqq \tfrac{1}{3}e^{2(l-c)/3m}. \tag{14}$$

Proof of Claim 2.1. When the lth item, say p_i, is packed in W, the first $\lfloor 3m/2 \rfloor$ bins must each be filled to a level exceeding $1-s_i$. Thus we must have

$$(1-g_l)\left\lfloor \frac{3m}{2} \right\rfloor + \sum_{i=1}^{l-1} g_i \leqq m. \tag{15}$$

Analogous to the continuous approximation made in the proof of Theorem 1, let $g(x)$ and $G(x)$ be defined by

$$(1-g(x))\frac{3m}{2}+G(x)=m \tag{16}$$

and

$$G(x)=\int_0^x g(x)\,dx$$

with $g(0)=\frac{1}{3}$ (any item placed beyond bin $3m/2$ must be larger than $\frac{1}{3}$). Then it is easily verified that, for any integer $l \geqq 1$,

$$G(l) \leqq A(l)+O(1) \tag{17}$$

and

$$g(l) \leqq g_l + O\left(\frac{1}{m}\right). \tag{18}$$

The solution to the differential equation (16) for G is readily found to be

$$G(x) = \left\lceil \frac{m}{2} e^{2x/3m} - 1 \right\rceil. \tag{19}$$

Thus from (17) we have

$$A(l) \geqq \frac{m}{2}[e^{2l/3m} - 1] + O(1) \tag{20}$$

and, by differentiating (19) and using (18),

$$g_l \geqq \tfrac{1}{3} e^{2l/3m} + O\left(\frac{1}{m}\right). \tag{21}$$

The claim follows from (20) and (21) by observing that

$$e^{2l/3m} - e^{2(l-c)/3m} = O\left(\frac{1}{m}\right)$$

for any constant c. □

We now show by contradiction that for all L,

$$\mathrm{FF}(L) \leqq (\tfrac{5}{2} + \psi)m + 2c$$

where $m = \mathrm{OPT}_R(L)$ and c is the constant in Claim 2.1. Thus suppose that this bound is violated during the FF packing of some list L. As in the proof of Theorem 1, we may assume that no occupied bin ever becomes empty during the FF packing of L, and we may assume that the bound is violated for the first (and only) time when the last item p_n is packed. To simplify notation, we shall also assume that m is even; although it should be clear that this does not affect the result in any significant way, the arguments can be generalized essentially by replacing $3m/2$ and $5m/2$ by $\lfloor 3m/2 \rfloor$ and $\lfloor 5m/2 \rfloor$ throughout.

The argument will concentrate on the structure of the packing beyond bin $3m/2$. Let $(3m/2)+q$ be the index of the rightmost (highest-indexed) bin to contain an item of size $\frac{1}{2}$ or smaller during $(0, a_n]$. Clearly $q > \psi m$, for otherwise, since there can be at most m items larger than $\frac{1}{2}$, we would have

$$\mathrm{FF}(L) \leqq \frac{3m}{2} + q + m \leqq \left(\frac{5}{2} + \psi\right)m.$$

Let p_j, with $s_j \leqq \frac{1}{2}$, be any such item packed in bin $(3m/2)+q$, and let bin $(3m/2)+r$, $r \geqq q$, denote the rightmost nonempty bin at time a_j, when p_j is packed. Let P be the set of items packed in bins to the right of bin $(3m/2)+r$ during $(0, a_n]$. Note that each item in P is larger than $\frac{1}{2}$, does not depart during $(0, a_n]$ (since that would create an empty bin), and is the rightmost item at the time it is packed.

We call an item p_k that is placed in a bin of W during $[a_j, a_n]$ an *α-item* if there exists some $p_i \in P$ such that $s_i + s_k \leqq 1$ (i.e., p_i and p_k could fit in a single bin). Notice that the size of each α-item is strictly between $\frac{1}{3}$ and $\frac{1}{2}$, because each is packed to the right of bin $3m/2$ and because each fits with some item from P. We are interested in the α-items because they are among the items that might be packed with an item from P in an optimal packing. We now show how the existence of α-items leads to the desired contradiction; following that we shall deal with the case in which no α-items exist.

Let p_u be the *rightmost* item in P that is small enough to fit with some α-item, and consider any item $p_i \in P$ that FF packs to the right of p_u. Since p_i is too large to fit with any α-item, by the choice of p_u, it follows that $s_i > s_u$. Now consider the set Z_u of bins from W that contain only a single item at time a_u, when p_u is packed. The items in bins of Z_u at a_u are all larger than $1-s_u$, or p_u would have been packed in such a bin, and hence they are all larger than $1-s_i$. Furthermore, any items that are placed in bins of Z_u after time a_u either are larger than $1-s_u > 1-s_i$ or are α-items and hence must be larger than $1-s_i$, since p_i is too large to fit with any α-items. Thus we conclude that no item in a bin of Z_u at time a_n can fit in the same bin with any item packed to the right of p_u by FF.

Let P' denote the set of items packed to the right of p_u by FF. Each item in P' is larger than $\frac{1}{2}$ and thus must go in a separate bin in the optimal packing. No item in Z_u at time a_n can fit in any of those bins and hence, since each is larger than $\frac{1}{3}$, they must occupy at least $z_u/2$ additional bins in the optimal packing, where $z_u = |Z_u|$. We then have

$$|P'| + \frac{z_u}{2} \leqq m \quad \text{or} \quad |P'| \leqq m - \frac{z_u}{2}.$$

Therefore, if p_u is packed by FF in bin $(3m/2)+r'$,

$$\text{FF}(L) = \frac{3m}{2} + r' + |P'| \leqq \frac{5m}{2} + \frac{2r' - z_u}{2}.$$

But $2r' - z_u$ is simply the number w of items in W at time a_u, so we can write

$$\text{FF}(L) \leqq \frac{5m}{2} + \frac{w}{2}. \tag{22}$$

We now find an upper bound for w.

From Claim 2.1, the cumulative size of items in the packing at time a_u is at least

$$(1-s_u)\frac{3m}{2} + A(w) \geqq (1-s_u)\frac{3m}{2} + \frac{m}{2}[e^{2(w-c)/3m} - 1].$$

We know that p_u is small enough to fit with some α-item. Since there are at least $r \geqq q$ occupied bins in W before any α-item is packed, we have g_q as a lower bound to the size of all α-items. Hence $1 - s_u \geqq g_q$. Using $q > \psi m$ and the bound in (14) to substitute into the above bound on cumulative item size at time a_u, we obtain the bound

$$\frac{m}{2} e^{2(\psi m - c)/3m} - \frac{m}{2}[e^{2(w-c)/3m} - 1].$$

Since this quantity can be no larger than m, we have

$$\frac{m}{2}[e^{2(\psi m - c)/3m} + e^{2(w-c)/3m}] - \frac{m}{2} \leqq m.$$

By routine algebra, this inequality leads to

$$w \leqq \frac{3m}{2} \log\left[3 - \frac{\sqrt{13}-1}{2} e^{-2c/3m}\right] + c,$$

which can be rewritten as

$$w \leqq \frac{3m}{2} \log\left[\left(\frac{\sqrt{13}-1}{2}\right)^2 + \frac{\sqrt{13}-1}{2}(1 - e^{-2c/3m})\right] + c.$$

Using $1-e^{-y} \leqq y$ and $\log(x+y) \leqq \log x + y$, $x \geqq 1$, $y \geqq 0$, this simplifies to

$$w \leqq 3m \log\left(\frac{\sqrt{13}-1}{2}\right) + \left(\frac{\sqrt{13}-1}{2}\right)c + c \leqq 2\psi m + 3c.$$

Substituting this bound for w into (22), we obtain the desired contradiction.

It remains for us to handle the case when no α-items exist. In this case, we consider the packing at time a_j, and let Z_j be the set of one-item bins in W at that time. Since p_j is a rightmost item of size $\frac{1}{2}$ or less, all items in bins of Z_j to the right of p_j at time a_j are larger than $\frac{1}{2}$. Furthermore, all items in bins of Z_j to the left of p_j at time a_j must also be larger than $\frac{1}{2}$, or p_j would have been placed in one of those bins. Therefore, no item in P can fit with any item in Z_j at time a_j. Since there are no α-items, no item placed in a bin of Z_j after time a_j can fit with any item in P either, and thus no item in a bin of Z_j at time a_n can fit with any item in P. Thus we have, as before,

$$\mathrm{FF}(L) \leqq \frac{3m}{2} + r + \left(m - \frac{z_j}{2}\right) = \frac{5m}{2} + \frac{w}{2}$$

where w is the number of items in W at time a_j. But, since $s_j \leqq \frac{1}{2}$, the cumulative size of items in the packing at time a_j is at least

$$(1-s_j)\frac{3m}{2} + A(w) \geqq \frac{3m}{2} + \frac{m}{2}[e^{2(w-c)/3m} - 1].$$

Since this cannot exceed m, we then obtain

$$w \leqq \frac{3m}{2} \log \frac{3}{2} + c,$$

and hence, substituting into (22), we have

$$\mathrm{FF}(L) \leqq \left(\frac{5}{2} + \frac{3}{4}\log\frac{3}{2}\right)m + \frac{c}{2} < \left(\frac{5}{2} + \psi\right)m + 2c,$$

as desired. □

4. Lower bounds for W_R (FF). In this section we again begin by considering separately the case of $W_R(\mathrm{FF}, k)$, $k \geqq 2$.

THEOREM 3. *For any $k \geqq 2$,*

$$W_R(\mathrm{FF}, k) \geqq \frac{k+1}{k} + \frac{1}{k^2}.$$

Proof. The proof consists of showing that for arbitrarily large integers m we can construct lists L with $\mathrm{OPT}_R(L) \leqq m$ such that

$$\mathrm{FF}(L) = \frac{k^2+k+1}{k^2}m - 2.$$

In the following we shall construct the list L and describe its FF processing at the same time. The sum of the sizes of the items in the FF packing after each step will be close to the maximum of m, and hence we shall specify the steps in terms of "removing" certain items, "forming" new items from the amount removed, and then "packing" the newly formed items. The corresponding arrival and departure times needed for a formal description of L in terms of triples (a_i, d_i, s_i) can be inferred directly from this description.

The construction is broken down into 3 successive stages, corresponding to the initial packing process for bins 1 to m, bins $m+1$ to $((k+1)m/k)-2$, and bins $((k+1)m/k)-1$ to $((k^2+k+1)m/k^2)-2$. So that all these quantities will be integers, we shall assume that m is a multiple of k^2.

Stage 1. The construction begins simply by packing the first m bins completely full with items of size ε, where ε is a suitably chosen small positive number. Constraints on the size of ε will be indicated in Stages 2 and 3 below.

Stage 2. In this stage we pack bins $m+1$ through $((k+1)m/k)-2$. Just before we pack bin $m+i$, $1 \leqq i \leqq (m/k)-2$, the current packing will be as follows:

(a) all bins have level exactly $m/(m+i-1)$;

(b) each of bins 1 through m contains only ε-items;

(c) for $1 \leqq j \leqq i$, bin $m+j$ contains t items of size $(j/(m+j))+\Delta$ and $k-1$ items of size

$$\frac{1}{k-1}\left[\frac{m}{m+i-1}-t\left(\frac{j}{m+j}+\Delta\right)\right]$$

where t is the least nonnegative integer such that this last item size is no larger than $1/k$, and Δ is a suitably small positive constant.

The first step in packing bin $m+i$ is to remove an amount equal to $m/(m+i)\times(m+i-1)$ from each of the preceding $m+i-1$ bins. For bins 1 through m, we do this simply by taking out a sufficient number of ε-size items (the desired amount can be obtained exactly if ε was chosen appropriately). For bin $m+j$, $1 \leqq j < i$, we remove the $k-1$ "large" items from that bin *and* as many of the "small" items as possible, under the constraint that the total removed minus $m/(m+i)(m+i-1)$ be no larger than $(k-1)/k$. Then all but $m/(m+i)(m+i-1)$ of this is formed into $k-1$ identical new large items, and these are repacked. This "shaving" process, consisting of the removal of some items in a bin, followed by the repacking of slightly smaller items into that same bin, is performed one bin at a time for each of the bins $m+j$, $1 \leqq j < i$. We now prove, in a slightly more general form than is needed now, that the new large items formed in this shaving process will indeed be repacked into the same bin by FF.

CLAIM 3.1. *If the current levels of all bins to the left of bin $m+j$ are at least as large as the new level for bin $m+j$ and the new level for bin $m+j$ is at least*

$$1-\frac{m}{k(m+j)}+\Delta,$$

then the new large items formed from bin $m+j$ in the "shaving" process will be packed back into bin $m+j$ by First Fit.

Proof of Claim 3.1. Let $\alpha=(j/(m+j))+\Delta$ denote the size of the small items in bin $m+j$, and let β denote the size of the newly formed large items. Note that the rule used for determining β ensures that

$$\beta+\frac{\alpha}{k-1}>\frac{1}{k}, \tag{23}$$

for otherwise an additional small item would have been removed. The existence of at least one additional small item is ensured by the lower bound on the level of bin $m+j$, since

$$\frac{1}{k-1}\left(1-\frac{m}{k(m+j)}+\Delta\right)>\frac{1}{k}.$$

We now show that $\beta + l > 1$, where l is the least current level among the bins to the left of bin $m+j$, so each of the large items will fail to fit in the preceding bins and hence will be placed in bin $m+j$ by FF, as required. The proof is divided into two cases:

Case 1. Suppose bin $m+j$ has more than one small item remaining. Then, since l is at least as large as the new level for bin $m+j$, we have

$$l \geqq (k-1)\beta + 2\alpha$$

so, using (23),

$$\beta + l \geqq k\beta + 2\alpha \geqq k\left(\beta + \frac{\alpha}{k-1}\right) > k\left(\frac{1}{k}\right) = 1.$$

Case 2. Suppose bin $m+j$ has exactly one small item remaining. Then we have

$$(k-1)\beta + \alpha \geqq 1 - \frac{m}{k(m+j)} + \Delta$$

which implies that

$$\beta \geqq \frac{1}{k-1}\left(1 - \frac{m}{k(m+j)} - \frac{j}{m+j}\right) \geqq \frac{m}{k(m+j)}.$$

Thus

$$\beta + l \geqq \frac{m}{k(m+j)} + \left(1 - \frac{m}{k(m+j)} + \Delta\right) = 1 + \Delta > 1.$$

The claim follows. □

In the present setting, Claim 3.1 implies that the large items will always be repacked as required, since the common bin level always exceeds $k/(k+1)$ during Stage 2, and for the claim to apply it need only exceed

$$1 - \frac{m}{k(m+j)} + \Delta \leqq 1 - \frac{m}{k\left(m + \frac{m}{k} - 2\right)} + \Delta \leqq \frac{k}{k+1}\left(\frac{m-2}{m - \frac{2k}{k+1}}\right) + \Delta,$$

which is less than $k/(k+1)$ for Δ sufficiently small.

The second step in packing bin $m+i$ is to form new items from the total of $m/(m+i)$ that has been shaved off the preceding bins, so that these new items will be packed into bin $m+i$ by FF and will have the required properties. All we do in this step is form items of size $(i/(m+i)) + \Delta$ until the amount remaining does not exceed $(k-1)/k$, and that remaining amount is then formed into $k-1$ equal-sized large items. Given Claim 2 to follow and the specified levels of the preceding bins, it should be apparent that this can be done and that the items will all be placed in bin $m+i$ by FF. Furthermore, the packing has the overall form required for us to go on to bin $m+i+1$.

CLAIM 3.2. *If the integer t is chosen as small as possible so that*

$$\frac{k-1}{k} \geqq \frac{m}{m+i} - t\left(\frac{i}{m+i} + \Delta\right) = (k-1)\beta,$$

then

$$\frac{1}{k} \geqq \beta \geqq \frac{i}{m+i} + \Delta$$

(*i.e., the "large" item formed above is indeed a legal large item*).

Proof of Claim 3.2. The upper bound on β is necessarily satisfied by the way in which β is defined, so we need only verify that β is not smaller than $(i/(m+i))+\Delta$. Note first that t must be at least 1, since $i<m/k$ implies

$$\frac{m}{m+i}>\frac{k}{k+1}>\frac{k-1}{k}.$$

We now consider two cases.

Case 1. If $1\leqq i<(k-1)m/(k^2-k+1)$, we have

$$\frac{i}{m+i}+\Delta<\frac{k-1}{k^2}$$

(for Δ suitably small). Thus $\beta<(i/(m+i))+\Delta$ would imply

$$(k-1)\beta+\frac{i}{m+i}+\Delta<k\left(\frac{i}{m+i}+\Delta\right)<\frac{k-1}{k},$$

contradicting the minimality of t.

Case 2. If $(k-1)m/(k^2-k+1)\leqq i<m/k$, then

$$\frac{k-1}{k^2}<\frac{i}{m+i}+\Delta<\frac{1}{k+1}$$

and

$$\frac{k}{k+1}<\frac{m}{m+i}\leqq\frac{k^2-k+1}{k^2}.$$

Thus

$$\frac{m}{m+i}-\left(\frac{i}{m+i}+\Delta\right)<\frac{k^2-k}{k^2}+\frac{2-k}{k^2}\leqq\frac{k-1}{k},$$

so t must be equal to 1. Therefore

$$\beta=\frac{1}{k-1}\left(\frac{m}{m+i}-\left(\frac{i}{m+i}+\Delta\right)\right)>\frac{1}{k+1}>\frac{i}{m+i}+\Delta$$

as required. □

In order to prepare for Stage 3 after packing bin $((k+1)m/k)-2$, we carry out the "shaving" step for the next bin but do not pack any items into that bin. Thus, at the end of Stage 2, we have items in the first $((k+1)m/k)-2$ bins, and these bins are all at a level slightly above $k/(k+1)$. Specifically, the level of each of these bins is exactly

$$\frac{km}{(k+1)m-k}$$

and hence the sum of all the items in the bins is

$$m-\frac{mk}{(k+1)m-k}.$$

Stage 3. In this stage, all bins beyond bin $((k+1)m/k)-2$ will be packed with k items larger than $1/(k+1)$, with all items going into the same bin having the same

size and being essentially as small as possible. However, due to the "shaving" constraint given in Claim 3.1, it will not be possible to maintain the levels of bins 1 through $((k+1)m/k)-2$ at the same value, as was done during Stage 2. In order to shave some amount off the level of bin $m+j$, it is necessary that the level of that bin (and all preceding bins) be at least

$$1-\frac{m}{k(m+j)}+\Delta. \tag{24}$$

Thus we will never reduce the level of bin $m+j$ below this value. This constraint has no effect on the first m bins, since the desired amount can always be removed from these bins simply by removing a suitable number of ε-sized items (i.e., there is no repacking involved in the shaving process).

For sequences h_i, δ_i, and λ_i, $i=((k+1)m/k)-1, \cdots, ((k^2+k+1)m/k^2)-2$, to be specified below, pass $i-((k+1)m/k)+2$ of Stage 3 consists simply of "shaving" δ_i from the levels of each of bins 1 through λ_i, forming k items of size h_i from the amount removed, and then packing those k items into bin i. The shaving process is carried out in exactly the same manner as described for Stage 2, and we shall verify below that the lower bound (24) on bin levels is satisfied, so that the shaving can indeed be done in this manner.

Let $l(i)$, $i \geqq ((k+1)m/k)-1$, denote the least bin level just following pass $i-((k+1)m/k)+2$. From the above description, we will have

$$l(i)=\frac{km}{(k+1)m-k}-\sum_{j=((k+1)m/k)-1}^{i}\delta_j. \tag{25}$$

We now define the sequences h_i, δ_i, and λ_i as follows:

$$h_i=\frac{m}{(k+1)^2m-k^2(i+2)},$$

$$\delta_i=h_i-h_{i-1}=\frac{mk^2}{[(k+1)^2m-k^2(i+2)]\cdot[(k+1)^2m-k^2(i+1)]}$$

(where we take $h_{i-1}=1/(k+1)$ for $i=((k+1)m/k)-1$), and λ_i is the largest index, $\lambda_i \leqq ((k+1)m/k)-2$, such that $l(i)$ is not smaller than the minimum level that we are allowing in bin λ_i, which by (24) is

$$1-\frac{m}{k\lambda_i}+\Delta.$$

This choice of λ_i ensures that the shaving process can be carried out as required. From (24), (25) and the definition of δ_i as h_i-h_{i-1}, we have λ_i as the largest index satisfying

$$\begin{aligned} 1-\frac{m}{k\lambda_i}+\Delta \leqq l(i) &= \frac{km}{(k+1)m-k}-\sum_{j=((k+1)m/k)-1}^{i}\delta_j \\ &= \frac{km}{(k+1)m-k}-\left(h_i-\frac{1}{k+1}\right) \\ &= 1+\frac{k^2}{(k+1)^2m-k(k+1)}-h_i. \end{aligned} \tag{26}$$

Hence

$$\lambda_i = \left\lfloor \frac{m}{k\left(h_i + \Delta - \dfrac{k^2}{(k+1)^2 m - k(k+1)}\right)} \right\rfloor \tag{27}$$

$$\geqq \frac{m}{k\left(h_i - \dfrac{k^2}{(k+1)^2 m - k(k+1)}\right)} - 1$$

for Δ suitably small.

To see that $h_i \leqq 1/k$, we use the fact that $i \leqq ((k^2+k+1)m/k^2) - 2$ to obtain

$$h_i \leqq \frac{m}{(k+1)^2 m - k^2((k^2+k+1)m/k^2)}$$

$$= \frac{m}{(k+1)^2 m - (k^2+k+1)m} = \frac{m}{km} = \frac{1}{k}.$$

To show that the k items of size h_i created during pass $i - ((k+1)m/k) + 2$ of Stage 3 will be placed into bin i by FF, we need to show that $h_i + l(i) > 1$ and that $h_i > 1/(k+1)$. The first of these follows directly from (26), which yields

$$h_i + l(i) = 1 + \frac{k^2}{(k+1)^2 m - k(k+1)} > 1.$$

The second follows from the definition of h_i and the fact that $i > ((k+1)m/k) - 2$, since we have

$$h_i > \frac{m}{(k+1)^2 m - k^2((k+1)m/k)} = \frac{m}{(k+1)m} = \frac{1}{k+1}.$$

Finally, we must prove that the sum of the sizes of all the items in the packing just following each pass of Stage 3 remains no more than m. Since we remove a total of $\lambda_i \delta_i$ and add kh_i during pass $i - ((k+1)m/k) + 2$, it is sufficient to show that for each i

$$m - \frac{mk}{(k+1)m - k} + \sum_{j=((k+1)m/k)-1}^{i} (kh_j - \lambda_j \delta_j) \leqq m,$$

or:

CLAIM 3.3.

$$\sum_{j=((k+1)m/k)-1}^{i} (kh_j - \lambda_j \delta_j) \leqq \frac{mk}{(k+1)m - k}, \qquad \frac{(k+1)m}{k} - 1 \leqq i \leqq \frac{(k^2+k+1)m}{k^2} - 2.$$

Proof of Claim 3.3. We begin by considering individual terms in the summation. By (27), we have

$$k \cdot h_j - \lambda_j \delta_j \leqq k \cdot h_j - \delta_j \left(\frac{m}{k\left(h_j - \dfrac{k^2}{(k+1)^2 m - k(k+1)}\right)} - 1 \right).$$

Using $\delta_j = k^2 h_j h_{j-1}/m$, we obtain

$$k \cdot h_j - \lambda_j \delta_j \leqq k h_j - \frac{k^2 h_j h_{j-1}}{k\left(h_j - \dfrac{k^2}{(k+1)^2 m - k(k+1)}\right)} + \delta_j$$

$$\leqq k h_j - k h_{j-1} + \delta_j = (k+1)\delta_j.$$

Thus

$$\sum_{j=((k+1)m/k)-1}^{i} (k h_j - \lambda_j \delta_j) \leqq (k+1) \sum_{j=((k+1)m/k)-1}^{i} \delta_j$$

$$= (k+1)\left(h_i - \frac{1}{k+1}\right) \leqq (k+1)\left(\frac{1}{k} - \frac{1}{k+1}\right) = \frac{1}{k} < \frac{mk}{(k+1)m-k},$$

as required. □

Thus the First Fit packing proceeds as described during Stages 1, 2 and 3, and it requires $((k^2+k+1)m/k^2)-2$ bins. Furthermore, the sum of the sizes of the items in the packing at any point is no more than m. It remains for us to demonstrate that the optimal packing at each point would not encounter "fitting" problems that would force it to use more than m bins. However, because of the large number of ε-size items during all three stages, it is easy to see that such fitting problems do not arise. A packing into m bins can always be obtained from the FF packing by moving the ε-size items into bins $m+1$ through $2m$, filling each bin exactly to level 1 (or slightly less, if the sum of the current item sizes is less than m), which can be done if ε is chosen suitably small. Thus the optimum packing never needs more than m bins, and the stated bound on W_R (FF, k) follows. This completes the proof of Theorem 3. □

We are now ready to go on to the case of $k = 1$. We shall first show how the $7m/4$ construction just described for $k = 2$ can be extended to an $11m/4$ construction for $k = 1$, by appropriately adding on about m items larger than $\frac{1}{2}$. Following this, we sketch a different (and more complicated) construction that can be used to strengthen the lower bound.

THEOREM 4.

$$W_R \text{ (FF)} \geqq \tfrac{11}{4}.$$

Proof. The construction will be described in the same format as used in the proof of Theorem 3 and will be composed of three stages.

Stage 1. For $m+1 \leqq i \leqq (7m/4)-2$, let α_i denote the size of the smallest item in bin i at the end of the $(7m/4)-2$ construction. Note that

$$\alpha_i = \frac{i-m}{i} + \Delta, \qquad m+1 \leqq i \leqq \frac{3m}{2} - 2;$$

$$\alpha_i = h_i = \frac{m}{9m - 4(i+2)}, \qquad \frac{3m}{2} - 1 \leqq i \leqq \frac{7m}{4} - 2.$$

Our first stage is identical with the $(7m/4)-2$ construction except that instead of packing two items of size $\frac{1}{2}$ in bin $(7m/4)-2$, we only pack a single item of size

$$1 - \alpha_{(7m/4)-3}$$

into that bin. This item will not fit in any of the preceding bins since their levels all exceed $\frac{1}{2}$ and this item is itself larger than $\frac{1}{2}$.

Stage 2. For $i=(7m/4)-1,\cdots,(5m/2)-6$, we pack bin i with a single item larger than $\frac{1}{2}$ by executing the following steps, where $u(i)$ is defined by

$$u(i)=\frac{7m}{2}-4-i.$$

Step 1. Remove all but a single item of size $\alpha_{u(i)}$ from bin $u(i)$ and remove items of size ε totaling

$$\alpha_{u(i)}-\alpha_{u(i)-1}$$

from each of bins 1 through m.

Step 2. Form a new item of size $1-\alpha_{u(i)-1}$ from the amount removed and pack it according to FF into bin i.

To verify that the new item will actually be packed into bin i by FF, we observe that at the conclusion of Stage 1 all bins are filled to a level greater than $\alpha_{(7m/4)-3}$. Therefore, after applying Step 1 prior to packing bin i, bins 1 through m will all be filled to a level greater than

$$\alpha_{(7m/4)-3}-\sum_{j=(7m/4)-1}^{i}[\alpha_{u(j)}-\alpha_{u(j)-1}]=\alpha_{u(i)-1}.$$

Since bins $m+1$ through $u(i)-1$, and bins $(7m/4)-2$ through $i-1$, are all at levels greater than $\frac{1}{2}$, and since bins $u(i)$ through $(7m/4)-3$ are all at levels of at least $\alpha_{u(i)}$, it follows that the new item of size $1-\alpha_{u(i)-1}$ will not fit in any of these bins and thus will be packed in bin i by FF.

To verify that the amount removed in Step 1 is always larger than the size of the new item formed in Step 2, we observe that the amount removed from bin $u(i)$ is always at least $\alpha_{u(i)}$, and hence it suffices to show that

$$m(\alpha_{u(i)}-\alpha_{u(i)-1})+\alpha_{u(i)}>1-\alpha_{u(i)-1}.$$

We do this by proving the following:

CLAIM 4.1. *For* $m+1<j\leqq(7m/4)-3$,

$$m(\alpha_j-\alpha_{j-1})+\alpha_j+\alpha_{j-1}>1.$$

Proof of Claim 4.1. We consider three cases.

Case 1. $3m/2\leqq j\leqq(7m/4)-3$. In this case, we have

$$m(\alpha_j-\alpha_{j-1})=m\left(\frac{m}{9m-4(j+2)}-\frac{m}{9m-4(j+1)}\right)$$

$$=\frac{4m^2}{[9m-4(j+2)][9m-4(j+1)]}>\frac{4m^2}{[3m]\cdot[3m]}=\frac{4}{9}.$$

Since α_j and α_{j-1} both exceed $\frac{1}{3}$, the result follows.

Case 2. $j=(3m/2)-1$. In this case, we have

$$m(\alpha_j-\alpha_{j-1})=m\left(\frac{m}{3m-4}-\frac{m-4}{3m-4}\right)-m\Delta=\frac{4m}{3m-4}-m\Delta$$

which, by itself, exceeds 1 for Δ sufficiently small.

Case 3. $m+1<j\leqq(3m/2)-2$. In this case, we have

$$m(\alpha_j-\alpha_{j-1})=m\left(\frac{j-m}{j}-\frac{j-1-m}{j-1}\right)=m\left(\frac{m}{j(j-1)}\right).$$

Thus

$$m(\alpha_j-\alpha_{j-1})+\alpha_j+\alpha_{j-1}=\frac{m^2+2j(j-1)-m(2j-1)}{j(j-1)}+2\Delta$$
$$>1+\frac{(m-j)(m-j+1)}{j(j-1)}>1,$$

and the claim is proved. □

It remains for us to show that, at any point during Stage 2, there exists an optimum packing into m or fewer bins. We already know that the sum of the sizes of the items in the FF packing is never more than m, but we still have to show that "fitting" problems do not force an optimum packing to use more than m bins. To show this it suffices to show that the items to the right of bin m in the FF packing can always be packed into m bins, since any gaps in partially filled bins in such a packing can be filled exactly using items of size ε from bins 1 through m (for a suitably chosen ε). Indeed, it follows immediately from our construction that the items to the right of bin m can always be packed into $(3m/4)-2$ bins. Such a packing can be formed from the FF packing just following the placement of the item larger than $\frac{1}{2}$ in bin i, $i\geqq(7m/4)-1$, simply by combining the two items in bin j and bin $u(j)-1$ in a single bin for each j, $(7m/4)-2\leqq j<i$. By the construction, each such pair of items sums exactly to 1 and hence fits in a single bin. Furthermore, this reduces the number of occupied bins to the right of bin m from $i-m$ to

$$(i-m)-\left(i-\left[\frac{7m}{4}-2\right]\right)=\frac{3m}{4}-2,$$

as required.

Stage 3. In this stage, we first remove all but a single item of size ε from each of bins 1 through m, and we remove all but a single smallest item from bin $m+1$. Note that, since the remaining items of size ε would all fit in a single bin with the item remaining in bin $m+1$, for ε suitably small, there exists an optimum packing of the current items into $(3m/4)-2$ bins. We now form $(m/4)+2$ items of size 1 and pack them according to FF. Since these items will not fit into any of the first $(5m/2)-6$ bins, they will be placed into bins $(5m/2)-5$ through $(11m/4)-4$, and hence the First Fit packing requires $(11m/4)-4$ bins. Furthermore, since the items in the packing prior to the formation of these $(m/4)+2$ new items could be packed into $(3m/4)-2$ bins, the number of bins in the optimal packing is increased to at most m by the addition of these items, and therefore the optimum packing never exceeds m bins during Stages 1, 2, and 3. Theorem 4 follows immediately. □

Note that the construction just given to prove that W_R (FF) $\geqq\frac{11}{4}$ also suffices to prove the same lower bound for W_R (FFM), where FFM is the previously described variant of First Fit that segregates items larger than $\frac{1}{2}$ from the other items. Thus we have

$$2.75\leqq W_R\text{ (FFM)}\leqq\tfrac{5}{2}+\log\tfrac{4}{3}\approx 2.788.$$

Although the construction we shall sketch in a moment seems to suggest that the worst-case bounds for FF are worse than those for FFM, it does not appear that this should be taken to recommend FFM over FF in practice, since one would expect FF to be generally less wasteful of space on the average.

We now sketch an alternative construction, based on a more complicated packing strategy, that uses the possibility of mixing items larger than $\frac{1}{2}$ with items smaller than $\frac{1}{2}$ in a single bin to tighten the lower bound for FF. This informal presentation is

intended to illustrate some of the complications that can arise from strategies that use such mixing, which in turn contribute to the apparent difficulty of proving a significantly stronger upper bound for FF than that given in Theorem 2. The construction we shall give to illustrate this yields a 2.766 bound which can be improved to 2.77 by optimizing the key parameters.

Let m be some large multiple of 64. The example is constructed in the following stages:

Stage 1. The construction begins by following the procedure described in the proof of Theorem 3 (for $k=2$), but halting at the point of a $105m/64$ example.

Stage 2. We now want to extend the packing by $m/8$ bins, i.e., to within a constant of $113m/64=(7m/4)+(m/64)$, with only items smaller than $\frac{1}{2}$ occurring in the final packing. To do this, we first use the shaving procedures described earlier to reduce the level of each bin i, $1\leqq i<(3m/2)-2$, to $\max\{1-\gamma+m\varepsilon', 1-(m/2i)\}$, where ε' is chosen suitably small and γ is chosen as indicated below. Next, we pack $m/8$ items of size $1-\gamma+i\varepsilon'$, $1\leqq i<m/8$, in *decreasing* sequence into bins $(105m/64)+1$ through $113m/64$. Using a standard argument on the total of all the item sizes, the smallest value of γ can be calculated such that this total does not exceed m when the last of the $m/8$ items is packed. (The calculation, which we omit, yields a value for γ that is slightly less than $\frac{1}{2}$.)

Our next step is to convert the $(1-\gamma)$-items to γ-items. To do this we first pack an item of size $\gamma-\varepsilon'$. Of course, we must assume that this can be done without causing the total to exceed m. (Although this could be assured by packing only $(m/8)-1$ items in the previous step, we will assume for simplicity that this is not necessary since it can only affect the additive constant in any case.) Since the minimum bin level is $1-\gamma+\varepsilon'$ (in bin $113m/64$) at the time the item of size $\gamma-\varepsilon'$ is packed, this item will go in bin $113m/64$ (recall the decreasing levels in the region of bins from $(105m/64)+1$ to $113m/64$). Just after packing this item in bin $113m/64$, we remove from that bin the other item, which has size $1-\gamma+\varepsilon'$. This procedure is then iterated back to bin $(105m/64)+1$, with the end effect of replacing each item of size $1-\gamma+i\varepsilon'>\frac{1}{2}$ with a complementary item of size $\gamma-i\varepsilon'<\frac{1}{2}$, and these new items are in *increasing* order by size. Since we replaced items larger than $\frac{1}{2}$ by items smaller than $\frac{1}{2}$, the sum of the item sizes remains less than m. This completes the extension to a packing using items no larger than $\frac{1}{2}$.

Stage 3. The next step is to remove ε-items from bins 1 through m until their levels are at $\gamma<\frac{1}{2}$ and to apply the shaving procedures to reduce the level of each bin i, $m+1\leqq i\leqq(3m/2)-2$, to $1-(m/2i)$ (some may already be at this level). We then use the amount removed to construct $m/8$ new items of size $1-\gamma+i\varepsilon'$, $1\leqq i\leqq m/8$, and we pack these in decreasing sequence into bins $(105m/64)+1$ to $113m/64$. These items will fit exactly with the corresponding complementary items. A straightforward calculation shows that the sum of the item sizes does not exceed m, and arguments similar to those used earlier show the continued existence of an optimal packing into m or fewer bins.

Stage 4. We are now in a position to execute the same procedure used in Theorem 4, to extend the $11m/4$ example, for extending our $113m/64$ example, repeatedly removing all but a smallest item from a bin and packing a new item slightly larger than the gap remaining in that bin, working backwards according to bin index. Each of the new items will then start a new bin on the right, and the ε-items remaining at the end can be used to form additional items of size 1. It can be verified that this yields m additional items larger than $\frac{1}{2}$ (and m additional bins), for a grand total of $177m/64$ bins (within an additive constant). Furthermore, the pairing process described

in the proof of Theorem 4 carries over here to show that an optimum packing has no more than m bins. Thus we have our desired example.

The parameters $105m/64$ and $113m/64$ in the example were chosen for simplicity. Maximizing over these parameters yields values of $1.63m$ and $1.77m$, respectively, thus providing a $2.77m$-example. Further tightening of the construction can no doubt improve this further, although significant improvements do not appear to be possible without altering or generalizing the method we have used.

5. Lower bounds for arbitrary on-line algorithms. Given the performance bounds that we have derived for the First Fit algorithm, it is natural to ask how these bounds would compare to those for other on-line algorithms. In this section we address this question, showing that FF performs essentially as well (in the worst-case sense) as any other on-line algorithm. We also show that our use of OPT_R, rather than OPT_{NR}, as the performance standard for our comparisons makes very little difference as to the final outcome. Once again we begin with the cases in which all items are no more than $1/k$ in size, $k \geqq 2$.

THEOREM 5. *If A is any on-line dynamic bin packing algorithm, then for any integer $k \geqq 2$, we have*

$$W_R(A, k) \geqq W_{NR}(A, k) \geqq \frac{k+1}{k} + \frac{1}{k(k+1)} = 1 + \frac{k+2}{k(k+1)}.$$

Proof. Specifically, we will show that there exist lists L with $\mathrm{OPT}_R(L)$ arbitrarily large and all items p_i satisfying $s_i \leqq 1/k$ such that

$$A(L) \geqq \left(1 + \frac{k+2}{k(k+1)}\right)(\mathrm{OPT}_{NR}(L) - 1).$$

We provide a procedure for generating such examples, tailored to algorithm A, with the nature of the later items being dependent on how A packs the earlier ones. The input to our procedure is the algorithm A and an integer m that is divisible by $k(k+1)$, where $m+1$ will serve as our upper bound for OPT_{NR}. We describe the procedure in much the same style as we described our previous lower bound constructions, the only significant difference being that in this case we can branch to different constructions based on how A has packed the items encountered so far. The procedure, along with a proof that it works (we initially argue only that $\mathrm{OPT}_R(L) \leqq m+1$, leaving until the end the proof that $\mathrm{OPT}_{NR}(L) \leqq m+1$), proceeds as follows:

Step 1. Choose $\varepsilon = 1/[k(k+1)(1000)m]$.

Step 2. Create $(m+1)/\varepsilon$ items of size ε and have A pack them. (Note that the list so far can be packed into $m+1$ bins and in fact *requires* that many bins.)

Step 3. If there are fewer than $(k+2)m/(k+1)$ nonempty bins in the packing, go to Step 4. Otherwise, do the following:

3.1. Remove all but a single ε-item from each of $(k+2)m/(k+1)$ of the bins and remove everything from all other nonempty bins.

3.2. Create km items of size $1/k$ and have A pack them. (Note that the bins with one ε-item can each receive at most $k-1$ such items and the other bins can each receive at most k, so there must be at least

$$\left(km - (k-1)\left(\frac{k+2}{k+1}\right)m\right)\Big/k = \frac{2m}{k(k+1)}$$

new bins started, for a total of at least

$$m\left(\frac{k+2}{k+1}+\frac{2}{k(k+1)}\right)=m\left(1+\frac{k+2}{k(k+1)}\right)$$

bins, as desired. Moreover, an optimum packing could place all km items of size $1/k$ in m bins, with the ε-items all going in a single additional bin.)

3.3. Halt, with $A(L)\geqq m(1+(k+2)/k(k+1))$ for this case.

Step 4. There are currently $m'=((k+2)m/(k+1))-d$ nonempty bins for some $d>0$. Remove as few ε-items as possible from each bin so that its level has the form $(i/(k+1))+\varepsilon$ for some i, $0\leqq i\leqq k$. For each such i, let A_i be the set of currently nonempty bins with level $(i/(k+1))+\varepsilon$, and let $a_i=|A_i|$. Note that we have the following relationships:

$$\sum_{i=0}^{k} a_i = m', \tag{4.1}$$

$$a_k \geqq \left(\frac{m}{k+1}+dk\right)+(k-1)a_0. \tag{4.2}$$

(The latter inequality follows from the fact that, using upper bounds on the contents of each bin at the end of Step 2, we have

$$a_k+\frac{k}{k+1}(m'-(a_k+a_0))+\frac{a_0}{k+1}\geqq m,$$

and hence

$$a_k\geqq(k+1)\left(m-\frac{k}{k+1}m'+\frac{k-1}{k+1}a_0\right).$$

Substituting $((k+2)m/(k+1))-d$ for m' yields (4.2).)

Step 5. Let $X=\sum_{i=0}^{k-1} a_i(k-i)$. This is the maximum number of items of size $(1/(k+1))+\varepsilon$ that would fit in the m' currently nonempty bins. Create $X+(m/(k+1))+dk$ items of this size and have A pack them.

(Note that all the items in the packing can still be packed into $m+1$ bins. The sum of the item sizes is bounded above by

$$\left(\frac{k+2}{k+1}m-d\right)\left(\frac{k}{k+1}+k\varepsilon\right)+\left(\frac{m}{k+1}+dk\right)\left(\frac{1}{k+1}+\varepsilon\right)$$

$$=m\left(\frac{k(k+2)}{(k+1)^2}+\frac{1}{(k+1)^2}\right)+\varepsilon m\left(\frac{k(k+2)}{k+1}+\frac{1}{k+1}\right)=m+\varepsilon m(k+1)<m+1.$$

Thus all we need show is that the items larger than ε can be packed into $m+1$ bins, since the choice of ε ensures that the remaining gaps can be filled exactly with ε-items. There are at most m' items larger than ε, since each has size $(1/(k+1))+\varepsilon$ and no bin had more than that amount removed in Step 4. These can be packed, k per bin, into

$$\frac{m'}{k}\leqq\frac{k+2}{k+1}\cdot\frac{m}{k}\leqq m$$

bins (since $k\geqq 2$), as required.)

Step 6. Delete all items of size $(1/(k+1))+\varepsilon$ that were placed in the A_i bins, $0 \leqq i \leqq k$, and remove enough additional items of that size (if necessary) from the other bins so that exactly $(m/(k+1))+dk$ such items remain. Note that this deletes precisely X of those items. Next, for $1 \leqq i \leqq k$, delete ε-items from each bin of type A_i to reduce its level from $(i/(k+1))+\varepsilon$ to $((i-1)/k)+\varepsilon$, a reduction of $(1+k-i)/k(k+1)$ per bin. Finally, from a_0 of the A_k bins (note that there are at least a_0 bins of type A_k by (4.2) and the fact that $k \geqq 2$), remove ε-items totaling $1/k$ in size, reducing the bin level to $((k-2)/k)+\varepsilon$.

The sum of the sizes of the items removed in this step is at least

$$\begin{aligned}\frac{X}{k+1}+\sum_{i=1}^{k}\left(\frac{1+k-i}{k(k+1)}\right)a_i+\frac{a_0}{k} &= \frac{X}{k+1}+\frac{(m'-a_0)}{k(k+1)}+\frac{X-ka_0}{k(k+1)}+\frac{a_0}{k} \\ &= \frac{X}{k}+\frac{m'}{k(k+1)}.\end{aligned} \tag{6.1}$$

Step 7. So long as the sum of the sizes of all items in the packing is less than $m+\varepsilon(m'+(m/(k+1))+dk)$, repeatedly create an item of size $1/k$ and have A pack it. (Note that this will never cause the sum of the item sizes to exceed

$$m+\varepsilon\left(m'+\frac{m}{k+1}+dk\right)+\frac{1}{k}<m+1.$$

Note also that by (6.1) at least X items of size $1/k$ will be created.)

Step 8. Halt. We claim that if M is the number of nonempty bins in the packing constructed by A, then

$$M \geqq m\left(1+\frac{k+2}{k(k+1)}\right),$$

and furthermore that for this list $\mathrm{OPT}_R(L)=m+1$.

We first prove that the number of bins in the packing constructed by A is as stated. For each i, $1 \leqq i \leqq k$, let b_i denote the number of bins at the end of Step 6 that contained i items of size $(1/(k+1))+\varepsilon$ and no ε-items. Note that $\sum_{i=1}^{k} ib_i = (m/(k+1))+dk$. Also note that a bin with i such items at the end of Step 6 has level at most

$$\frac{i}{k+1}+\frac{k-i}{k}+i\varepsilon$$

at the end of Step 7. Also, at the end of Step 7 no bin of type A_i, $0 \leqq i \leqq k$, can have level exceeding $((k-1)/k)+\varepsilon$. If we let b_0 denote the number of nonempty bins that contain only items of size $1/k$ at the end of Step 7, then we have

$$\begin{aligned}m+\left(m'+\frac{m}{k+1}+dk\right)\varepsilon &\leqq m'\left(\frac{k-1}{k}+\varepsilon\right)+b_0+\sum_{i=1}^{k} b_i\left(\frac{i}{k+1}+\frac{k-i}{k}+i\varepsilon\right) \\ &= m'\left(\frac{k-1}{k}\right)+\sum_{i=1}^{k} b_i\left(\frac{k^2+k-i}{k(k+1)}\right)+b_0+\varepsilon\left(m'+\frac{m}{k+1}+dk\right),\end{aligned}$$

or

$$m \leqq m'\left(\frac{k-1}{k}\right)+\sum_{i=0}^{k} b_i-\left(\frac{m}{k+1}+dk\right)\Big/(k(k+1)). \tag{7.1}$$

We also have $M = m' + \sum_{i=0}^{k} b_i$. Combining this with (7.1), we obtain

$$\begin{aligned} M &\geqq m + \frac{m'}{k} + \left(\frac{m}{k+1} + dk\right) \Big/ k(k+1) \\ &= m\left(1 + \frac{k+2}{k(k+1)} + \frac{1}{k(k+1)^2}\right) - \frac{d}{k} + \frac{d}{k+1} \\ &= m\left(1 + \frac{k+2}{k(k+1)}\right) + \frac{1}{k(k+1)}\left(\frac{m}{k+1} - d\right). \end{aligned}$$

Since $d = ((k+2)m/(k+1)) - m'$ and $m' \geqq m$, we have $d \leqq m/(k+1)$, and the desired lower bound for M follows.

We next show that the items in existence at the end of Step 7 can be packed into $m+1$ bins. By the operation of this step, we know that the sum of the item sizes is less than $m+1$. The idea of our packing is as follows: Combine each item of size $(1/(k+1)) + \varepsilon$ with a collection of ε-items totaling $(1/k(k+1)) - \varepsilon$ in size, thus obtaining an ensemble of size $1/k$. These ensembles are then combined with the items of size $1/k$ and packed k per bin, every bin but one being completely full. The remaining ε-items can be used to fill up the rest of the packing, and since there will be no wasted space and the sum of the item sizes is less than $m+1$, no more than $m+1$ bins will be used. This construction principle will work so long as there are enough ε-items to make all the ensembles, i.e., to mate with all the items of size $(1/(k+1)) + \varepsilon$. Since there are precisely $(m/(k+1)) + dk$ items of size $(1/(k+1)) + \varepsilon$ left at this step, the sum of the sizes of the ε-items must be at least

$$\left(\frac{m}{k+1} + dk\right)\left(\frac{1}{k(k+1)} - \varepsilon\right)$$

in order for our construction to work.

The sum of the ε-items in the packing can be estimated by noting that at the end of Step 6 it must be at least

$$\frac{k-1}{k} a_k - \frac{a_0}{k}$$

which, applying (4.2) and the assumption that $k \geqq 2$, yields

$$\begin{aligned} \frac{k-1}{k} a_k - \frac{a_0}{k} &\geqq \frac{k-1}{k}\left[\left(\frac{m}{k+1} + dk\right) + (k-1)a_0\right] - \frac{a_0}{k} \\ &\geqq \frac{k-1}{k}\left(\frac{m}{k+1} + dk\right) + a_0\left(\frac{(k-1)^2 - 1}{k}\right) \geqq \frac{1}{k}\left(\frac{m}{k+1} + dk\right). \end{aligned}$$

Thus the number of ε-items in the packing is sufficient for us to form our ensembles, and hence we have $\mathrm{OPT}_R(L) = m+1$.

Finally we must show that $\mathrm{OPT}_{NR}(L) = m+1$, i.e., that L can be packed into $m+1$ bins with no rearrangement. In the case where we halted in Step 3, the argument is easy. The nonrearranged packing starts out with all the ε-items packed into $m+1$ bins in such a way that the ε-items which survive Step 3.1 are all in bin $m+1$. Thus, after Step 3.1, the first m bins are completely empty, and the km items of size $1/k$ created in Step 3.2 can be packed, k per bin, into these bins.

In the case where we halted at Step 8, it is easier to describe the optimal packing by running time "backwards." We start with the packing as it exists at the end of

Step 7 and proceed through the steps in reverse order. When we pass the moment of destruction (removal) for an item, we add it to the packing. When we pass the moment of creation for an item, we delete it from the packing.

At the end of Step 7, the items of size $(1/(k+1))+\varepsilon$ and $1/k$ are packed k per bin, with the space left over filled up with ε-items. Step 7 removes all the items of size $1/k$, of which there are at least X. Step 6 then puts back in precisely X items of size $(1/(k+1))+\varepsilon$, plus a number of ε-items. Each item of size $(1/(k+1))+\varepsilon$ can then be placed in a spot vacated by an item of size $1/k$, with the ε-items filling up the left-over space. Step 5 then deletes all the items of size $(1/(k+1))+\varepsilon$, and the ε-items created in Step 4 can fill up the gaps. Step 3 does nothing in this case, and Step 2 simply deletes everything. Thus, as required, we have $\mathrm{OPT}_{NR}(L)=m+1$. □

THEOREM 6. *For any on-line dynamic bin packing algorithm, A,*

$$W_R(A)\geqq\tfrac{5}{2}$$

and

$$W_{NR}(A)\geqq\tfrac{43}{18}\sim 2.388\cdots.$$

Proof. Specifically, we show how to construct lists L_1 and L_2 with arbitrarily large optimum packings such that

$$A(L_1)\geqq\tfrac{43}{18}(\mathrm{OPT}_{NR}(L_1)-1),$$
$$A(L_2)\geqq\tfrac{5}{2}(\mathrm{OPT}_{R}(L)-1).$$

To do this, we modify and extend the construction procedure given in the proof of Theorem 5, specialized to $k=2$.

Steps 1 and 2 are performed exactly as given, for both L_1 and L_2. If the number of nonempty bins at the end of Step 2 is at least $(k+2)m/(k+1)=4m/3$, the remainder of the construction will also be the same for L_1 and L_2. In this case, we perform Steps 3.1 and 3.2 as before, obtaining a set of items of size ε and size $\frac{1}{2}$ for which A uses at least $5m/3$ bins. We then complete the construction by replacing Step 3.3 by the following steps:

3.3a. Remove all items of size $\frac{1}{2}$ in the $4m/3$ bins that contain ε-items. From the bins that contain only items of size $\frac{1}{2}$, of which there are at least $m/3$, remove items of size $\frac{1}{2}$ until there are exactly $m/3$ such bins, each containing a single item of size $\frac{1}{2}$. Note that at this point the sum of the sizes of the items in the packing is $(m/6)+(4m\varepsilon)/3$.

3.3b. Create $5m/6$ items of size 1 and have A pack them. These must go in $5m/6$ new bins, yielding a total of at least $(5m/3)+(5m/6)=5m/2$ bins.

3.3c. Halt. It is easy to see that we have $\mathrm{OPT}_R(L)=\mathrm{OPT}_{NR}(L)=m+1$, so the desired lists have been constructed in this case.

In the case where, for some $d>0$, only $(4m/3)-d$ bins are nonempty after Step 2, the constructions for L_1 and L_2 differ. We first describe the modifications necessary for constructing L_1. Step 3, as before, does nothing in this case, and Step 4 is executed without change. However, we revise Step 5 as follows (with X defined as before):

5a. Create $X+(m/3)+2d$ items of size $\frac{1}{3}$ (instead of $\frac{1}{3}+\varepsilon$) and have A pack them. Since the m' bins that were nonempty after Step 4 must all be filled at most to level $\frac{2}{3}+\varepsilon$, there will be a total of at least $4m/3$ nonempty bins at this point, at least d of which contain only items of size $\frac{1}{3}$.

5b. Choose a set D of d bins that contain only items of size $\frac{1}{3}$, and remove all items of size $\frac{1}{3}$ from the packing except for one in each bin in D.

5c. Create $X+(m/3)+d$ items of size $\frac{1}{3}+\varepsilon$ and have A pack them.

Note that this modified Step 5 leaves a situation identical to that at the end of the original Step 5, except for the bins in D, which have one item of size $\frac{1}{3}$ instead of size $\frac{1}{3}+\varepsilon$. Steps 6 and 7 are performed as before, and Step 8 is replaced by the following:

8a. There are at least $5m/3$ nonempty bins, at least $m/3$ of which are not A_i-bins or D-bins. Choose a set B of $m/3$ of these latter bins. Remove all items not in A_i-bins, D-bins, or B-bins, and remove all but a single smallest item from each of the D-bins and B-bins. (Note that the items deleted all have sizes of either $\frac{1}{2}$ or $\frac{1}{3}+\varepsilon$.) For each item of size $\frac{1}{3}+\varepsilon$ that was deleted, remove an additional $\frac{1}{6}-\varepsilon$ in ε-items from an A_2-bin. There must be enough of these in the packing since, as we argued following Step 8 in the proof of Theorem 5, there were enough ε-items left in A_2-bins so that *every* item of size $\frac{1}{3}+\varepsilon$ in the packing after Step 7 could be matched with such a collection of ε-items. This also means that all items of size $\frac{1}{3}+\varepsilon$ remaining in the packing at this point can still be matched with $\frac{1}{6}-\varepsilon$ of ε-items remaining in A_2-bins. Form such a matching and remove all other ε-items from the packing, leaving a single ε-item if the bin would otherwise become empty. The sum of the sizes of the items left in the packing at this point is no more than

$$m'\varepsilon+\frac{d}{3}+\frac{1}{2}\frac{m}{3}\leqq\frac{m}{6}+\frac{d}{3}+\frac{4m\varepsilon}{3}.$$

8b. Create $\lceil(5m/6)-(d/3)\rceil$ items of size 1 and have A pack them. Each requires a new bin, so that we end up with a total of

$$\frac{5m}{3}+\left\lceil\frac{5m}{6}-\frac{d}{3}\right\rceil\geqq\frac{5m}{2}-\frac{m}{9}=\frac{43}{18}m.$$

(At the same time, all the items can be packed into $m+1$ bins. The items of size 1 use up $\lceil(5m/6)-(d/3)\rceil$ bins, and the items of size $\frac{1}{2}$ and $\frac{1}{3}+\varepsilon$, together with their associated ε-items, go two per bin and hence use $m/6$ additional bins (recall that m is divisible by $k(k+1)=6$). This leaves d items of size $\frac{1}{3}$ and at most m' ε-items. The items of size $\frac{1}{3}$ go three per bin into $\lfloor d/3\rfloor$ bins, with the left-over items of size $\frac{1}{3}$ and the remaining ε-items fitting in a single remaining bin. If we let $d'=d-3\cdot\lfloor d/3\rfloor$, we have that the total number of bins used in this packing is

$$\left\lceil\frac{5m}{6}-\frac{d}{3}\right\rceil+\frac{m}{6}+\left\lfloor\frac{d}{3}\right\rfloor+1=\left(\frac{5m}{6}-\frac{d}{3}+\frac{d'}{3}\right)+\frac{m}{6}+\left(\frac{d}{3}-\frac{d'}{3}\right)=m+1,$$

as required.)

8c. Halt. L_1 has been constructed.

It is straightforward to check that L_1 can be packed into $m+1$ bins, *without* rearrangement, again by running the packing process backwards. At Step 8a the items of size 1 can be replaced by ensembles of size $\frac{1}{2}$, along with possibly some additional ε-items. (Our sum-of-sizes arguments allow for the possibility of perhaps *one* extra ensemble, but if it exists there must be room for it in the bin with left-over ε's and $\frac{1}{3}$'s.) We then continue backwards as in the proof of Theorem 1.

The construction of the list L_2 in the case that there are only $(4m/3)-d$ nonempty bins following Step 2 proceeds exactly as the construction of L_1 through Step 5, using the same modification of Step 5 to ensure that there are d D-bins containing only a single item of size $\frac{1}{3}$. For future use, we now introduce some additional notation.

Let E denote the set of bins that contain only items of size $\frac{1}{3}+\varepsilon$ at the end of Step 5, and let $e=|E|$. Partition D and E into D_1 and D_2, and E_1 and E_2, respectively, where the subscript corresponds to the number of items in each bin in the set. Let $d_1=|D_1|$, $d_2=|D_2|$, $e_1=|E_1|$, and $e_2=|E_2|$. Let A_0' denote the subset of bins from A_0 that contain just a single ε-item at the end of Step 5, and let $a_0'=|A_0'|$. Figure 2 illustrates the possible contents of the bins of each of these types at the end of Step 5.

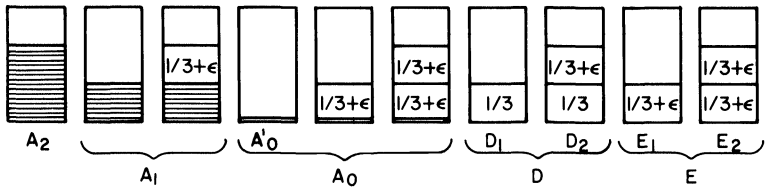


FIG. 2. *Types of bins at the end of Step* 5.

LEMMA 6.1. *The following relationships hold among the set sizes*:

(R1) $$a_2 \geqq \frac{m}{3}+2d+a_0,$$

(R2) $$a_0 \leqq \frac{m}{2}-\frac{3d}{2} \quad \text{and} \quad d \leqq \frac{m}{3}-\frac{2a_0}{3},$$

(R3) $$d_1+e_1+2d_2+2e_2 \geqq \frac{m}{3}+2d+2a_0',$$

(R4) $$e \geqq \frac{m}{6}+a_0'+\frac{e_1+d_1}{2}.$$

Proof of Lemma 6.1. (R1) is simply a restatement of (4.2) for $k=2$. (R2) follows from the fact that

$$m \leqq \frac{a_0}{3}+a_1+a_2=\frac{a_0}{3}+\left(\frac{4m}{3}-a_0-d\right)$$

(since, at the end of Step 2, each A_0 bin is at most $\frac{1}{3}$ full). (R3) follows from the fact that at most $|X|-2a_0'$ items of size $\frac{1}{3}$ or larger go in A_i bins during Step 5. (R4) follows from (R3), since

$$2(d_2+e_2)+(d_1+e_1) \geqq \frac{m}{3}+2d+2a_0'$$

implies

$$d+e \geqq \frac{1}{2}\left(\frac{m}{3}+2d+2a_0'\right)+\frac{1}{2}(e_1+d_1)$$

which implies

$$e \geqq \frac{m}{6}+a_0'+\frac{e_1+d_1}{2}.$$ □

Following the completion of Step 5, the construction of L_2 proceeds as follows:

Step 6. Remove ε-items totaling $\frac{1}{6}$ in size from each A_2-bin and totaling $\frac{1}{6}+\varepsilon$ from each A_1-bin that contains an item of size $\frac{1}{3}+\varepsilon$, thus reducing the level of each

such bin to $\frac{1}{2}+\varepsilon$. Then remove a single item of size $\frac{1}{3}+\varepsilon$ from each A_0-bin that contains two such items. The types of bins are now as shown in Fig. 3. If $e > \lceil (m/3)+(5a_0/6) \rceil$, repeatedly remove all items from a single E-bin until equality is reached, set $F = \varnothing$, and go on to Step 8.

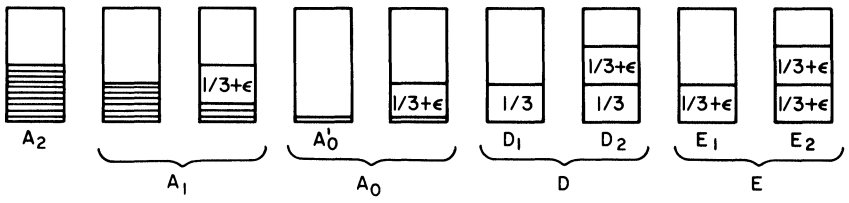


FIG. 3. *Types of bins at the end of Step* 6.

Step 7. So long as the sum of the bin levels, with each rounded down to the nearest multiple of $\frac{1}{6}$, is less than m and the total number of items larger than ε is less than $2m$, do the following:

7.1. Create an item of size $\frac{1}{2}$ and have A pack it.

7.2. If this item is placed in an A_1 bin (in which case that bin contained only ε-items and had a level of $\frac{1}{3}+\varepsilon$), remove from that bin all but one of the ε-items. If this item is placed in an A_0 bin that contained an item of size $\frac{1}{3}+\varepsilon$, delete the item of size $\frac{1}{3}+\varepsilon$.

(Note that Step 7 guarantees that at all times during its execution the current set of items can be packed into $m+1$ bins. Let us denote by F the set of bins that contain only items of size $\frac{1}{2}$ at this point, let $f = |F|$, and let a_0'' be the number of A_0 bins that still contain only a single item, of size ε. Note that $a_0'' \leqq a_0' \leqq a_0$. We claim that the number M of nonempty bins, i.e., $M = a_2+a_1+a_0+d+e+f$, satisfies

$$M \geqq \left\lceil \frac{5m}{3}+\frac{5a_0''}{6} \right\rceil.$$

To see this, first suppose Step 7 halted because there were $2m$ items of size larger than ε. Then we must have

$$2m \leqq a_1+(a_0-a_0'')+2(d+e+f)$$

or

$$d+e+f+\frac{(a_1+a_0)}{2} \geqq m+\frac{a_0''}{2}.$$

Hence

$$\begin{aligned} M &\geqq \frac{a_2}{2}+\frac{a_2+a_1+a_0}{2}+m+\frac{a_0''}{2} \\ &\geqq \frac{1}{2}\left(\frac{m}{3}+2d+a_0\right)+\frac{1}{2}\left(\frac{4m}{3}-d\right)+m+\frac{a_0''}{2} \qquad \text{(by (R1))} \\ &= \frac{11m}{6}+\frac{d}{2}+\frac{a_0+a_0''}{2} \geqq \frac{11m}{6}+\frac{2a_0''}{2} > \frac{5m}{3}+a_0'' > \frac{5m}{3}+\frac{5a_0''}{6} \end{aligned}$$

and, since M is an integer, the claim follows in this case.

On the other hand, if Step 7 halted because the sum of the rounded bin levels became too large, then we have

$$m \leqq \tfrac{1}{2}(a_2+a_1+a_0-a_0'')+\tfrac{5}{6}(d_1+e_1)+\tfrac{2}{3}(d_2+e_2)+f,$$

or

$$M \leqq m+\frac{1}{2}(a_2+a_1+a_0)+\frac{a_0''}{2}+\frac{1}{6}(d+e)+\frac{2}{6}(d_2+e_2).$$

Applying (R3) and $a_0+a_1+a_2=(4m/3)-d$, we obtain

$$M \geqq m+\frac{1}{2}\left(\frac{4m}{3}-d\right)+\frac{a_0''}{2}+\frac{1}{6}\left(\frac{m}{3}+2d+2a_0'\right)$$

$$\geqq \frac{5m}{3}+\frac{5a_0''}{6}+\frac{1}{6}\left(\frac{m}{3}-d\right),$$

and hence

$$M \geqq \left\lceil \frac{5m}{3}+\frac{5a_0''}{6} \right\rceil$$

since $d \leqq m/3$ and M is an integer.)

Step 8. Remove all but a total of $\frac{1}{3}+\varepsilon$ in ε-items from each A_2 bin. These are now identical to the A_1 bins that received neither an item of size $\frac{1}{3}+\varepsilon$ nor an item of size $\frac{1}{2}$. Call the joint class of such bins A_2'. Note that $a_2' = |A_2'| \geqq a_2$. Remove from each A_1 bin that contains an item of size $\frac{1}{3}+\varepsilon$ all items except for a single item of size $\frac{1}{3}+\varepsilon$ and a single ε-item. This makes them identical to some of the A_0 bins. Call this collection of identical bins A_1'. Let A_0''' be the subset of the A_0 bins that contain a single item of size $\frac{1}{2}$ and a single ε-item, with A_0'' remaining as the set of A_0 bins that contain only a single ε-item. Note that

$$a_0''' = |A_0'''| \leqq a_1+a_0-a_0'' = \frac{4m}{3}-d-a_2-a_0''.$$

Remove all but a single item of size $\frac{1}{3}$ from each D_1 bin, all but a single item of size $\frac{1}{3}+\varepsilon$ from each E bin, and all but a single item of size $\frac{1}{2}$ from each F bin. Figure 4 illustrates the types of bins remaining at this point. If $d \leqq 5a_0''/6$, set $G = \varnothing$ and go to Step 10.

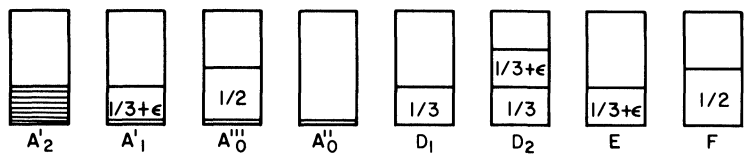


FIG. 4. *Types of bins at the end of Step* 8.

Step 9. Create $a_0''+d_1+\lceil d-5a_0''/6\rceil \leqq d+d_1+a_0''/6+\frac{5}{6}$ items of size $\frac{2}{3}$ and have A pack them. They can only be placed in A_0'' bins, D_1 bins, or new bins, so at least $\lceil d-5a_0''/6\rceil$ of them start new bins. Let G denote the set of such new bins. Note that at this point the number of nonempty bins is at least

$$\left\lceil \frac{5m}{3}+\frac{5a_0''}{6} \right\rceil + \left\lceil d-\frac{5a_0''}{6} \right\rceil \geqq \frac{5m}{3}+d.$$

(We claim once again that the current collection of items can be packed into $m+1$ bins. The sum of their sizes is not too large, since it can be bounded by

$$\frac{2}{3}\left(a_0'' + d_1 + \left\lceil d - \frac{5a_0''}{6}\right\rceil\right) + \frac{1}{3}\left(\frac{4m}{3} + e\right) - \frac{a_0''}{3} + \frac{a_0'''}{3} + \frac{d_2}{3} + \frac{f}{2} + \frac{5m\varepsilon}{3}$$

$$\leqq \frac{2}{3}\left(d_2 + 2d_1 + \frac{a_0''}{6} + \frac{5}{6}\right) + \frac{4m}{9} + \frac{e}{3} - \frac{a_0''}{3} + \frac{1}{6}\left(\frac{4m}{3} - d - a_2 - a_0''\right)$$

$$+ \frac{d_2}{3} + \frac{1}{2}\left(\left\lceil \frac{5m}{3} + \frac{5a_0''}{6}\right\rceil - \left(\frac{4m}{3} + e\right)\right) + \frac{5m\varepsilon}{3}$$

$$\leqq \frac{5m}{6} + \frac{5d_2}{6} + \frac{7d_1}{6} + \frac{a_0''}{36} - \frac{e + a_2}{6} + \frac{5}{9} + \frac{5}{12} + \frac{5m\varepsilon}{3}$$

$$\leqq \frac{5m}{6} + \frac{5d_2}{6} + \frac{7d_1}{6} + \frac{a_0''}{36} - \frac{1}{6}\left(\frac{m}{2} + 2d + 2a_0'' + \frac{d_1}{2}\right) + \frac{35}{36} + \frac{5m\varepsilon}{3}$$

(by (R1) and (R2))

$$\leqq \frac{3m}{4} + \frac{d_2}{2} + \frac{9d_1}{12} - \frac{11a_0''}{12} + 1 \leqq \frac{3m}{4} + \frac{3d}{4} + 1 \leqq \frac{3m}{4} + \frac{3}{4}\left(\frac{m}{3}\right) + 1 = m + 1.$$

The total number of bins needed to pack the items larger than ε is at most

$$\left(d + d_1 + \frac{a_0''}{6} + \frac{5}{6}\right) + \left\lceil \frac{1}{2}\left(\frac{4m}{3} - d - a_0'' - a_2' + d_2 + e + f\right)\right\rceil$$

since all d items of size $\frac{1}{3}$ can go in bins with items of size $\frac{2}{3}$, and all the items of size $\frac{1}{3}+\varepsilon$ and $\frac{1}{2}$ can go two per bin. This quantity is bounded above by

$$\frac{2m}{3} + \frac{d}{2} + d_1 + \frac{d_2}{2} - \frac{a_0''}{3} - \frac{1}{2}\left(\frac{m}{3} + 2d + a_0''\right) + \frac{1}{2}\left(\left\lceil \frac{5m}{3} + \frac{5a_0''}{6}\right\rceil - \frac{4m}{3}\right) + \frac{5}{6}$$

$$\leqq \frac{2m}{3} + \frac{d}{2} - \frac{5a_0''}{12} + \frac{5}{6} + \frac{5}{12} \leqq \frac{2m}{3} + \frac{1}{2}\left(\frac{m}{3}\right) + \frac{5}{4} = \frac{5m}{6} + \frac{5}{4} < m + 1.$$

The items of size ε can then be used to fill in the remaining gaps.)

Step 10. Remove all but a single smallest item from each bin, which may be size ε, $\frac{1}{3}$, $\frac{1}{3}+\varepsilon$, $\frac{1}{2}$, or $\frac{2}{3}$. Letting $g = |G|$ denote the number of items of size $\frac{2}{3}$ remaining, create

$$\frac{5m}{2} - \left(g + \left\lceil \frac{5m}{3} + \frac{5a_0''}{6}\right\rceil\right)$$

items of size 1 and have A pack them.

(Since each item must start a new bin, we are guaranteed that there will be $5m/2$ nonempty bins at the end of this step. Moreover, the collection of items can still be packed into $m+1$ bins. To show this, we consider two cases:

Case (i). $d \leqq 5a_0''/6$. In this case, $g = 0$ by Step 8. There are thus no more than $(5m/6) - (5a_0''/6)$ items of size 1, each of which requires an entire bin in the packing. The remaining

$$\left\lceil \frac{5m}{3} + \frac{5a_0''}{6}\right\rceil - \left(\frac{4m}{3} - d\right)$$

items of size greater than ε can be packed, two per bin, into at most

$$\frac{1}{2}\left(\frac{m}{3}+\left(\frac{5a_0''}{6}+d\right)+\frac{5}{6}\right)\leqq\frac{m}{6}+\frac{5a_0''}{6}+\frac{5}{12}$$

bins, for a total of at most $m+\frac{5}{12}$ bins. The remaining $(4m/3)-d$ ε-items can then be packed into the space available in any one of the bins containing an item of size $\frac{1}{3}+\varepsilon$.

Case (ii). $d>5a_0''/6$. In this case $g\geqq|d-5a_0''/6|>0$. The items of size 1 and size $\frac{2}{3}$ use up $(5m/6)-(5a_0''/6)$ bins, but now all but

$$d-\left\lceil d-\frac{5a_0''}{6}\right\rceil\leqq\frac{5a_0''}{6}$$

of the items of size $\frac{1}{3}$ in D bins will fit in a bin that contains an item of size $\frac{2}{3}$. The remaining items of size greater than ε can go two to a bin, for a total number of bins that is at most

$$\frac{5m}{6}-\frac{5a_0''}{6}+\left\lceil\frac{1}{2}\left(\frac{5a_0''}{6}+\left\lceil\frac{5m}{3}+\frac{5a_0''}{6}\right\rceil-\frac{4m}{3}\right)\right\rceil\leqq m+\frac{11}{12}<m+1.$$

The remaining items of size ε can again be packed into the space available in some bin containing an item of size $\frac{1}{3}+\varepsilon$.

Thus in both cases we have that at most $m+1$ bins are required to pack the set of items in the packing following Step 10.)

Step 11. Halt. The desired list L_2 has been constructed. □

6. Some concluding remarks. For the cases in which all item sizes are bounded by $1/k$, $k\geqq 2$, our performance bounds for FF are reasonably tight and quite close to the best that can be achieved by *any* on-line algorithm, even if we use the nonrearranged optimum as our standard for comparison. For example, when $k=2$ we have

$$1.75\leqq W_R\ (\mathrm{FF}, 2)\leqq 1.788$$

while

$$1.66\leqq W_{NR}(A, 2)\leqq W_R\ (A, 2)$$

for any on-line packing algorithm A.

For the unrestricted ($k=1$) case, our bonds are somewhat less tight, with

$$2.75\leqq W_R\ (\mathrm{FF})\leqq 2.897$$

and

$$2.5\leqq W_R\ (A),\qquad 2.38\leqq W_{NR}(A)$$

for arbitrary on-line algorithms A. For practical purposes, however, it is probably sufficient simply to know that all these algorithms have worst-case performance in the vicinity of $2\frac{1}{2}$ to 3 times optimal.

It is interesting to note also the rather large increase in the performance bound for FF when going from $k=2$ to $k=1$. For dynamic bin packing the increase is approximately 55%, from about 1.8 to about 2.8, which is to be contrasted with the analogous increase of only 13%, from 1.5 to 1.7, for static bin packing.

Extensions on the work presented here might take any of several directions. There is certainly room for improvements in our bounds, and methodological and esthetic considerations provide ample justification for seeking such improvements. In

particular, the bounds for the $k = 1$ case deserve further attention, and the general on-line lower bound for W_{NR} seems like an especially good candidate for additional effort. It would also be of interest to obtain bounds on the ratio $\mathrm{OPT}_{NR}/\mathrm{OPT}_{R}$.

More generally, having studied the *worst-case* performance of on-line dynamic bin packing algorithms, it is natural to ask about the *expected* performance of such algorithms, under various probabilistic assumptions about the arrival times, departure times, and sizes of the items. Such results have been difficult to obtain even for static bin packing (e.g., see [1]), but perhaps something useful can still be said for the dynamic case. Finally, although we have restricted our attention to on-line algorithms in this paper, there is no reason why analogous questions might not be asked about *arbitrary* fast approximation algorithms for dynamic bin packing.

REFERENCES

[1] E. G. COFFMAN, JR., K. SO, M. HOFRI AND A. C. YAO, *A stochastic model of bin-packing*, Inform. and Control, 44 (1980), pp. 105–115.

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

[3] M. R. GAREY AND D. S. JOHNSON, *Approximation algorithms for bin packing problems: A survey*, in Analysis and Design of Algorithms in Combinatorial Optimization, G. Ausiello and M. Lucertini, eds., CISM Courses and Lectures No. 266, Springer-Verlag, Vienna, 1981, pp. 147–172.

[4] D. S. JOHNSON, A. DEMERS, J. D. ULLMAN, M. R. GAREY AND R. L. GRAHAM, *Worst-case performance bounds for simple one-dimensional packing algorithms*, this Journal, 3 (1974), pp. 299–325.

[5] D. E. KNUTH, *The Art of Computer Programming*, Vol. 1, Addison-Wesley, Reading, MA, 1968, § 2.5.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0003 $01.25/0

WHETHER A SET OF MULTIVALUED DEPENDENCIES IMPLIES A JOIN DEPENDENCY IS NP-HARD*

PATRICK C. FISCHER† AND DON-MIN TSOU‡

Abstract. The problem of determining, given a set of multivalued dependencies, whether or not they logically imply a given join dependency is shown to be computationally intractable.

Key words. relational database, dependency theory, multivalued dependency, join dependency

1. Introduction. Some of the semantics of a relational database [8] can be expressed in terms of different dependencies, of which the best known are *functional* dependencies (FDs) [3], *multivalued* dependencies (MVDs) [6], [9], and *join* dependencies (JDs) [12]. Understanding the relationships among sets of dependencies can therefore aid in the logical design of databases.

One constraint that should be obeyed when a relation r is decomposed into several relations $r_1, \cdots, r_k$ is that the join of the constituent relations should be equal to the original relation r, i.e., no information is lost. The *losslessness* of such a decomposition is equivalent to the existence of a JD, which is expressed as a collection of the sets of attributes in each of the relations, viz., $*[R_1, \cdots, R_k]$. It is well known that an MVD is a special case of a JD with $k = 2$.

We restrict ourselves here to the complexity of determining whether a set of dependencies implies a given dependency. In particular, we settle the principal remaining open question in this area concerning FDs, MVDs, and JDs. Specifically, we will show that it is NP-hard to determine, given a set of MVDs (with or without the inclusion of FDs), whether a given JD is implied, i.e., whether whenever a relational database satisfies the given MVDs it must also satisfy the JD. We summarize the other cases in Table 1 below.

TABLE 1
Complexity of testing implications.

Set of \ Implies a given	FD	MVD	JD
FDs	In P [5]	In P [4], [10]	In P [1]
FDs and MVDs	In P [4]	In P [4], [10], [14]	NP-hard (this paper)
FDs and JDs	In P [13]	In P [13], [16]	NP-hard [13]

2. Preliminaries. We follow, as much as possible, the terminology of [15].

DEFINITION 1. A *relation scheme* V is a set of *attributes* $A_1 \cdots A_p$ taken from a *universe* U. A *relation* r is a collection of *tuples* $\langle d_1, \cdots, d_p \rangle$ where each component d_i contains a value from the *domain* of values for the attribute A_i.

* Received by the editors March 15, 1982, and in revised form April 29, 1982. This research was partially supported by the National Science Foundation under grant MCS-8007706.

† Computer Science Department, Vanderbilt University, Nashville, Tennessee 37235.

‡ Wang Laboratories, Lowell, Massachusetts 01851.

A *functional dependency* $X \to Y$, where X, Y are sets of attributes contained in U, holds for a relation r if for any two tuples t_1, t_2 of r if t_1 and t_2 agree on the components associated with the attributes in X, i.e., on the X-components, then they also agree on the Y-components.

A *join dependency* $*[V_1, \cdots, V_k]$ holds for a relation r if whenever there are tuples $t_1, \cdots, t_k$ of r having the property that for any i, j $(1 \leqq i, j \leqq k)$, t_i agrees with t_j on the components $V_i \cap V_j$, then the tuple t' (which must be well defined by the previous condition) will also be in r, where t' agrees with t_i on the V_i-components for $1 \leqq i \leqq k$.

A *multivalued dependency* $X \twoheadrightarrow Y$ is equivalent to the JD $*[XY, XZ]$ where $Z = U - XY$. (XY is a customary abbreviation for $X \cup Y$.) Hence, it holds in r if whenever two tuples, t_1, t_2, agree on the X-components, a tuple which agrees with t_1 of the XY-components and with t_2 on the XZ-components is also present in r.

In our proof of the main theorem we shall use a reduction from the hitting set problem [11]. We shall also use the tableau characterization of Aho, Ullman et al. [1], [2], [12]. We state the necessary definitions and results next.

DEFINITION 2. Let $S = \{s_1, s_2, \cdots, s_n\}$ be a set and $T = \{T_1, \cdots, T_m\}$ be a family of subsets of S. The *hitting set problem* is to determine, given S and T, whether or not there exists a set $W \subset S$ such that for each i, $1 \leqq i \leqq m$, $T_i \cap W$ contains exactly one element of S. Without loss of generality, we assume

$$\bigcup_{i=1}^{m} T_i = S, \qquad \bigcap_{i=1}^{m} T_i = \varnothing.$$

THEOREM (Karp, [11]). *The hitting set problem is* NP*-complete.*

DEFINITION 3. A *tableau* is a table with a column corresponding to each attribute in the universe U. Entries in a tableau are either a distinguished variable (a) or a nondistinguished variable (b_j for some j). Note that we have dropped the column subscripts from the tableau entries in [1] since they are unnecessary. (Comparisons are never made between variables in different columns and entries never change columns.)

THEOREM (Aho, Beeri, Ullman [1]). *The join of relation schemes $R_1, \cdots, R_k$ is lossless; i.e. $*[R_1, \cdots, R_k]$ holds if and only if a row of distinguished variables (all a's) is derivable in a tableau as follows:*

1) *The initial tableau consists of one row for each given relation scheme. For $1 \leqq j \leqq k$, row j will have an a in each attribute position corresponding to members of R_j and will have b_j in all other positions.*

2) *The tableau is modified by using rules corresponding to the system of given dependencies: (The* FD *rule is not used in this paper because our construction produces a system of* MVD*s only.) The* MVD *rule is: given an* MVD *$X \twoheadrightarrow Y$, whenever two rows of the tableau agree on the positions corresponding to the attributes in X, i.e., on the "X-positions", two new rows can be generated by interchanging all of the entries in the Y-positions. If either of the new rows is identical to one already in the tableau it is not listed a second time.*

Note that the MVD rule for tableaux exactly parallels the MVD characterization in Definition 1. The tableau resulting from all possible applications of the rules is called the *chase* of the initial tableau.

3. The main theorem.

THEOREM. *It is* NP*-hard to determine, given a set of* MVD*s and a* JD*, whether the* JD *is implied by the* MVD*s.*

Proof. We begin with an instance of the hitting set problem as given in Definition 2 and construct the following set of MVDs and a collection of $2n+1$ relation schemes. The universe set of attributes will be

$$U = \{A_0A_1A_2 \cdots A_mB_1B_2 \cdots B_m\} \cup \{C_{ij} | 1 \leqq i \leqq m, 1 \leqq j \leqq n\}.$$

The relation schemes to be joined are

$$Z_0 = U - A_0,$$
$$Z_j = A_0A_1 \cdots A_mC_{1j}C_{2j} \cdots C_{mj} \quad \text{for each } j,\ 1 \leqq j \leqq n,$$
$$Z_{n+j} = A_0X_jY_j \quad \text{for each } j,\ 1 \leqq j \leqq n,$$

where $X_j = \{C_{ij} | s_j \in T_i\}$ and $Y_j = \{B_i | s_j \in T_i\}$.

The MVDs are

(Type I) $A_0A_1 \cdots A_m \twoheadrightarrow A_iB_iC_{i1}C_{i2} \cdots C_{in}$ for each i, $1 \leqq i \leqq m$,

(Type II) $A_0X_j \twoheadrightarrow \{A_iB_iC_{i1}C_{i2} \cdots C_{in} | s_j \in T_i\}$ for each j, $1 \leqq j \leqq n$,

(Type III) $B_1B_2 \cdots B_m \twoheadrightarrow A_0$.

We wish to verify that a row of all a's can be derived from the initial tableau representing $Z_0, Z_1, \cdots, Z_{2n}$ via the rules for the given $m+n+1$ MVDs if and only if there exists a hitting set W for T. From the theorem in [1], the JD $*[Z_0, Z_1, \cdots, Z_{2n}]$ would then be implied by the MVDs if and only if there is a hitting set. Hence the implication problem must be NP-hard.

The proof proceeds through a series of lemmas. First, however, we give an example of a tableau derived from our construction.

Example. Let $n=3$, $m=2$, $T_1 = \{s_1, s_2\}$, $T_2 = \{s_3\}$. There would be 7 rows in the initial tableau:

	A_0	A_1	B_1	C_{11}	C_{12}	C_{13}	A_2	B_2	C_{21}	C_{22}	C_{23}
r_0	b_0	a	a	a	a	a	a	a	a	a	a
r_1	a	a	b_1	a	b_1	b_1	a	b_1	a	b_1	b_1
r_2	a	a	b_2	b_2	a	b_2	a	b_2	b_2	a	b_2
r_3	a	a	b_3	b_3	b_3	a	a	b_3	b_3	b_3	a
r_4	a	b_4	a	a	b_4	b_4	b_4	b_4	b_4	b_4	b_4
r_5	a	b_5	a	b_5	a	b_5	b_5	b_5	b_5	b_5	b_5
r_6	a	b_6	b_6	b_6	b_6	b_6	b_6	a	b_6	b_6	a

The MVDs would be:

(Type I)
$$A_0A_1A_2 \twoheadrightarrow A_1B_1C_{11}C_{12}C_{13}$$
$$A_0A_1A_2 \twoheadrightarrow A_2B_2C_{21}C_{22}C_{23}$$

(Type II)
$$A_0C_{11} \twoheadrightarrow A_1B_1C_{11}C_{12}C_{13}$$
$$A_0C_{12} \twoheadrightarrow A_1B_1C_{11}C_{12}C_{13}$$
$$A_0C_{23} \twoheadrightarrow A_2B_2C_{21}C_{22}C_{23}$$

(Type III)
$$B_1B_2 \twoheadrightarrow A_0.$$

If one applied the rule for the first MVD (Type I, $i=1$) to rows r_1 and r_2 above, the following new rows would be produced:

r_7	a	a	b_2	b_2	a	b_2	a	b_1	a	b_1	b_1
r_8	a	a	b_1	a	b_1	b_1	a	b_2	b_2	a	b_2

Returning to the proof, we will find the following abbreviations useful:

DEFINITION 4.

a) $R_0 = \{r_0, r_1, \cdots, r_{2n}\}$, the initial tableau;

b) $V_i = A_i B_i C_{i1} C_{i2} \cdots C_{in}$ for $1 \leqq i \leqq m$;

c) $H(j) = \{i | s_j \in T_i\}$ for $1 \leqq j \leqq n$.

We shall refer to the various V_i as "*V-groups*". The mnemonic for $H(j)$ is "hits for j". Also, in the lemmas below, we shall refer to "application of an MVD" when we mean the application of an MVD rule in a tableau.

LEMMA 1. *Let tableau R_{12} be generated from R_0 by applying Type* I *and Type* II MVD*s until no new rows are produced. Then in every row of R_{12} except r_0*: 1) *there is an a in the A_0 position*; 2) *within each V-group there are at most two a entries.*

Proof. By induction on the number of applications of MVDs. These properties clearly hold for $r_1, r_2, \cdots, r_{2n}$. Now assume the two properties hold for an intermediate tableau and consider the application of a Type I or Type II MVD. Since r_0 differs in the A_0-position from all other rows so far generated, it cannot participate in the application. Therefore both rows to which the MVD is applied have the given properties. Observe that both Type I and Type II MVDs have the effect of moving entire V-groups, never any proper subsets of a V-group, nor the A_0 entry. Thus both new rows are obtained by interchanging certain V-groups and hence also have both properties.

LEMMA 2. *A row with all a's can be derived, i.e. will be in the chase of R_0, if and only if the Type* III MVD *is applied to r_0 and to a row in R_{12} (see Lemma* 1) *having a's in all of the $B_1 \cdots B_m$-positions.*

Proof. It is obvious that an application of the Type III MVD as specified will yield a row of all a's. Conversely, suppose all a's can be derived. From Lemma 1, R_{12} contains no such row. Therefore the Type III MVD must be used. If neither of the two rows to which it is applied is r_0 then no new rows will be created since A_0 is a in both rows. Therefore one row must be r_0. Since r_0 has a in each of the $B_1 \cdots B_m$-positions so must the other row.

LEMMA 3. *A row with all a's will be in the chase of R_0 if and only if a row with a's in the $B_1 \cdots B_m$ positions is present in the tableau R' generated as follows*:

a) *A_0 is removed from the universe of attributes.*

b) *The initial tableau R'_0 consists of the rows $r_1, \cdots, r_{2n}$ given above but with the A_0-column deleted.*

c) *The* MVD*s are the Type* I *and Type* II MVD*s previously given with A_0 deleted. Restated, in terms of Definition* 4, *they are*:
Type I(i) $A_1 \cdots A_m \twoheadrightarrow V_i$ *for each i*, $1 \leqq i \leqq m$;
Type II(j) $X_j \twoheadrightarrow \bigcup_{i \in H(j)} V_i$ *for each j*, $1 \leqq j \leqq n$.

d) *The* MVD*s are applied until no new rows are produced (i.e. R' is the chase of R'_0 with respect to the* MVD*s given above).*

Proof. Immediate from Lemmas 1 and 2.

Henceforth R'_0 and R' will have the meanings given in Lemma 3.

DEFINITION 5. We define the following strings of length $n+2$ for each j, $1 \leqq j \leqq n$:

$$E_j = ab_j(b_j)^{j-1}a(b_j)^{n-j},$$

$$F_j = b_{n+j}a(b_{n+j})^{j-1}a(b_{n+j})^{n-j},$$

$$G_j = (b_{n+j})^{n+2}.$$

We will call these E-, F- and G-strings, respectively. In the example, the E-, F- and

G-strings would be

$$E_1 = a\ b_1a\ b_1b_1 \quad F_1 = b_4a\ a\ b_4b_4 \quad G_1 = b_4b_4b_4b_4b_4$$
$$E_2 = a\ b_2b_2a\ b_2 \quad F_2 = b_5a\ b_5a\ b_5 \quad G_2 = b_5b_5b_5b_5b_5$$
$$E_3 = a\ b_3b_3b_3a \quad F_3 = b_6a\ b_6b_6a \quad G_3 = b_6b_6b_6b_6b_6.$$

LEMMA 4. *In every row of the tableau R', each V-group has for its entries either E_j for some j, F_j for some j, or G_j for some j.*

Proof. By induction on the number of applications of MVDs. For the basis we restate the rows of R_0' according to the terminology of Definition 5.

For $1 \leqq j \leqq n$, r_j has E_j in each of the V-groups.

For $1 \leqq j \leqq n$, r_{n+j} has $\begin{cases} F_j \text{ in each group } V_i \text{ where } i \in H(j), \\ G_j, \text{ otherwise.} \end{cases}$

The inductive step is the same as in Lemma 1.

LEMMA 5. *A row of all a's will be in the chase of R_0 if and only if R' contains a row of the form*

$$F_{j_1}F_{j_2} \cdots F_{j_m} \quad \textit{for some choice of } j_1, j_2, \cdots, j_m.$$

Proof. Immediate from Definition 5 and Lemmas 3 and 4, since for each i, $1 \leqq i \leqq m$, V_i will have an a in the B_i-position if and only if V_i contains an F-string.

LEMMA 6. *Let R_1' consist of all n^m rows of the form $E_{j_1}E_{j_2} \cdots E_{j_m}$ for all possible choices in $1 \leqq j_i \leqq n$, $1 \leqq i \leqq m$. Then $R_1' \subset R'$.*

Proof. Given $j_1, j_2, \cdots, j_m$, we begin with rows r_{j_1} and r_{j_2} and apply MVD I(j_2), producing a row $r_2' = E_{j_1}E_{j_2}E_{j_1} \cdots E_{j_1}$. Next we apply MVD I($j_3$) to r_2' and r_{j_3}, producing $r_3' = E_{j_1}E_{j_2}E_{j_3}E_{j_1} \cdots E_{j_1}$. Similarly, we apply MVD I(j_4) through MVD I(j_m) in turn, finally producing r_m', the desired row.

To prove our main theorem it suffices by Lemma 5 to show that a row with an F-string in each V-group exists in R' if and only if T has a hitting set. We are now ready to show that the existence of a hitting set will cause R' to have such a row. The converse will require further analysis of R'.

LEMMA 7. *If T contains a hitting set, then R' has a row of the form*

$$F_{j_1}F_{j_2} \cdots F_{j_m},$$

i.e., each V-group contains an F-string.

Proof. Let $W = \{s_{w_1}, s_{w_2}, \cdots, s_{w_p}\}$ be a hitting set for T. Then for any $k \neq k'$, $H(w_k) \cap H(w_{k'}) = \varnothing$ since no T_i contains two distinct members of W. From Lemma 6, R' contains $r_0^* = E_{j_1}E_{j_2} \cdots E_{j_m}$, where for each i, $1 \leqq i \leqq m$, $s_{j_i} \in T_i \cap W$, i.e., E_{w_k} appears in each group V_i where $i \in H(w_k)$. We apply MVD II(w_1) to r_0^* and r_{n+w_1}, producing the row r_1^* which has F_{w_1} in each group V_i where $i \in H(w_1)$. We now apply MVD II(w_2) to r_1^* and r_{n+w_2} producing r_2^* which has F_{w_2} in each group V_i where $i \in H(w_1)$. Since $H(w_2)$ and $H(w_1)$ are disjoint, r_2^* retains F_{w_1} in each group V_i where $i \in H(w_1)$. Similarly, we apply MVD II(w_3) through MVD II(w_p), in turn. The final row r_p^* will have F_{w_k} in each group V_i where $i \in H(w_k)$ for $1 \leqq k \leqq p$. Since W is a hitting set, $\bigcup_{1 \leqq k \leqq p} H(w_k) = \{1, 2, \cdots, m\}$ so r_p^* has an F-string in each group V_i, $1 \leqq i \leqq m$.

DEFINITION 6.

a) An *EF-row* is a row in which:
   i) Each V-group contains an E-string or an F-string.
   ii) For each j, $1 \leqq j \leqq n$, if F_j appears in any of the V-groups then F_j is in group V_i if and only if $i \in H(j)$.

b) A *G-row* is a row which, for some j, $1 \leqq j \leqq n$, satisfies:

i) For the groups V_i where $i \in H(j)$ either all contain E_j or all contain F_j.

ii) All other V-groups contain G_j.

There are only $2n$ G-rows. Let us call them, for $1 \leqq j \leqq n$, GE_j or GF_j depending upon the choice of E_j or F_j in b)i) above.

LEMMA 8. *The tableau R' consists only of EF-rows and G rows.*

Proof. By induction on the number of applications of MVDs. In the initial tableau R'_0, rows r_1 through r_n consist only of E-strings and thus satisfy the second condition of Definition 6 a) ii) vacuously. Rows r_{n+1} through r_{2n} are clearly the G rows $GF_1, \cdots, GF_n$ (cf. Lemma 4).

The induction step proceeds by cases:

Case I. An MVD is applied to two EF-rows r and r'. If the MVD is Type I, then if either row has an F-string in any group V_i, the other row must have the same F-string in V_i. This is so because r and r' must match in the A_i-position, and F_j has b_j in this position, whereas any E-string has an a there. Thus the MVD would either interchange two identical F-strings, creating nothing new, or interchange two E-strings, creating two more EF-rows.

If the MVD is Type II, let us assume that it is II(j) for some j. Then r and r' match on $X_j = \{C_{ij} | i \in H(j)\}$. For r and r' to agree in the C_{ij} position for some $i \in H(j)$, either group V_i contains E_j in one case and F_j in the other or the two rows are identical on the V_i-positions. From Definition 6 a) ii) and the induction hypothesis there are only three possible subcases:

a) Both r and r' contain F_j in the V_i-positions for $i \in H(j)$. Then MVD II(j) will create no new rows since $\bigcup_{i \in H(j)} V_i$ is the portion interchanged.

b) One row contains F_j in V_i for $i \in H(j)$, the other contains E_j in these V-groups. Then the resulting rows will be EF-rows.

c) Both r and r' contain E-strings in all V_i where $i \in H(j)$. Then no new rows are obtained since the rows agree on these V_i.

Case II. An MVD is applied to two G-rows r and r'. First assume the MVD is Type I. Suppose $r = GE_j$ for some j. Then, for some $i \notin H(j)$, V_i contains G_j, which has b_{n+j} in the A_i-position. Then r' must also have G_j in group V_i, so r' is either GE_j or GF_j. But E_j and F_j disagree in the A_i position. Therefore $r' = r$. If $r = GF_j$, the same reasoning will show $r' = r$.

If the MVD is Type II(j) for some j, then either r and r' agree on all of the V_i-positions for $i \in H(j)$ or one of the rows is GE_j and the other is GF_j. In either case, no new rows are produced.

Case III. An MVD is applied to an EF-row r, and a G-row r'. The MVD cannot be Type I. To see this, assume r' is GE_j or GF_j. Then for some $i \notin H(j)$ group V_i of r' contains G_j, which matches no E-string or F-string of r in the A_i-position. Therefore, assume the MVD is Type II(j) for some j. Then $r' = GE_j$ or $r' = GF_j$ since all other G-rows would fail to match r on X_j. Thus r must have a's in the X_j positions. From Definition 6 a) ii) this means either r has E_j in each V_i where $i \in H(j)$, or r has F_j in each of these V-groups. The new rows will clearly be an EF-row and a G-row.

LEMMA 9. *If a row in which each V-group contains an F-string exists in R', then T has a hitting set W.*

Proof. Take any row r_F in which each V-group has an F-string. Let $\{w_1, w_2, \cdots, w_p\}$ be the set of indices of the F-strings in r_F, i.e.

$$r_F = F_{j_1} F_{j_2} \cdots F_{j_m}$$

and for each i, $1 \leqq i \leqq m$, $j_i \in \{w_1, w_2, \cdots, w_p\}$. From Lemma 8, for $1 \leqq i \leqq m$, group

V_i has F_{w_k} as its entries if and only if $i \in H(w_k)$. Since $F_j \neq F_{j'}$ whenever $j \neq j'$, it follows that $H(w_k) \cap H(w_{k'}) = \varnothing$ if $k \neq k'$. Furthermore, for $1 \leqq i \leqq m$, $i \in H(w_k)$ for some k since r_F has only F-strings.

From the definition of $H(j)$ we see that $W = \{s_{w_1}, s_{w_2}, \cdots, s_{w_p}\}$ is a hitting set for T.

The main theorem now follows from Lemmas 5, 7 and 9.

4. Conclusion. The main theorem plus the results in [13] show that the complexity of the inference problem (whether a set of dependencies D logically implies a single dependency d) depends on d more than D when all dependencies are MVDs or JDs. The order (number of components) of the JD produced by our construction, however, can be arbitrarily large. Since an MVD is a JD of order 2, this raises the question for other JDs of bounded order. In particular, what is the complexity of the inference problem when D contains arbitrary JDs and d is an order-3 JD or a JD of any bounded order? Does this complexity change if restrictions are also placed on the order of the JDs in D?

Another open question is the membership of the problem under discussion (D is MVDs, d is a JD) in the class NP. We have not been able to settle this. In [13] it was shown that when D is a set of FDs, no MVDs and a single JD and d is a JD, then the implication problem is in NP. However, the authors did not show membership in NP when D has MVDs and a JD; if this were so our case would obviously also be in NP.

In [17], it is shown that if D is one FD and one JD, and d is a JD, then the inference problem is NP-complete. Can an analogous result be obtained for MVDs?

REFERENCES

[1] A. V. AHO, C. BEERI AND J. D. ULLMAN, *The theory of joins in relational databases*, ACM Trans. Database Systems, 4 (1979), pp. 297–314.

[2] A. V. AHO, Y. SAGIV AND J. D. ULLMAN, *Efficient optimization of a class of relational expressions*, ACM Trans. Database Systems, 4 (1979), pp. 435–454.

[3] W. W. ARMSTRONG, *Dependency structures of data base relationships*, in Information Processing 74, North-Holland, Amsterdam, 1974, pp. 580–583.

[4] C. BEERI, *On the membership problem for function and multivalued dependencies in relational databases*, ACM Trans. Database Systems, 5 (1980), pp. 241–259.

[5] C. BEERI AND P. A. BERNSTEIN, *Computational problems related to the design of normal form relational schemes*, ACM Trans. Database Systems, 4 (1979), pp. 30–59.

[6] C. BEERI, R. FAGIN AND J. HOWARD, *A complete axiomatization for functional and multivalued dependencies*, Proc. ACM SIGMOD Conference, 1977, pp. 47–61.

[7] C. BEERI, R. FAGIN, D. MAIER, A. MENDELZON, J. ULLMAN AND M. YANNAKAKIS, *Properties of acyclic database schemes*, Proc. Thirteenth Annual ACM Symposium on Theory of Computing, 1981, pp. 355–362.

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

[9] R. FAGIN, *Multivalued dependencies and a new normal form for relational databases*, ACM Trans. Database Systems, 2 (1977), pp. 262–278.

[10] K. HAGIHARA, M. ITO AND K. TANIGUCHI, *Decision problem for multivalued dependencies in relational databases*, this Journal, 8 (1979), pp. 247–264.

[11] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Plenum Press, New York, 1972, pp. 85–104.

[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, Y. SAGIV AND M. YANAKAKIS, *On the complexity of testing implications of functional and join dependencies*, J. Assoc. Comput. Mach., 28 (1981), pp. 680–695.

[14] D.-M. TSOU, *Analysis of the logical design in relational databases*, Technical Rep. CS-80-11, Vanderbilt University, Nashville, TN, 1980.
[15] J. D. ULLMAN, *Principles of Database Systems*, Computer Science Press, Potomac, MD, 1979.
[16] M. Y. VARDI, *Inferring multivalued dependencies from functional and join dependencies*, Technical Rep., Weizmann Institute of Science, Rehovot, Israel, 1980.
[17] C. BEERI AND M. Y. VARDI, *On the complexity of testing implications of data dependencies*, Technical Rep., Hebrew University, Jerusalem, Israel, 1980.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0004 $01.25/0

THE EQUIVALENCE OF TWO SEMANTIC DEFINITIONS: A CASE STUDY IN LCF*

AVRA COHN†

Abstract. We present a case study in LCF of the equivalence of two semantic definitions for a small language. The language contains recursive and nonrecursive procedure declarations, and static binding of variables is intended. A standard semantics is proved equivalent to a closure semantics in which procedures denote closures (textual objects). This proof is discussed abstractly in [1]. A similar equivalence is proved by J. Stoy in [6, Chap. 13], based on work by R. Milne and C. Strachey [4]. We describe the formalization of the problem in LCF, the informal proof by structural induction over programs of the language, and the strategy for performing the proof mechanically. The strategy is quite general even though it involves rather subtle handling of lemmas and induction hypotheses. A basic understanding of Scott's theory of domains is assumed, including the least fixed point operator and domain constructing operations such as domain equations.

Key words. compiler correctness, semantics, theorem proving, verification

The LCF system. The LCF system [3] is designed to support interactive, goal-oriented performance of formal proofs in a particular logic (called PPLAMBDA), by the application of proof strategies implemented in a general purpose programming language (ML). The aim in each proof effort is to set out a goal to be achieved, and to design a tactic which implements a strategy for achieving or partially achieving the goal. A goal is a triple, consisting of the PPLAMBDA formula to be proved, a set (*simpset*) of representations of theorems expressing term equivalences (*simprules*) to be used as left-to-right rewrite rules, and a list of assumptions current in a proof effort. A tactic is an ML procedure which produces a list of subgoals from a goal, and produces a justification of the step from goal to subgoals. A justification is a function which maps theorems achieving the subgoals to a theorem achieving the original goal. A theorem with a list of hypotheses A' and a conclusion w' achieves a goal (w, ss, A) if the formulae w and w' are the same (up to renaming of bound variables) and if all formulae in A' either belong to A or are hypotheses of one of the theorems represented by some element of ss.

A tactic may simply be the inverse of a PPLAMBDA rule of inference, or it may encode a more complex pattern of reasoning. Tactics are combined into larger tactics by means of ML functions called tacticals, the most commonly used of which are THEN for sequencing of tactics, THENL for selective sequencing, and REPEAT for repeated application. For tactics $T1$ and $T2$, the tactic $T1$ THEN $T2$, when applied to a goal, applies $T1$ to the goal, and applies $T2$ to the ensuing subgoals. For tactics T and list of tactics $[T1, \ldots, Tn]$, T THENL $[T1, \ldots, Tn]$, applied to a goal, applies T to the goal to obtain n subgoals gi, and then applies Ti to gi, for each i. When inapplicable to a goal, a tactic raises an exception, using the ML exception-handling mechanism. For a tactic T, the tactic REPEAT T, applied to a goal, applies T to obtain either no, one or several subgoals, applies T to each, and continues, until (and if) an exception is generated. A tactic may produce no subgoals, in which case the goal has been achieved, and the justification can be evaluated (that is, the proof can be performed).

A set of standard tactics is provided in LCF, and others are written by the user, as needed. Both the standard and the user-defined tactics are implemented as

* Received by the editors February 17, 1981, and in revised form May 26, 1982.

† Computer Laboratory, University of Cambridge, Corn Exchange Street, Cambridge CB2 3QG, England. This research was done while the author was at the University of Edinburgh.

procedures in ML. Tactics are combined to express complex strategies to solve various goals; the combined justifications produce theorems achieving the goals. One of the research aims of work in LCF is to find and implement general strategies for proving classes of theorems.

The standard tactic SIMPTAC, applied to a goal, applies all of the simprules in the simpset to the formula part of the goal as many times as possible, to rewrite the formula. As all tactics do, it also provides a justification for the step from goal to subgoal. In addition, SIMPTAC can recognize some tautologies, and may thus produce from a goal an empty list of subgoals.

More precisely, each simprule is derived from a theorem (since each rewriting must be justified) of the form

$$A \vdash \{\forall x1 \ldots xn.\}\{w \supset\} t1 = t2$$

where theorems are written with a list of assumptions to the left of the turnstile and the conclusion to the right, $t1 = t2$ is a term equivalence in PPLAMBDA, w is a formula in PPLAMBDA, and curly brackets indicate optional components. An application of the simprule to some formula consists in a match of subterms of the formula to $t1$; if a match is possible, then the corresponding instance of $t2$ replaces the instance of $t1$ in the formula. A match cannot involve instantiation of variables free in the list, A, of assumptions of the theorem. If an antecedent is present, the corresponding instance of the antecedent must be reduced to a tautology by simplification before the replacement can be made; otherwise it is not made. This is called *conditional simplification*. If quantifiers are present, the theorem is specialized to arbitrary variables, which are then instantiable in matches.

SIMPTAC uses all such simprules provided as part of a goal, and some standard ones in addition. The object, in any proof effort, is to relegate as much of the work as possible to simplification. This methodology is based on the belief, or at least hope, that most of the steps in most proofs are routine, and the user should only have to guide the search over the difficult steps. At the same time, the methodology demands some care in the choice of simprules (as we shall see).

The formalization of some problems requires the introduction of new data types, objects, and axioms beyond those already in PPLAMBDA. To allow this, LCF includes a facility for building (hierarchies of) logical theories. Theorems already proved can be saved in theories for later use.

In this paper we examine the hierarchy of theories needed to formulate the syntax of a small language and its two semantic definitions, the goal expressing the equivalence of the semantics, and the tactics which achieve the goal. First, we consider the problem informally.

The problem, informally. In our toy language, we allow blocks in which local variables are declared, and procedures invoked. For simplicity, we consider blocks with exactly one declaration apiece, and procedures without parameters. We allow identifiers to denote only procedures.

I ranges over a domain ID of identifiers; p, $p1$ and $p2$ over a domain P of programs; and the variable a over a domain ATOM of (unspecified) atomic programs. Programs are given by

$$\begin{aligned} p ::= \; & a \mid \\ & \textbf{call } I \mid \\ & \textbf{let } I = p1 \textbf{ in } p2 \mid \\ & \textbf{letrec } I = p1 \textbf{ in } p2 \mid \\ & p1; p2. \end{aligned}$$

The second construct specifies procedure invocation, the third and fourth, respectively, blocks with nonrecursive and recursive procedure declaration, and the fifth, sequencing.

A standard (direct) semantics—in which procedures denote functions—is given for this language. It requires a domain, ENV, of environments in which identifiers are mapped to the meanings of programs. The meanings of programs are transformations on stores; the structure of the domain STORE is not important for our purposes. We let σ range over STORE. Thus we have

$$\rho \in \text{ENV} = \text{ID} \to \text{STORE} \to \text{STORE}.$$

We define semantic functions $\mathscr{A}$ for atomic programs, and $\mathscr{S}$ for programs:

$$\mathscr{A}\colon \text{ATOM} \to \text{STORE} \to \text{STORE},$$

$$\mathscr{S}\colon P \to \text{ENV} \to \text{STORE} \to \text{STORE}.$$

The clauses for $\mathscr{S}$ are

$$\begin{aligned}
&\mathscr{S}[\![a]\!]\rho && = \mathscr{A}[\![a]\!],\\
&\mathscr{S}[\![\textbf{call } I]\!]\rho && = \rho\, I,\\
&\mathscr{S}[\![\textbf{let } I = p1 \textbf{ in } p2]\!]\rho && = \mathscr{S}[\![p2]\!](\rho[\mathscr{S}[\![p1]\!]\rho/I]),\\
&\mathscr{S}[\![\textbf{letrec } I = p1 \textbf{ in } p2]\!]\rho && = \mathscr{S}[\![p2]\!](\text{FIX}(\lambda\rho' \,.\, \rho[\mathscr{S}[\![p1]\!]\rho'/I])),\\
&\mathscr{S}[\![p1; p2]\!]\rho && = \lambda\sigma \,.\, \mathscr{S}[\![p2]\!]\rho\,(\mathscr{S}[\![p1]\!]\rho\sigma).
\end{aligned}$$

($f[x/y]$ denotes the function $\lambda z.$ if $z = y$ then x else $f z$.) We assume that for all a, $\mathscr{A}[\![a]\!]\perp = \perp$. The only subtle clause is the one for recursive procedure declaration, in which I is recursively declared to denote $p1$ throughout $p2$. In that case, we take the meaning of the body $p2$ in an environment, $\hat{\rho}$, say, which is like ρ except on I; I is bound to the meaning of $p1$ in $\hat{\rho}$:

$$\hat{\rho} = \rho[\mathscr{S}[\![p1]\!]\hat{\rho}/I] = \text{FIX}\,(\lambda\rho' \,.\, \rho[\mathscr{S}[\![p1]\!]\rho'/I]).$$

We now define a closure semantics, $\bar{\mathscr{S}}$, for the language, which we will eventually prove equivalent to $\mathscr{S}$. The environments in $\bar{\mathscr{S}}$ map identifiers to closures, which are pairs consisting of programs and (declaration time) environments. Closures are representations of the meanings of procedures in $\mathscr{S}$, that is, representations of store transformations. The semantics can be viewed as defining an abstract interpreter for the language.

We let v range over CENV, a reflexive domain of closure environments:

$$v \in \text{CENV} = \text{ID} \to (P \times \text{CENV}).$$

The semantic function $\bar{\mathscr{S}}$ has type

$$\bar{\mathscr{S}}\colon P \to \text{CENV} \to \text{STORE} \to \text{STORE}.$$

Its clauses are:

$$\begin{aligned}
&\bar{\mathscr{S}}[\![a]\!]v && = \mathscr{A}[\![a]\!],\\
&\bar{\mathscr{S}}[\![\textbf{call } I]\!]v && = \bar{\mathscr{S}}[\![p']\!]v' \text{ where } (p', v') = v\, I,\\
&\bar{\mathscr{S}}[\![\textbf{let } I = p1 \textbf{ in } p2]\!]v && = \bar{\mathscr{S}}[\![p2]\!](v[(p1, v)/I]),\\
&\bar{\mathscr{S}}[\![\textbf{letrec } I = p1 \textbf{ in } p2]\!]v && = \bar{\mathscr{S}}[\![p2]\!](\text{FIX}\,(\lambda v'.v[(p1, v')/I])),\\
&\bar{\mathscr{S}}[\![p1; p2]\!]v && = \lambda\sigma \,.\, \bar{\mathscr{S}}[\![p2]\!]v(\bar{\mathscr{S}}[\![p1]\!]v\sigma).
\end{aligned}$$

In stating the equivalence of $\mathcal{S}$ and $\bar{\mathcal{S}}$, we first state the simulation relation (or congruence condition) between the respective "contexts" of the semantic functions, that is, between ρ of type ENV = ID → STORE → STORE and v of type CENV = ID → $P \times$ CENV. The obvious relation is

$$\forall I \,.\, \rho\, I = \bar{\mathcal{S}}[\![p']\!]v' \quad \text{where } (p', v') = v\, I.$$

That is,

$$\forall I \,.\, \rho\, I = \bar{\mathcal{S}}[\![\textbf{call } I]\!]v.$$

We abbreviate this relation between ρ and v as $\rho \sim v$.

The remainder of this section is devoted to the informal proofs, which we shall later formalize and mechanize.

Our goal is to prove

THEOREM.

$$\forall p \rho v \,.\, \rho \sim v \supset \mathcal{S}[\![p]\!]\rho = \bar{\mathcal{S}}[\![p]\!]v.$$

The proof is facilitated by a lemma.

LEMMA.

$$\forall \rho v p' v' J \,.\, \rho \sim v \supset \rho[\bar{\mathcal{S}}[\![p']\!]v'/J] \sim v[(p', v')/J].$$

Proof of lemma. We show

$$(\forall I \,.\, \rho\, I = \bar{\mathcal{S}}[\![\textbf{call } I]\!]v) \supset (\forall I \,.\, \rho[\bar{\mathcal{S}}[\![p']\!]v'/J]I = \bar{\mathcal{S}}[\![\textbf{call } I]\!](v[(p', v')/J])).$$

We assume that $(\forall I \,.\, \rho\, I = \bar{\mathcal{S}}[\![\textbf{call } I]\!]v)$.

Case $I \neq J$. We must show that

$$\rho\, I = \bar{\mathcal{S}}[\![\textbf{call } I]\!]v.$$

This follows from the assumption.

Case $I = J$. Now

$$\begin{aligned} \rho[\bar{\mathcal{S}}[\![p']\!]v'/J]I &= \bar{\mathcal{S}}[\![p']\!]v' \\ &= \mathcal{S}[\![\textbf{call } I]\!](v[(p', v')/J]) \end{aligned}$$

by definition of $\bar{\mathcal{S}}$. Q.E.D.

The proof of the theorem is by structural induction on programs. We use corner brackets to map concrete to abstract syntax.

Proof of theorem.

Basis. If $p = \perp$ or $p = \ulcorner a \urcorner$, we assume that $\rho \sim v$, and the proof is obvious.

Case $p = \ulcorner \textbf{call } I \urcorner$. We assume $\rho \sim v$.

$$\mathcal{S}[\![\textbf{call } I]\!]\rho = \rho\, I = \bar{\mathcal{S}}[\![\textbf{call } I]\!]v$$

by the definition of $\mathcal{S}$ and the assumption.

Step. We assume the theorem for $p1$ and $p2$ in place of p, and we assume $\rho \sim v$.

Case $p = \ulcorner \textbf{let } I = p1 \textbf{ in } p2 \urcorner$. We must show

$$\mathcal{S}[\![\textbf{let } I = p1 \textbf{ in } p2]\!]\rho = \bar{\mathcal{S}}[\![\textbf{let } I = p1 \textbf{ in } p2]\!]v,$$

that is,

$$\mathcal{S}[\![p2]\!](\rho[\mathcal{S}[\![p1]\!]\rho/I]) = \bar{\mathcal{S}}[\![p2]\!](v[(p1, v)/I]).$$

In order to use the induction hypothesis for $p2$, we must show

$$\rho[\mathcal{S}[\![p1]\!]\rho/I] \sim v[(p1, v)/I].$$

This follows by a use of the induction hypothesis for $p1$ on the subterm $\mathscr{S}[\![p1]\!]\rho$, with the assumption that $\rho \sim v$, and then by an application of the lemma, also with the assumption that $\rho \sim v$.

Case $p = \ulcorner$**letrec** $I = p1$ **in** $p2\urcorner$. We must show

$$\mathscr{S}[\![p2]\!](\mathrm{FIX}\,(\lambda\rho' \,.\, \rho[\mathscr{S}[\![p1]\!]\rho'/I])) = \mathscr{S}[\![p2]\!](\mathrm{FIX}\,(\lambda v' \,.\, v[(p1, v')/I])).$$

We call the two functionals $\mathscr{R}$ and $\mathscr{V}$ respectively, and prove

$$\mathscr{S}[\![p2]\!](\mathrm{FIX}\,\mathscr{R}) = \bar{\mathscr{S}}[\![p2]\!](\mathrm{FIX}\,\mathscr{V}).$$

This requires an inner computation induction in order to prove that FIX $\mathscr{R} \sim$ FIX $\mathscr{V}$, and so complete the proof, by a use of the induction hypothesis for $p2$.

The basis case is trivial.

Assume. $\bar{\rho} \sim \bar{v}$, for arbitrary $\bar{\rho}$ and $\bar{v}$.

Show. $\mathscr{V}\,\bar{v} \sim \mathscr{R}\,\bar{\rho}$, that is,

$$\rho[\mathscr{S}[\![p1]\!]\bar{\rho}/I] \sim v[(p1, \bar{v})/I].$$

By the outer induction hypothesis for $p1$, and the inner induction hypothesis, we have

$$\mathscr{S}[\![p1]\!]\bar{\rho} = \bar{\mathscr{S}}[\![p1]\!]\bar{v}.$$

Hence, by applying the lemma, with the assumption that $\bar{\rho} \sim \bar{v}$, the result follows.

Case $p = \ulcorner p1; p2\urcorner$. We assume that $\rho \sim v$.

$$\begin{aligned}\mathscr{S}[\![p1; p2]\!]\rho &= \lambda\sigma \,.\, \mathscr{S}[\![p2]\!]\rho(\mathscr{S}[\![p1]\!]\rho\sigma)\\ &= \lambda\sigma \,.\, \bar{\mathscr{S}}[\![p2]\!]v(\bar{\mathscr{S}}[\![p1]\!]v\sigma)\\ &= \bar{\mathscr{S}}[\![p1; p2]\!]v\end{aligned}$$

by a use of each induction hypothesis, with the assumption, in turn. Q.E.D.

The proof in LCF. To perform this proof in LCF, a certain amount of effort is invested in (1) formalizing the problem, and (2) formulating a goal and designing a strategy to solve it.

The proof, in all, consists of about one thousand primitive PPLAMBDA inferences, but it is generated by eight conceptually coherent tactics, three of which are standard, and five of which are designed for this (and other) proof(s). Less subjectively, in the proof of the lemma, simplification solves all of the cases in the case analysis. In the proof of the main theorem, simplification solves the undefined case of both the inner and outer inductions, as well as the **call** case. The role of simplification will be made clearer on closer examination.

Notation. The logic in which the problem is expressed, PPLAMBDA, is a predicate calculus whose terms are from the typed lambda calculus. PPLAMBDA expressions (terms and formulae) are written in quotation marks thusly: "...". PPLAMBDA constants include the selectors "FST" and "SND", where "FST (x, y), SND (x, y)" is "(x, y)". Basic formulae are "$t1 == t2$" and "$t1 \ll t2$", denoting equivalence or inequivalence of PPLAMBDA terms "$t1$" and "$t2$" in the domains to which the types of the terms $t1$ and $t2$ correspond. A PPLAMBDA term can be a conditional "**if** t **then** $t1$ **else** $t2$" for truth-valued "t", an application "$t1\ t2$" or a lambda abstraction "$\backslash x \,.\, t$" (which is read "$\lambda x \,.\, t$"). An implication with an antecedent formula "$t1 == t2$", say, and consequent "$t2 == t1$", is written "$t1 == t2$ IMP $t2 == t1$". The symbol for the quantifier $\forall$ in PPLAMBDA is "!". PPLAMBDA types are written in quotation marks preceded by a colon; for example,

":tr" is the constant PPLAMBDA type for truth values. There are three PPLAMBDA constants with this type, "TT", "FF" and "UU" for true, false and undefined, respectively. Each PPLAMBDA type corresponds to a domain (cpo). PPLAMBDA type operators include the binary operators sum (+), product (×), and function space formation (→), and the unary operator for lifting (adding a new minimum element to) domains (u).

To help the reader, we adopt the following notation. Inference rules of PPLAMBDA are written with a single horizontal line, with the argument(s) of the rule above, and the theorem returned by the rule below. Goals are written sometimes as triples, and sometimes as boxes with three horizontal components, containing, from top to bottom, the formula to be achieved, the theorems contributing to the simpset, and the list of current assumptions. Tactics are suggested by a double horizontal line, above which is a goal and below which is a list of subgoals. (The justification part of the tactic is usually obvious.) For succinctness, compound tactics are written in horizontal columns or trees of columns, with the tactical THEN suggested by direct adjacency in the column, and THENL by branching. The tactical REPEAT is denoted by a * following the tactic to be repeated, and (T THEN SIMPTAC) for a tactic T is written T^+. (This informal notation could obviously be made rigorous without difficulty.)

To illustrate, the standard inference rule GEN, which is parameterized on a term x (so that we will sometimes call the rule GEN x) is written

$$\frac{A \vdash w}{A \vdash \forall x \,.\, w} \qquad \text{(where } x \text{ is not free in } A\text{)}.$$

GEN generalizes a theorem to a given variable, if possible. The standard tactic GENTAC, inverting the rule, is written

$\forall x \,.\, w$
ss
A

═══

$w[x'/x]$
ss
A

(where x' is a variant of x not free in A)

where ss is a simpset and A is a list of assumptions, and $w[x'/x]$ means w with all free occurrences of x replaced by x'. The justification of GENTAC is obviously in terms of GEN. Again, the standard tactic CONDCASESTAC is written

(w, ss, A) where w contains a subterm "**if** t **then** $t1$ **else** $t2$"

═══

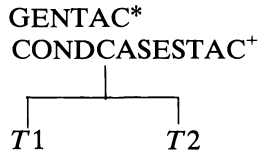
w
"$t == TT$" ⊢ "$t == TT$" ss
"$t == TT$" A

w
"$t == FF$" ⊢ "$t == FF$" ss
"$t == FF$" A

w
"$t == UU$" ⊢ "$t == UU$" ss
"$t == UU$" A

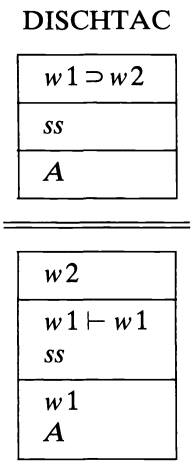
where w is some formula with a subterm which contains a conditional expression, as indicated. The tactic finds such a term (failing if there is none). The justification is in terms of a standard cases rule of inference. Note that each subgoal has one assumption added to the list of assumptions, and one simprule added to the simpset. The additions to the simpsets are constructed by the standard inference rule ASSUME, which maps a formula w to the theorem $w \vdash w$.

A tactic such as (REPEAT GENTAC THEN CONDCASESTAC THEN SIMPTAC THENL [$T1$, $T2$]) for some tactics $T1$ and $T2$, would be abbreviated to



(We might write a tactic of this sort if we expected simplification to manage to solve the undefined-case subgoal, and $T1$ and $T2$, respectively, to advance the proofs of the other two subgoals.) Tactics, T, which are parameterized on a theorem or other object, t, are written applicatively as $T\,t$.

Tactics for the proof. Of those parts of the proof which are not managed by simplification, some steps are accomplished by the standard tactics GENTAC and CONDCASESTAC, described earlier. The rest of the proof consists in steps which require specially designed tactics. A consideration of the informal proof suggests some useful general strategies; for example, in proving an implication, we often assume the antecedent and set out to prove the consequent. As the justification of this step is by the discharging the extra assumption on which the theorem achieving the subgoal rests, we write



Note that this version of DISCHTAC uses the theorem $w1 \vdash w1$ as a simprule, as well as an assumption; this is useful here, but not always desirable (or even possible).

The proof of the main theorem is by structural induction on programs. As the standard rule of induction in LCF is computation induction, any other induction rule, including the rule of structural induction for any well-founded structure, must be derived from computation induction as a "hidden" induction on some other appropriately defined function. This derivation is standard for many structures, and indeed has been mechanized as an ML procedure by R. Milner [2, App.] as part of a general package for constructing and working in theories of certain kinds of structures. We

do not elaborate further here on this derivation, except to specify the induction tactic that we need in our proof, which we shall call PINDUCTAC, for program induction.

PINDUCTAC
(w, ss, A)

$w[\bot/p]$
ss
A

$w[\ulcorner a \urcorner/p]$
ss
A

$w[\ulcorner \textbf{call } I \urcorner/p]$
ss
A

$w[\ulcorner \textbf{let } I = p1 \textbf{ in } p2 \urcorner/p]$
ss
IH1 IH2 A

$w[\ulcorner \textbf{letrec } I = p1 \textbf{ in } p2 \urcorner/p]$
ss
IH1 IH2 A

$w[\ulcorner p1; p2 \urcorner/p]$
ss
IH1 IH2 A

Here w is a formula with a free variable p of type P, and IH1 and IH2 are respectively, $w[p1/p]$ and $w[p2/p]$, and a, I, $p1$, $p2$ are fresh variable names. The three subgoals on the left are basis cases, and the three on the right, induction steps. In the steps, we add the induction hypotheses to the list of assumptions of the subgoals.

Another tactic needed (for the inner induction) must also be derived from the rule of computation induction (a simpler tactic than the standard one, in fact). It is parameterized on a list of subterms of the form FIX fun$i$.

SIMPLEINDUCTAC1 [FIX fun1, . . . , FIX fun$n$]
w[FIX fun$i$/fi], ss, A

$w[\bot/fi]$
ss
A

$w[\text{fun}i\ fi'/fi]$
$w[fi'/fi] \vdash w[fi'/fi]$ ss
$w[fi'/fi]$ A

The tactic produces the basis and step subgoals for simulataneous computation induction. (The standard induction tactic does not expect subterms of the form FIX fun*i* to occur in the formula, but rather, subterms *Fi*, and takes as parameters theorems of the form $\vdash Fi = \text{FIX fun}i$.) This is used by a tactic (SIMPLEINDUCTAC) which finds the terms FIX fun*i* in the formula of the goal.

For various reasons, a theorem may be useful at some point in a proof effort, but not as a simplification rule; for example, it may simply be of the wrong form to be used as a simprule. In these cases, we may wish to introduce the conclusion of the theorem into the list of assumptions, planning to apply it in the proof effort (by methods to be discussed presently). The tactic we have in mind is

USELEMMATAC $(A' \vdash w')$
(w, ss, A)

w
ss
w' A

(This is useful when $A' \subseteq A \cup \text{Hypo}(ss)$, where Hypo (ss) is the set of hypotheses of theorems on which the simprules in ss are based.) The justification discharges w' from a theorem given it, and evaluates Modus Ponens on the result, and the theorem $A' \vdash w'$.

Another common reason that a theorem may be useful in a proof, but not as a simplification rule (although it may even be otherwise admissible as a simprule), is that simplification matches only to the left-hand side of the equivalence. This match does not necessarily determine an instantiation for all of the variables free in the right-hand side (or in the antecedent, if there is one). For example, consider a simprule based on

$$A \vdash \{\forall x\, y\,.\}w[x, y] \supset t[x] = t[y]$$

where x and y are not free in A, and where w is a formula, t is a term, and $w[x1, \ldots, xn]$ indicates that exactly $x1, \ldots, xn$ are free in w (and likewise $t[x1, \ldots, xn]$). In using this theorem to simplify some formula $t[x'] = t[y']$, for example, x will be instantiated to x', but y will not be instantiated. The replacement of $t[x']$ by $t[y]$ will be made if $w[x', y]$, the instantiated antecedent, can be simplified to a tautology. This can happen only by coincidence, since the variable y is not related to the formula being proved. Even if the antecedent is proved by coincidence of variable names, the replacement of $t[x']$ by $t[y]$ will introduce the arbitrary variable y into the proof (it will occur in the formula of the subgoal produced by the simplification: $t[x'] = t[y]$). Nothing false can be proved in LCF, since theorems can only be the results of applying PPLAMBDA rules of inference, but some tactics can give unsolvable subgoals. That is likely to be the case if a conditional simprule is applied by coincidence. (The new subgoal involving y may also be solvable, but again only by coincidence.) To avoid this sort of simplification, we will assume that theorems of the form discussed (i.e. in which the antecedent or the right-hand side of the consequent contain instantiable variables not in the left-hand side of the consequent) cannot be engaged as simprules. In the proof of the equivalence of $\mathcal{S}$ and $\bar{\mathcal{S}}$, both of the induction hypotheses and the lemma have this form, so we must consider other ways to use such facts.

To use a theorem of this form in a proof, we suppose it has had its conclusion added to the assumption list of our goal by USELEMMATAC. Two methods for using the theorem are adequate for our purposes.

In the simpler situation, an instance of the antecedent, say $w[x', y']$ is already an assumption. (It must also be the case that in the theorem in question, all variables free in the consequent are free in the antecedent; that is true in our example.) Then matching the assumption $w[x', y']$ to the antecedent $w[x, y]$, determines an instantiation $[x'/x, y'/y]$. We assume the first formula to get

$$w[x', y'] \vdash w[x', y'],$$

and evaluate Modus Ponens with the theorem we are trying to use and the new one to prove

$$A, w[x', y'] \vdash t[x'] = t[y'],$$

which does make sense, and can reasonably be used as a simprule, as it involves only the particular x' and y' which occur in the assumption $w[x', y']$.

To accomplish this step in proofs, we wish to search the list of assumptions for a pair of formulae which satisfy the above constraints, and for any we find, we assume the two formulae, evaluate Modus Ponens, and use the new theorem as a simprule if possible. As this is a very primitive form of resolution [5], we call the tactic which does it RESTAC.

RESTAC

$w1$
ss
$\forall x1 \ldots xn \,.\, w$ $\forall y1 \ldots ym \,.\, z1 \ldots zp \,.\, w' \supset w''$ A

where w matches w' by instantiating yi to yi'
but not instantiating zi
(the y and z can be in any order)

$w1$
$[\forall x1 \ldots xn \,.\, w, \forall y1 \ldots z1 \ldots zp \,.\, w' \supset w''] \vdash \forall z1 \ldots zp \,.\, w''[yi'/yi]$ as above
$\forall z1 \ldots zp \,.\, w''[yi'/yi]$ as above

That is, RESTAC partitions the assumption list into formulae whose bodies are and are not implications, and for each pair in the cartesian product of the lists attempts to match the nonimplication with the antecedent of the implication, proving a new theorem if possible, adding it to the simpset (if possible), and its conclusion to the assumption list.

RESTAC is potentially useful whenever an assumption is added to the assumption list of a goal. This allows the new assumption to "resolve" with other assumptions (which may have been added by DISCHTAC, PINDUCTAC, CONDCASESTAC, or other tactics).

There are obviously much more sophisticated procedures which one could apply to a list of assumptions to derive new, possibly useful theorems, but this simple one is sufficient for the current proof, and probably many others.

Of course, it may be the case that one wishes to use a theorem, an instance of whose antecedent is not already in the list of assumptions. (Suppose that all free variables of the antecedent also appear free in the consequent.) During a proof effort, when we eventually arrive at a subgoal whose formula is an instance of the consequent, (i.e. when we are ready to use the theorem), we can generate the correct instance of the antecedent as a subgoal.

A tactic which clearly reflects this reasoning must search the assumption list of a goal for a formula whose consequent is matched by the formula of the goal, and use the instantiation determined by the match to produce a subgoal whose formula is the correct instance of the antecedent. To use implicative assumptions in this way, we write

USEIMPASSUMPTAC

$w2'$, ss, $[\ldots, \forall x1 \ldots xn \,.\, y1 \ldots ym \,.\, w1 \supset w2, \ldots]$
where $w2'$ matches $w2$ with xi instantiated to xi',
yi are not instantiated;
the x and y may be in any order, and
all free variables of $w1$ occur free in $w2$

$(\forall y1 \ldots ym \,.\, w1[xi'/xi]$, ss, as above$)$

The justification part of the tactic obviously uses Modus Ponens.

Having written DISCHTAC, USELEMMATAC, RESTAC and USEIMPASSUMPTAC, we are now prepared to do the proof in LCF.

The tactical proof. The goal which the lemma achieves has formula component

$$\rho \sim v \supset \rho[\bar{\mathscr{P}}[\![p']\!]v'/J] \sim v[(p', v')/J].$$

In the simpset, we include an axiom defining function extension, and the following pair of (easy) theorems:

$$\vdash \forall Iv \,.\, \bar{\mathscr{P}}[\![\textbf{call } I]\!]v = \bar{\mathscr{P}}[\![p']\!]v' \quad \text{where } (p', v') = v\,I,$$

$$\vdash \bar{\mathscr{P}}[\![\bot]\!] = \bot.$$

The compound tactic which solves the lemma is:

DISCHTAC
GENTAC$^+$
CONDCASESTAC$^+$.

The correspondence to the informal proof is clear; DISCHTAC accomplishes the step of proving the consequent under the assumption of the antecedent; GENTAC does the step of proving for arbitrary values; the following SIMPTAC rewrites terms involving function extension according to the axiom defining it, so that conditional terms such as (**if** $I = J$ **then** $\bar{\mathscr{P}}[\![p']\!]v'$ **else** $\rho\,I$) appear in the formula, and it applies the assumption of the antecedent to the subterm $\rho\,I$; CONDCASESTAC performs cases analysis on the conditional terms, and the following SIMPTAC solves all three cases. Applied to a goal with the appropriate formula and simpset, the compound tactic produces an empty list of subgoals, and a justification which when applied to an empty list of theorems (respectively achieving the empty list of subgoals) yields a theorem achieving the original goal.

The goal which the main theorem achieves has the formula

$$\forall p \rho v \,.\, \rho \sim v \supset \mathcal{S}[\![p]\!]\rho = \bar{\mathcal{S}}[\![p]\!]v.$$

The simpset includes six theorems, which are the clauses for $\mathcal{S}$, and the five theorems, which are the clauses for $\bar{\mathcal{S}}$; the theorem

$$\vdash \bar{\mathcal{S}}[\![\textbf{call } I]\!]v = \mathcal{S}[\![p']\!]v' \quad \text{where } (p', v') = v\,I$$

is not necessary in this proof. The compound tactic which solves the main theorem is

USELEMMATAC lemma
PINDUCTAC$^+$
GENTAC*
DISCHTAC
RESTAC$^+$
USEIMPASSUMPTAC

RESTAC$^+$ SIMPLEINDUCTAC$^+$
RESTAC$^+$

Again, the correspondence to the informal proof is quite clear. USELEMMATAC adds the lemma to the assumptions of the goal to be achieved. PINDUCTAC produces the undefined and the other five cases, and the following SIMPTAC solves the undefined case. GENTAC* accomplishes the step of proving for arbitrary values; DISCHTAC, as before, does the step of proving the consequent under the assumption of the antecedent. RESTAC resolves the assumption introduced by DISCHTAC with the induction assumptions introduced by PINDUCTAC, to yield the correct consequents (as simprules). The following SIMPTAC solves the **sequencing** case by applying the induction hypotheses, and solves the **call** case by applying the assumption introduced by DISCHTAC. It also simplifies the subterm $\mathcal{S}[\![p1]\!]\rho$ which occurs in the **let** case to $\bar{\mathcal{S}}[\![p1]\!]v$. This leaves the modified **let** and the **letrec** case. USEIMPASSUMPTAC is invoked to use the induction hypothesis for $p2$ on each of the remaining two subgoals, producing subgoals whose formulae are the required instances of the antecedents of the induction hypotheses. In the **letrec** case, SIMPLEINDUCTAC does the inner induction, and the following SIMPTAC solves the undefined case. In both remaining cases, the lemma, carried along as an assumption thus far, is resolved (respectively) by RESTAC with the assumption contributed by DISCHTAC, and the new (inner) induction assumption, to produce new theorems useful as simprules. In each case, SIMPTAC then produces the empty list of subgoals.

Again, the compound tactic solves the goal in one application. Further, it reflects a general strategy for proving implicative formulae by structural induction. The strategy copes with induction hypotheses which cannot necessarily be engaged as rewrite rules, and with previously proved lemmas which also cannot be used thusly. Only the inner computation induction is in any way special to this proof.

The formalization in LCF. The new types, constants and axioms required to state the problem are arranged in a hierarchy of LCF theories.

- SYNT: The syntax of the language, for which the types ":ID", ":ATOM" and ":P" are introduced;
- SHASEM: those aspects shared by both semantic definitions;
- SSEM: The semantic function "S" (for $\mathcal{S}$) is defined;
- SSSEM: The semantic function "SS" (for $\bar{\mathcal{S}}$) is defined;

• EQUIV: The theory in which the equivalence of "S" and "SS" is stated and proved. In addition, we construct a theory FUNEXT, of function extension, used in both semantic definitions. This depends on a general theory of EQUALITY in which we introduce a constant "EQ" of type ":$*\to *\to$tr", for any type ":$*$". In the theory of function extension, we need a constant "extend" of type ":$(*\to **)\to (**\to (*\to (*\to **)))$", governed by the axiom

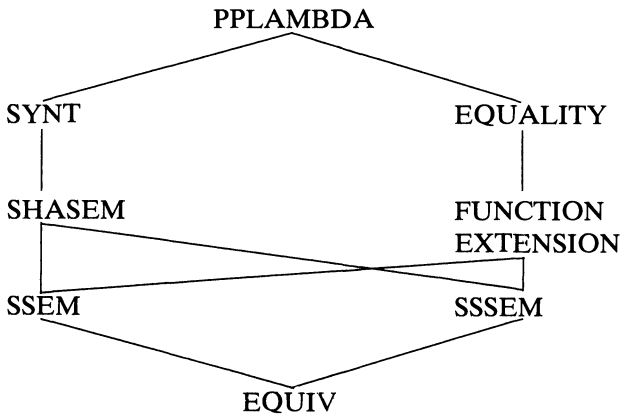
$$\vdash \text{"}!f \text{ val var } y \text{ . extend } f \text{ val var } y == \textbf{if } \text{EQ } y \text{ var } \textbf{then } \text{val } \textbf{else } f\, y\text{"}.$$

Writing "ro" for ρ, we can then write "extend ro(S p1 ro)I" for $\rho[\mathscr{S}[\![p1]\!]\rho/I]$.

The theories are connected in a network as shown below, in which, for theories $T1$ and $T2$,

$$\begin{array}{c} T1 \\ | \\ T2 \end{array}$$

means that all types, constants, axioms and theorems of $T1$ are accessible within $T2$. PPLAMBDA is the pure logic of LCF:



For the sake of computer printing, the following conventions apply:

σ is written as	"sig"
ρ	"ro"
$\mathscr{S}$	"S"
$\bar{\mathscr{S}}$	"SS"
$\mathscr{A}$	"A"
$\mathscr{S}[\![p]\!]\rho$	"S p ro".

Both the type ":P" for the programs and ":CENV" for closure environments are recursive types. In SYNT, the theory of syntax, we introduce the new type ":P", and represent programs as separated sums of the five possible types of constructs, three of which involve ":P". to this end, we define some abbreviations:

":CALL"	for ":ID"
":LET"	for ":ID×P×P"
":LETREC"	for ":ID×P×P"
":SEQ"	for ":P×P".

We represent ":P" by defining an isomorphism between the type and the representation:

$$P \underset{\text{ABSHP}}{\overset{\text{REPHP}}{\rightleftarrows}} (\text{ATOM } u + \text{CALL } u + \text{LET } u + \text{LETREC } u + \text{SEQ } u).$$

The two new function constants needed are:

"ABSHP: $(\text{ATOM } u + \text{CALL } u + \text{LET } u + \text{LETREC } u + \text{SEQ } u) \to P$"
"REPHP: $P \to (\text{ATOM } u + \text{CALL } u + \text{LET } u + \text{LETREC } u + \text{SEQ } u)$"

where u is the PPLAMBDA type operator corresponding to lifting of domains. These are governed by the axioms:

$$\vdash \text{"!}p \text{ . ABSHP(REPHP } p) == p\text{"}$$
$$\vdash \text{"!}x \text{ . REPHP(ABSHP } x) == x\text{"}.$$

Likewise, in SSSEM, we introduce a new type ":CENV" and define an isomorphism:

$$\text{CENV} \underset{\text{ABSCENV}}{\overset{\text{REPCENV}}{\rightleftarrows}} (\text{ID} \to (P \times \text{CENV}))$$

using

"ABSCENV: $(\text{ID} \to (\hat{P} \times \text{CENV})) \to \text{CENV}$"
"REPCENV: $\text{CENV} \to (\text{ID} \to (\hat{P} \times \text{CENV}))$"

and the axioms

$$\vdash \text{"!}v \text{ . ABSCENV(REPCENV } v) == v\text{"},$$
$$\vdash \text{"!}x \text{ . REPCENV(ABSCENV } x) == x\text{"}.$$

Thus the expression $(\mathscr{S}[\![p']\!]v'$ where $(p', v') = v\, I)$ is written as "SS(FST (REPCENV $v\, I$))(SND(REPCENV $v\, I$))". In both cases, it is a simple matter to prove in LCF that the abstraction and representation functions are strict.

Typical new constants of the theory SYNT are "mkcall: $\text{CALL} \to P$" and "mkseq: $\text{SEQ} \to P$" to construct **call** and **sequencing** programs, respectively; we then write "SS(mkcall I)v" and "SS(mkseq($p1, p2$))v" for $\mathscr{S}[\![\textbf{call}\ I]\!]v$ and $\mathscr{S}[\![p1; p2]\!]v$. Likewise, we have the other constructors "mkatom", "mklet" and "mkletrec". (The axioms defining these constructors can be generated in a uniform way, given the relevant type definitions; this is in fact done by R. Milner's structural induction generating package discussed earlier. The current proof was done before that package was written, and there are some inconsistencies between the present treatment and the uniform treatment, but in principle, the package could have been used here to build the theory SYNT.)

In the theory of $\mathscr{S}$,(SSEM), a fixed point definition is given of "S", since "S" is recursively defined, from which it is easy to derive in LCF the separate semantic clauses:

$\vdash$ "!ro . S UU ro == UU",
$\vdash$ "!a ro . S(mkatom a)ro == $A\ a$",
$\vdash$ "!I ro . S(mkcall I)ro == ro I",
$\vdash$ "!$I\,p1\,p2$ ro . S(mklet($I, p1, p2$))ro == S $p2$(extend ro(S $p1$ ro)I)",
$\vdash$ "!$I\,p1\,p2$ ro . S(mkletrec ($I, p1, p2$))ro ==
S $p2$(FIX(\ro' . extend ro(S $p1$ ro')I))",
$\vdash$ "!$p1\,p2$ ro . S(mkseq ($p1, p2$))ro == \sig . S $p2$ ro(S $p1$ ro sig)".

Likewise, in SSSEM, we define "SS" as a least fixed point, and prove

$\vdash$ "!v. SS UU v == UU",
$\vdash$ "!$a\,v$. SS(mkatom a)v == $A\,a$",
$\vdash$ "!$I\,v$. SS(mkcall I)v ==
SS(FST(REPCENV $v\,I$))(SND(REPCENV $v\,I$))",
$\vdash$ "!$I\,p1\,p2\,v$. SS(mklet (I, $p1$, $p2$))v ==
SS $p2$(ABSCENV(extend(REPCENV v)($p1$, v)I))",
$\vdash$ "!$I\,p1\,p2\,v$. SS(mkletrec (I, $p1$, $p2$))v ==
SS $p2$(FIX(\v'. ABSCENV(extend(REPCENV v)($p1$, v')I)))",
$\vdash$ "!$p1\,p2\,v$. SS(mkseq ($p1$, $p2$))v == \sig . SS $p2$ v(SS $p1$ v sig)"

as well as the useful theorems

$\vdash$ "SS UU UU == UU",
$\vdash$ "!I. SS(mkcall I)UU == UU".

All twelve clauses depend on the strictness of "ABSHP", "REPHP", "ABSCENV" and "REPCENV". The proofs in LCF are accomplished mainly by simplification.

Details of the proof of the theorem in LCF. Readers wishing to know no more about the proof may skip this section!

The tactical proofs correspond quite closely to the informal proofs; this can be seen by inspecting the tactics which solve the two goals.

Since one cannot use predicate constants or variables within PPLAMBDA (as one *can* use function constants and variables) the relation $\sim$ must appear as the formula it abbreviates. To prevent this being cumbersome in the definition of goals, we define an ML function rel with type (term → term → formula), which constructs the formula given two terms of appropriate type (that is, type ":ENV" and ":CENV"). The procedure rel is analogous to the standard ML function mkimp of type (formula × formula → formula) which constructs implications from pairs of formulae.

The formula of the goal for the lemma is

```
mkimp (rel "ro" "v",
       rel "extend ro (S p' v')J",
           "ABSCENV (extend (REPCENV v)(p', v')J)").
```

The simpset for this goal contains the axiom from FUNEXT defining "extend", the axioms from SSEM relating "ABSCENV" and "REPCENV", and the two theorems (easily) proved in SSSEM:

$\vdash$ "!$I\,v$. SS (mkcall I)v ==
SS (FST (REPCENV $v\,I$))(SND (REPCENV $v\,I$))",
$\vdash$ "SS UU UU == UU".

The compound tactic shown earlier solves the goal. When it is stored, under the name lemma, say, in EQUIV theory, free variables are automatically quantified to give:

$\vdash$ "!ro $v\,p'\,v'\,J$. (!I . ro I == SS (mkcall I)v) IMP
(!I . extend ro (SS $p'\,v'$)$J\,I$ ==
SS (mkcall I)(ABSCENV (extend (REPCENV v)(p', v')J)))".

The goal which the main formula achieves is constructed similarly, using rel. Its simpset includes the thirteen theorems mentioned earlier, as well as the (easily proved) facts asserting the strictness of "ABSCENV" and "REPCENV".

We begin the proof by applying to the goal

PINDUCTAC$^+$
GENTAC*
DISCHTAC.

PINDUCTAC returns six subgoals, of which SIMPTAC subsequently solves two, the atomic and undefined cases. After the rest of the compound tactics, the remaining four subgoals have the following formulae:

call case: "ro I' == SS (mkcall I')v"
let case: "S p2 (extend ro (S p1 ro)I') ==
SS p2 (ABSCENV (extend (REPCENV v)(p1, v)I'))"
letrec case: "S p2 (FIX (\ro' . extend ro (S p1 ro')I')) ==
SS p2 (FIX (\v' . ABSCENV (extend (REPCENV v)(p1, v')I')))"
sequencing case: "\sig . S p2 ro (S p1 ro sig) ==
\sig . SS p2 v(SS p1 v sig)"

and the following assumption and corresponding simprule, in each case, contributed by DISCHTAC:

"!I ro . ro I == SS (mkcall I)v".

In addition, all but the **call** case have the two induction hypotheses as assumptions, which we shall call IH2 and IH2:

"!ro v . (!I. ro I == SS (mkcall I)v) IMP S p1 ro == SS p1 v",
"!ro v . (!I. ro I == SS (mkcall I)v) IMP S p2 ro == SS p2 v".

We then apply RESTAC; the assumption added by DISCHTAC is resolved with IH1 and IH2, in turn, to give two new simprules:

"S p1 ro == SS p1 v",
"S p2 ro == SS p2 v".

Next we apply SIMPTAC. In the **call** case, the assumption

$\vdash$ "!I. ro I == SS (mkcall I)v"

in the simpset is applied, and the subgoal is solved. The **sequencing** subgoal is also solved, using the two new elements of the simpset, and the **let** subgoal has a rewritten formula

"S p2 (extend ro (SS p1 v)I') ==
SS p2 (ABSCENV (extend (REPCENV v)(p1, v)I'))".

Not all uses of the induction hypotheses can be applied in this fashion. For example, the **letrec** and modified **let** subgoals cannot be solved by an application of RESTAC, because the corresponding antecedents,

"!I. extend ro (SS p1 v)I' I ==
SS (mkcall I) (ABSCENV (extend (REPCENV v)(p1, v)I'))",

"!I. FIX (\ro' . extend ro (S p1 ro')I')I ==
SS (mkcall I) (FIX (\v' . ABSCENV (extend (REPCENV v)(p1, v')I')".

do not appear as assumptions (as they did, say, in the **sequencing** case). Instead, we apply the tactic USEIMPASSUMPTAC to produce subgoals whose formulae are the correctly instantiated antecedents.

Some care is needed here to use the correct induction hypothesis; we wish to use IH2 at this point, not IH1, although the conclusion of IH1 does match either of the current subgoals' formulae. We wish, that is, to respect the induction variables "$p1$" and "$p2$" as induction variables by never instantiating them. In the uses of IH1 and IH2 by RESTAC—in solving the **sequencing** case, for example—the induction variables are respected simply because they are free in IH1 and IH2. In USEIMPASSUMPTAC, we can achieve this effect by allowing a match only when variables to be instantiated in a formula are not free in that formula.

This problem would occur in any structural induction proof in which a property is assumed of more than one substructure (and in which simplification does not handle all uses of the induction hypotheses).

The lemma we wish to bring to bear, at this point, is the one we have already proved, and called lemma:

"! ro $v\ p'\ v'\ J$. (!I. ro I == SS (mkcall I)v) IMP
(!I. extend ro (SS $p'\ v'$)$J\ I$ ==
SS (mkcall I)(ABSCENV (extend (REPCENV v)(p', v')J)))".

We could not, first of all, use the lemma as a simprule without first putting it in a canonical form (with no quantifiers in the consequent), but that can easily be done. However, we cannot even use the variant of the lemma as a conditional simprule, again because it has a variable (v) instantiable in the antecedent but not occurring in the left-hand side of the consequent. Instead, we introduce it into the list of assumptions, using USELEMMATAC; this can be inserted at any point in the sequence of tactics thus far, but we adopt a convention of adding lemmas at the beginning.

To the **letrec** subgoal, we then apply SIMPLEINDUCTAC, to correspond to induction on $\mathscr{R}$ and $\mathscr{V}$ in the informal proof, to obtain a subgoal with formula

"!I. extend ro (S $p1$ indvar1)$I'\ I$ ==
SS (mkcall I)(ABSCENV (extend (REPCENV v)($p1$, indvar2)I'))"

for fresh variables "indvar1" and "indvar2", corresponding to $\bar{\rho}$ and $\bar{v}$ in the informal proof. SIMPLEINDUCTAC adds the simprule (the new induction hypothesis)

$\vdash$ "!I. indvar1 I == SS (mkcall I) indvar2".

The **letrec** subgoal's formula now contains a subterm "S $p1$ indvar1", which we wish to rewrite as "SS $p1$ indvar2", based on IH1. To do this, we have to resolve (by applying RESTAC) the assumption of "!I. indvar1 I ==SS (mkcall I) indvar2" with the assumption of IH1 to prove

$\vdash$ "S $p1$ indvar1 == SS $p1$ indvar2"

(with the two assumptions as hypotheses) and to use that as a simprule. In both subgoals, then, we also wish to resolve the assumption of "!I. ro I == SS (mkcall I)v" with the lemma, to prove

$\vdash$ "!$p'\ v'\ J\ I$. extend ro (SS $p'\ v'$)$J\ I$ ==
SS (mkcall I)(ABSCENV (extend (REPCENV v)(p', v')J))".

With this new theorem as a simprule, both cases are solved, and the proof can be evaluated.

Conclusions. We have shown how the equivalence of two semantic definitions has been stated and proved in the LCF system. In the formalization we use:

- an ML procedure rel to construct the formula expressing a relation between PPLAMBDA objects (namely, the simulation relation between contexts), since PPLAMBDA does not support predicate constants;
- an ML procedure, due to R. Milner, for deriving a tactic for performing structural induction on suitable well-founded structures;
- definitions of circular types via isomorphisms between the types and their representations.

At several points in the proof effort we observed that the use of certain implicative theorems as left-to-right rewrite rules did not produce the desired effect; in each case, the problem was that the match of a subterm to the left-hand side of the consequent of the conclusion of the theorem did not necessarily instantiate all of the variables occurring in the antecedent, or in the right-hand side of the consequent. This applied to both uses of the lemma, and to all uses of the two induction hypotheses.

In the simpler cases, the desired instance of the antecedent of the theorem was already an assumption of the goal (and there were no variables free in the consequent not also free in the antecedent). This situation pertained to all uses of the induction hypotheses aside from those of IH2 in the **let** and **letrec** cases, as well as to both uses of the lemma. For these cases, the simplest solution seemed to be to apply a very simple resolution tactic which checked all assumptions for matches to the antecedent in question, proving the desired instance of the conclusion by Modus Ponens.

When the desired instance of the antecedent was not already in the simpset and could not have been anticipated, (and when all variables free in the antecedent were also free in the consequent) we used a tactic which attempted a match of the current subgoal's formula to the consequents of all implicative assumptions. This produced the desired instance of the antecedent as the formula of the subgoal (as simplification could not). In addition, this tactic respected the induction variables by only instantiating bound variable in matching, so that if the assumption in question was an induction hypothesis, the (free) induction variables were not instantiated.

Some thought was required about the contents of the simplification sets of goals, to facilitate the automatic parts of the proof effort. For example, as observed, by designing a version of DISCHTAC which did not add a simprule, we would have prevented SIMPTAC from solving the **call** call of the main theorem. By excluding the theorem

$$\vdash \text{“!}I\,v\,.\ \text{SS (mkcall}\ I)v\ ==\ \text{SS(FST(REPCENV}\ v\ I))(\text{SND(REPCENV}\ v\ I))$$

from the simpset of the goal for the main theorem, we made the proof simpler by never having to make and justify that particular rewriting; but realizing that that theorem is not needed for the **call** case (whereas the corresponding ones were needed for the other cases) required a certain prior understanding of the proof. The point is that while simplification does large parts of the proof automatically—at least in the sense that only a small proportion of the primitive inference steps in the actual proofs performed are generated by tactics other than SIMPTAC—the selection of simprules is a delicate matter requiring thought and planning. The structuring of the proof effort into the lemma and the main theorem also required thought.

In general, all case studies of this sort are feasibility studies. The mechanical generation of a fairly long and (even informally) nontrivial proof such as this is an encouraging result. The tactics designed for the purpose seem to apply in quite general proof situations, and the compound tactic that solved the main goal also reflects (apart from the inner induction) a quite general strategy, for proving implications by structural induction. Finally, while LCF is only a framework in which proof strategies can be

expressed, and makes no claim to being an automatic theorem prover, the simple resolution tactic described here gives some indication of how more automatic theorem-proving strategies might be implemented as tactics in ML.

REFERENCES

[1] AVRA JEAN COHN, *Machine assisted proofs of recursion implementation*, Ph.D. Thesis, Dept. of Computer Science, Edinburgh University, Edinburgh, 1979.

[2] AVRA COHN AND ROBIN MILNER, *On using Edinburgh LCF to prove the correctness of a parsing algorithm*, Edinburgh University Computer Science Department Report CSR-113-82, 1982; and Cambridge University Computer Laboratory Technical Report 20, 1982.

[3] MICHAEL J. C. GORDON, ROBIN MILNER AND CHRISTOPHER P. WADSWORTH, *Edinburgh LCF*, Lecture Notes in Computer Science 78, Springer-Verlag, Berlin, 1979.

[4] R. E. MILNE AND C. STRACHEY, *A Theory of Programming Language Semantics*, Chapman and Hall, London, 1976.

[5] J. A. ROBINSON, *Logic: Form and Function*, Edinburgh University Press, Edinburgh, 1979.

[6] JOSEPH E. STOY, *Denotational Semantics: The Scott–Strachey Approach to Programming Language Theory*, MIT Press, Cambridge, MA, 1977.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0005 $01.25/0

COMPUTING RATIONAL ZEROS OF INTEGRAL POLYNOMIALS BY p-ADIC EXPANSION*

RÜDIGER LOOS†

Abstract. We design, analyze and measure several algorithms for finding the rational zeros of univariate polynomials with integral coefficients of any size. The quadratic Newton–Hensel type algorithm behaves best in theory and practice. It has a maximum computing time of $O(n^3L(d)^2)$, where n is the degree and d measures the coefficient size of the given squarefree polynomial using classical arithmetic. We extend the method to Gaussian rational numbers.

Key words. rational zeros of univariate polynomials, polynomial factorization, Newton–Hensel algorithm, p-adic expansion

1. The problem. Given a polynomial $A(x)=\sum_{i=0}^{n} a_i x^i$ of degree n with integer coefficients of arbitrary size, we search for rational numbers s/t, $t\neq 0$, such that $A(s/t)=0$. Obviously the problem is equivalent to finding all linear factors of A, since s/t being a rational zero implies $(tx-s)$ being a linear factor of A. The problem can be considered as a special instance of the problem of finding real zeros of A or of the problem of factorizing A into irreducible factors.

In order to measure the complexity of a polynomial $A=\sum_{i=0}^{n} a_i x^i$ we use the degree n in its only indeterminate and the length $L_\beta(d)=\lfloor \log_\beta |d| \rfloor + 1$ of the seminorm $d=\sum_{i=0}^{n}|a_i|$. Here, β is the basis of the classical integer arithemtic and is of the order of the largest integer fitting in a single word. Since β is a constant we write $L(a)$ instead of $L_\beta(a)$ and simplify the analysis by assuming $n<\beta$ and $d<\beta^\beta$, i.e., we consider the degree and the length of the integers to be single precision numbers.

The maximum computing time of our best algorithm for finding all rational zeros is $O(n^3L(d)^2)$. The strongest competing algorithm tests for integral zeros of a related polynomial using the modified Uspensky real root isolation method and refinement to intervals of length less than 1 and achieves an upper bound of $O(n^4L(d)^2)$ [1]. Surprisingly, the specialization of Musser's [2] polynomial factorization algorithm based on Berlekamp's and Hensel's algorithms achieves also a bound of $O(n^3L(d)^2)$; however, the constants involved are higher.

The applications of rational zero findings are numerous. Working in algebraic extensions $Q(\alpha)$ one would like to be sure that α is a true irrational number. So, the defining polynomial for α should be square free and, in addition, rational zero free. In Gosper's [3] algorithm for indefinite summation one needs the positive integral zeros of a certain resultant, which tends to have large coefficients even for small problems. A similar problem arises if one searches for quadratic factors, or in general, for factors of any fixed degree. Therefore, one needs rational zero finding methods which are efficient for large coefficients.

2. Exhaustive search. The idea, which comes first to mind and which goes back to Newton, is an exhaustive search among the factors of the trailing coefficient $tc(A)$ of A and the leading coefficient $lc(A)$. Assuming s/t, $t\neq 0$, reduced to lowest terms, $A(s/t)=0$ implies $a_n s^n + a_{n-1}s^{n-1}t + \cdots + a_1 s t^{n-1} + a_0 t^n = 0$ from which $s|a_0$ and $t|a_n$ follows. Hence the exhaustive search algorithm is:

ALGORITHM L ← IPRZEH(A).
[Integral polynomial rational zero exhaustive search. A is a univariate integral polynomial. L is a list of its rational zeros.]

* Received by the editors April 19, 1979, and in final revised form March 1, 1982.
† Institut für Informatik I, Universität Karlsruhe, D-7500 Karlsruhe, West Germany.

(1) [Initialize.] Set $a_n = |lc(A)|$, $a_0 = |tc(A)|$, $L = (\ )$, the empty list.
(2) [Exhaustive search.]
for $s = -a_0, \cdots, a_0$ do
if $s = 0 \vee s|a_0$ then
for $t = -a_n, \cdots, a_n$ do
if $t \neq 0$ & $t|a_n$ then {reduce s/t; if $A(s/t) = 0$
11081 then augment L by s/t}.

The computing time of step 1 is dominated (see [4] for a definition of dominance) by $L(a_0)+L(a_n)$. Evaluation of $A(s/t)$ by Horner's method needs $O(n^2\{L(s)+L(t)\}^2 + n^2L(d)\{L(s)+L(t)\})$ or, using $L(a_0)$, $L(a_n) \leqq L(d)$ and $s \leqq |a_0|$, $t \leqq |a_n|$, $O(n^2L(d)^2)$ steps. Therefore the total computing time is $O(|a_0||a_n|n^2L(d)^2)$, which makes the algorithm exponential in d; the obvious method is only useful for very small a_0 and a_n.

We ran the examples in Table 1 from the collection [5] with Collins' SAC2/ALDES system on a Burroughs 7700.

TABLE 1

Polynomial	Rational zeros	Seconds
1. $6x^4-11x^3-x^2-4$	$-\frac{2}{3}, 2$	0.54
2. $2x^3+12x^2+13x+15$	-5	0.29
3. $2x^4-4x^3-3x^2-5x-2$	2	0.17
4. $6x^5+11x^4-x^3-5x-6$	$\frac{2}{3}, -\frac{3}{2}$	0.80
5. $x^5-5x^4+2x^3-25x^2-21x+270$	$-2, 3, 5$	0.70
6. $2x^6+x^5-9x^4-6x^3-5x^2-7x-6$	$\frac{1}{2}, -2$	0.45

3. The linear Newton–Hensel algorithm. The main drawback of the exhaustive search method is the size of the search space, which depends on a_0 and a_n. Exponential computing times in the size of numerical coefficients can be eliminated by mapping the problem into an image domain with elements of fixed size. Here we take the complete residue system $0, 1, \cdots, p-1$ representing Z_p, the integers modulo p, where p is a prime. The algorithm starts with picking a small prime, maps A to $A^* \equiv A \bmod_p$ with coefficients in Z_p and applies the exhaustive search to the p elements of Z_p, resulting in a list L^* of zeros of A^* over Z_p.

In order to simplify the discussion, we assume for the moment that $a_n = 1$, in which case A is called monic. As a consequence $t|a_n$ restricts the rational zeros of A to integral zeros of A. The map $A^* = h_p(A)$ is a homomorphism leaving $0, 1$ and addition and multiplication invariant, the only operations involved in testing $A(s) = 0$. Hence, if L^* is empty, A has no integral zeros. On the other hand, A^* must have at least as many zeros as A has integral zeros, multiplicities counted. However, in general A^* will have more zeros than A has integral ones. For example, x^4-1 has no integral zero over Z but 1 over Z_2. If the integral zeros of A lie in the range of Z_p, then they form a subset of L^*, which must be found by testing. The important problem remains, how to come back to Z having the zeros of A^* in Z_p, if the integral zeros of A lie not in the range of Z_p.

There are two answers. The first one uses the Chinese remainder theorem after having constructed L^* for several primes. Each cross section of the lists L^* consisting of at most one element from each list forms a modular representation of a possible integral root of A. Since there is no indication which components of the modular representation belong to each other, the combinatorial problem arises to try all possible cross sections. This leads again to a computing time exponential in n and $L(d)$.

A better solution is based on a special version of a Newton–Hensel type theorem as given by Lewis [6].

THEOREM 1. *Let $A(x)$ be an integral polynomial and let α be a zero of $A(x) \bmod p$, which is not a zero of the derivative $A'(x) \equiv 0 \bmod p$, p a prime. Then for each $r \geqq 1$ the congruence $A(x) \equiv 0 \bmod p^r$ has a solution $\alpha_r \equiv \alpha \bmod p$.*

Proof. For $r = 1$ the result is the hypothesis. Suppose the theorem holds for $r \leqq N$. Set $b_N = A(\alpha_N)/p^N$, where the division is exact by the induction hypothesis. Further set $\alpha_{N+1} = \alpha_N + p^N y$, with indeterminate y. Then

$$\begin{aligned} A(\alpha_{N+1}) &\equiv A(\alpha_N) + p^N y A'(\alpha_N) \bmod p^{N+1} \\ &\equiv p^N (b_N + y A'(\alpha)) \bmod p^{N+1}. \end{aligned}$$

Since $A'(\alpha) \not\equiv 0 \bmod p$ by assumption, y can be set to $(-b_N/A'(\alpha)) \bmod p$, so $A(\alpha_{N+1}) \equiv 0 \bmod p^{N+1}$ holds. □

Note the similarity in $\alpha_{N+1} = \alpha_N - (((A(\alpha_N)/p^N)/A'(\alpha)) \bmod p)p^N$ to the usual linear Newton approximation. The successive α_i are p-adic expansions of α to order i. In a computational context this expansion is called lifting.

The constructive proof allows us to fill the gap in our algorithm by using the p-adic expansion of the zeros of A^* up to an order N such that the integral zeros of A lie in the range $-\lfloor p^N/2 \rfloor, \cdots, \lfloor p^N/2 \rfloor$. In order to satisfy the hypothesis that the elements in L^* are not zeros of $A'(x) \bmod p$ we require A to be squarefree and choose p as the smallest prime such that $A \bmod p$ is also squarefree, which means that p does not divide the (nonzero) discriminant of A. That p remains small enough to allow the exhaustive search in Z_p is established by:

THEOREM 2. *Let $A(x)$ be a squarefree integral polynomial of degree n and semi-norm d and let p_K, the k-th prime, be the first prime not dividing the discriminant of A. Then $k = O(nL(nd))$ and $p_k = O(nL(nd))$.*

Proof. Since A is squarefree, its discriminant $D \neq 0$. Let $p_1, p_2, \cdots, p_{k-1}$ be the first primes dividing D and let $p_k \nmid D$. Hence $2 \leqq p_1 < p_2 < \cdots < p_{k-1} \leqq |D|$ and $2^{k-1} \leqq p_1 \cdots p_{k-1} \leqq |D|$ and $k = O(L(D))$. The discriminant D is the resultant of A and A' which is the determinant of Sylvester's matrix. By Hadamard's inequality [7, 4.6.1 (24)] $|D| = O((\sqrt{n}d)^{n-1}(nd\sqrt{n}-1)^n)$, $L(D) = O(nL(nd))$ and $k = O(nL(nd))$, proving the first part. The second part is stated and proved in [7, 2nd ed., 4.6.2 Ex. 23]. □

Theorem 2 bounds the number of primes which must be tested before the hypothesis in the Newton–Hensel construction becomes true, and also the size of the search space Z_{p_k}.

Finally, we remove the restriction that A is monic. The textbooks apply the transformation $B(x) = a_n^{n-1} A(x/a_n)$ which associates with every rational zero $\alpha = s/t$ of A the integral zero $\beta = a_n s/t$ of the monic polynomial B and vice versa, given β, $\alpha = \beta/a_n$ reduced to lowest term. Computationally, this method is not attractive since the coefficient norm of B is $O(d^n)$ which makes the lifting algorithm slow.

Suppose we start with A and $p \nmid a_n$, map A to $A_1 \equiv A/a_n \bmod p$, find a zero α_1 with $A_1(\alpha_1) \neq 0$ and lift α_1 to α_n, A_1 to A_N for some positive integer N such that $A_N(\alpha_N) \equiv 0 \bmod p^N$. How is the integer α_N related to $\alpha = s/t$, the zero of A over the integers? Following Musser [2], the answer is given by:

THEOREM 3. *Let A be an integral polynomial with leading coefficient a_n, p a prime with $p \nmid a_n$ and N a positive integer. If A has a rational zero $\alpha = s/t$ then $A_N \equiv A/a_n \bmod p^N$ has a zero α_N over Z_{p^N} and $\alpha_N \equiv s \cdot \bar{a}_n \cdot a_n^{-1} \bmod p^N$, where $\bar{a}_n$ is the integer a_n/t.*

Proof. $A(s/t)=0$ implies $a_n=\bar{a}_n t$. Since otherwise $p \nmid a_n$, we have $p \nmid t$ and therefore a_n and t have inverses in Z_{p^N} for every $N>0$. Hence $\alpha_N \equiv st^{-1} \equiv st^{-1}a_n a_n^{-1} = s\bar{a}_n a_n^{-1} \bmod p^N$ is a zero of $A_N = A/a_n^{-1} \bmod p^N$. □

Since the converse of Theorem 3 does not hold, we have to test whether s/t corresponding to α_N is a rational zero. Given α_N we get the trial zero $\alpha = s/t$ by reducing the integers $\bar{s}$ and $\bar{t}$ to lowest terms, where according to the theorem we have to set $\bar{s} = \alpha_N a_n \bmod p^N$ and $\bar{t} = a_n \bmod p^N$.

Finally, we need a bound on N. We require that for all possible rational zeros s/t of A the integers $a_n s$ and t have to lie in the range $-\lfloor p^N/2 \rfloor, \cdots, +\lfloor p^N/2 \rfloor$, i.e., N is the smallest integer such that $-\lfloor p^{N-1}/2 \rfloor \leqq |a_n||a_0| < \lfloor p^N/2 \rfloor$.

ALGORITHM L ← IPRZL(A).

[Integral polynomial rational zeros by linear lifting. A is a squarefree integral polynomial, L is a list of its rational zeros.]

(1) [Suitable prime.] Let p be the smallest prime, such that $A \bmod p$ is squarefree.

(2) [Bound.] Set $b = 2|a_n||a_0|$ and choose the smallest N with $b < p^N$.

(3) [Zeros over Z_p.] Compute a list L^* of the zeros $(Aa_n^{-1}) \bmod p$ by exhaustive search.

(4) [Linear lifting.] According to Theorem 1 lift the zeros in L^* of $(Aa_n^{-1}) \bmod p$ to zeros (of Aa_n^{-1}) $\bmod p^N$.

(5) [Adjust for leading coefficient and test.] For each α^* in L^* reduce $(\alpha^* a_n \bmod p^N)/(a_n \bmod p^N)$ to lowest terms s/t and, if $A(s/t)=0$, include s/t in L.

4. Analysis of the linear Newton–Hensel algorithm.

THEOREM 4. *The maximum computing time of* IPRZL *is* $O(n^3 L(d)^3)$.

Proof. In step 1 we compute $A \bmod p$ and $A' \bmod p$ in $O(nL(d))$ steps and form the $gcd(A, A')$ over Z_p using $O(n^2)$ steps at most. By Theorem 2 at most $O(nL(nd)) = O(nL(d))$ primes have to be tested, n considered single precision. Hence $t_1 = O(\{nL(d)+n^2\}nL(d)) = O(n^3 L(d) + n^2 L(d)^2)$.

In step 2 $L(p^N)$ is bounded by $L(b)$ and $N = O(L(d))$. Hence $t_2 = O(L(d)^2 + N^2 L(p)^2) = O(L(d)^2)$ for a single precision prime p.

In step 3 we form $A \bmod p$ at $O(nL(d))$ cost, $A^* \equiv (Aa_n^{-1}) / \bmod p$ at $O(n)$ cost. Then for the prime $p = O(n(L(d))$ according to Theorem 2 we test p field elements α^* of Z_p for $A^*(\alpha^*) \equiv 0 \bmod p$ at $O(n)$ cost. Therefore $t_3 = O(n^2 L(d))$ at most.

In step 4 α^* is bounded during all lifting steps by $O(L(d))$. $A^* \equiv Aa_n^{-1} \bmod p^N$ is computed N times at $O(nL(d))$ cost. Then $A^*(\alpha^*)$ is N times evaluated at $O(n^2 L(d)^2)$ cost and there are at most n α^*'s. Therefore $t_4 = O(n \cdot N \cdot n^2 L(d)^2 L(d)^2) = O(n^3 L(d)^3)$.

In step 5 for at most n α^*'s s/t is computed with $O(L(d)^2)$ cost and then $A(s/t)$ is tested with $O(n^2 L(d)^2)$ cost. Therefore $t_5 = O(n^3 L(d)^2)$.

Clearly, the computing time of the lifting step dominates the overall computing time. □

Empirically it turns out that improvements in step 1, 2, 3 or 5 are rather marginal since by far the most time is spent in the linear lifting process in step 4. In order to compare the asymptotic analysis with the actual computing time used we made a time profile $t_1, \cdots, t_5$ for steps 1–5 of the algorithm. We used polynomials of degree 4 having 1 rational zero with maximum bit length of the random coefficients of 9 and 116, corresponding approximately to 3 and 35 decimal digits. The results are given in Table 2 in seconds.

TABLE 2
Time profile of steps 1–5 of the linear Newton–Hensel algorithm.

Bits	t_1	t_2	t_3	t_4	t_5	Total time [s]
9	.068	.002	.0072	.125	.033	.300
9	22%	1%	24%	42%	11%	100%
116	.056	.131	.072	20.228	.290	20.78
116	0%	1%	0%	97%	2%	100%

Table 2 verifies the theoretical computing time analysis and indicates that an improvement can only be expected if the lifting process can be improved.

5. The quadratic Newton–Hensel algorithm. The results of the empirical and theoretical analyses suggest reconsideration of Theorem 1 for improvement. In the real Newton approximation method there is a quadratic version besides the linear version, of which Theorem 5 states the p-adic analogue.

THEOREM 5. *Let $A(x)$ be an integral polynomial and let α be a zero of $A(x) \bmod p$ which is not a zero of the derivative $A'(x) \equiv 0 \bmod p$, p a prime. Then for each $r \geqq 1$ the congruence $A(x) \equiv 0 \bmod p^{2^r}$ has a solution $\alpha_{2^r} \equiv \alpha \bmod p$.*

Proof. The proof of Theorem 1 carries over. Set $\alpha_{2N} = \alpha_N + p^N y$, with indeterminate y and $b_N = A(\alpha_N)/p^N$. Then

$$\begin{aligned} A(\alpha_{2N}) &\equiv A(\alpha_N) + p^N y A'(\alpha_N) \bmod p^{2N} \\ &\equiv p^N (b_N + y A'(\alpha_N)) \bmod p^{2N}. \end{aligned}$$

Since $A'(\alpha_N) \not\equiv 0 \bmod p^N$ holds, if $A'(\alpha) \not\equiv 0 \bmod p$ by assumption, y can be set to $(-b_N/A')(\alpha_N)) \bmod p^N$ and $A(\alpha_{2N}) \equiv 0 \bmod p^2 N$ holds. □

The quadratic Newton–Hensel algorithm IPRZ differs from IPRZL only in step 4 by expanding the monic $A \bmod p$ to $A \bmod p^2, \cdots, A \bmod p^{2^i}$ and the monic derivative $A' \bmod p$ to $A' \bmod p^2, \cdots, A' \bmod p^{2^i}$, where i is the smallest integer such that $2^i \geqq N$. In fact, if N is not a power of 2 the last modulus can be replaced by p^N, resulting in a nontrivial improvement in the most expensive last step of the expansion. We have now fewer expansion steps, but expand A' in addition to A. Nevertheless the net effect is a gain as shown by:

THEOREM 6. *The maximum computing time of the quadratic Newton–Hensel algorithm for finding all rational zeros is $O(n^3 L(d)^2)$, where n is the degree and d the seminorm of the polynomial A.*

Proof. The only change compared to the linear algorithm occurs in step 4. The expansion of the derivative changes the constants involved, but not the asymptotic time. However, we have only $i \leqq \log_2 N + 1$ expansion steps compared to N in IPRZL, where $N = O(L(d))$. Since $L(L(d)) = O(1)$ t_4 drops to $O(n^3 L(d)^2)$, as does also the total time. □

In Table 3 the time profiles of Table 2 are repeated with the same polynomials for the new algorithm.

TABLE 3
Time profile of steps 1–5 of the quadratic Newton–Hensel algorithm.

Bits	t_1	t_2	t_3	t_4	t_5	Total time [s]
9	.068	.002	.072	.116	.033	.291
9	23%	1%	25%	40%	11%	100%
116	.055	.130	.072	4.163	.327	4.474
116	1%	3%	2%	88%	6%	100%

The gain for the polynomial with the small coefficients is small, but increases to a factor of 4–5 for the larger coefficients.

Since the gain occurs asymptotically in $L(d)$ in Table 4 we compare polynomials with one rational zero of degree 4 for varying coefficient size, indicating that the quadratic version is always preferable also from a practical point of view.

TABLE 4
Comparison of computing times of the linear and quadratic Newton–Hensel algorithms on a Burroughs 7700 for varying coefficient size $L(d)$.

Bits	Time Linear lifting	Time [s] Quadratic lifting
9	.30	.29
19	.46	.38
29	1.62	.92
38	2.68	.96
48	2.67	.78
58	3.62	1.00
67	4.69	1.07
77	10.7	1.72
87	6.74	1.36
96	9.93	2.10
106	17.3	4.73
116	20.8	4.74

Finally, in Table 5 we study the influence of the degree on the empirical times of both algorithms for polynomials with one rational zero and random coefficients of 29 bits.

TABLE 5
Comparison of computing times of the linear and quadratic Newton–Hensel algorithms on a Burroughs 7700 for varying degree n.

Degree	Time Linear lifting	Time [s] Quadratic lifting
4	1.86	.86
8	2.93	.92
16	5.43	1.77
32	16.4	5.13
64	29.1	10.7

6. Generalizations. Let us call the field $\{r+si | r, s \in Q\}$ the Gaussian rational numbers. The subring of Gaussian integers is a unique factorization domain. Let $A(z)$ be a squarefree polynomial with Gaussian integer coefficients. In order to find a Gaussian rational zero of $A(z)$ we generalize the previous algorithm. We choose a rational prime p which is also a Gaussian prime and does not divide the resultant of A and A'. The exhaustive search is performed with $r+si$, $0 \leqq r, s < p$, yielding $p^2 = O(n^2 L(d)^2)$ tests, each of cost $O(n)$, therefore with $O(n^3 L(d)^2)$ total cost.

Theorem 1 has been generalized by Lauer [10] (this issue, pp. 395–410) and can be applied to the Gaussian case, and the same bounds hold if we replace the absolute value by the norm operation. During the quadratic lifting process, polynomial

arithmetic with Gaussian integer coefficients is performed with computing times codominant to the integer coefficient case. We arrive therefore at

THEOREM 7. *Let* $A(z)=\sum_{j=0}^{n}(r_j+s_ji)z^j$ *be a squarefree polynomial with Gaussian integer coefficients. Let* $d=\sum_{j=0}^{n}(|r_j|+|s_j|)$ *be the seminorm of* A. *Then all Gaussian rational zeros of* A *can be computed in* $O(n^3L(d)^2)$ *time.*

Another generalization consists in computing the p^k test factors $x^k+a_{k-1}x^{k-1}+\cdots+a_0$ over $GF(p)[x]$ with $1\leqq k\leqq\lfloor n/2\rfloor$. However, if $p>n$, a situation which is possible since $p=O(nL(d))$, then it is preferable to compute a complete factorization with Berlekamp's algorithm, which results in at most n^k polynomials of degree k to be lifted and tested. In this way we arrive at $O(n^{k+4}L(d)^2)$ as maximum computing time to compute all factors of degree k of an integral polynomial.

7. Common shifted factors. Let $A(x)=A_1(x)A_2(x)$ be an integral polynomial with nontrivial factors A_1 and A_2, and similarly $B(x)=B_1(x)B_2(x)$. If there exists a rational number r such that $B_1(x)=A_1(x+r)$ then B_1 (or A_1) is called a *common shifted factor* of A and B with shift r. In Gosper's [3] and Karr's [9] algorithms for indefinite summation it is necessary to compute all common shifted factors of two polynomials. The following algorithm computes all common shifted factors of two polynomials A and B.

ALGORITHM

(1) [Resultant.] Compute $C(x)=\operatorname{res}_y(A(y),B(x+y))$.

(2) [Rational zeros.] Let $(r_1,\ldots,r_k)$ be a list of all rational zeros of C.

(3) [Common shifted factors.] For each r_i compute $D_i(x)=gcd(A(x+r_i),B(x))$. The D_i are the common shifted factors of A and B.

THEOREM 8. *The algorithm computes all common shifted factors of* A *and* B.

Proof. In order to prove the correctness of the algorithm, we apply the resultant identity $\operatorname{res}_x(A(x),B(x))=a_m^n\prod_{i=1}^{m}B(\alpha_i)$, where a_m is the leading coefficient of the polynomial A with m complex zeros α_i and n is the degree of B. If B has leading coefficient b_n we get $C(x)=a_m^n\prod_{i=1}^{m}B(x+\alpha_i)=a_m^n\prod_{i=1}^{m}b_n\prod_{j=1}^{n}(x+\alpha_i-\beta_j)=a_m^nb_n^m\prod_{i,j}(x-(\beta_j-\alpha_i))$. Therefore, $C(x)$ has the zeros $\beta_j-\alpha_i$, $1\leqq j\leqq n$, $1\leqq i\leqq m$. Now, let $r_k=r_{ij}=\beta_j-\alpha_i$ be a rational zero of $C(x)$ as it is computed in step (2). Therefore $A(\alpha_i)=A(\beta_j+r_{ij})=B(\beta_j)=0$ and β_j is a common zero of $A(x+r_{ij})$ and $B(x)$ which implies a nontrivial greatest common divisor D_{ij} of $A(x+r_{ij})$ and $B(x)$. □

THEOREM 9. *The maximum computing time for Algorithm* 7.1 *is* $O(n^{10}+n^8L(d)^2)$, *where* $n=\max(\deg A,\deg B)$ *and* $d=\max(d_A,d_B)$.

Proof. To compute $B(x+y)$ by Taylor's expansion needs $O(n^2(n+L(d)))$ steps, $B(x+y)$ has at most degree n in x and y and $L(d_{B(x+y)})=O(n+L(d))$. The computation of C can be done in $O(n^{2\cdot2+1}(n+L(d))^2)=O(n^6+n^5L(d)+n^4L(d)^2)$ steps. C is of degree n^2 at most and $L(d_C)=O(n(n+L(d)))$.

In step (2) we again use Theorem 6 and get $O(n^{2\cdot3}(n(n+L(d))^2)=O(n^{10}+n^8L(d)^2)$. We have at most n^2 rational zeros, each $r=s/t$ has a length $L(r)=L(s)+L(t)$ of at most $O(n(n+L(d)))$.

In step (3) we compute $A(x+r)$ by Taylor's expansion and evaluation with Horner's rule. This needs

$$\begin{aligned}&O(n^2L(r)^2+n^2L(d)L(r))\\&\quad=O(n^2(n(n+L(d)))^2+n^2L(d)n(n+L(d)))\\&\quad=O(n^6+n^4L(d)^2)\end{aligned}$$

steps. $A(x+r)$ has a degree n at most and $L(d_{A(x+r)}) = O(L(d)+nL(r)) = O(n^2(n+L(d)))$. The cost of a *gcd* calculation of two polynomials of maximum degree m and seminorm e is $O(m^3L(e)+m^2L(e)^2)$ using a modular algorithm [4]. To compute $D(x)$ costs at most $O(n^3(n^2(n+L(d)))+n^2(n^2(n+L(d)))^2) = O(n^6+n^4L(d)^2)$. Finally, the total cost of step 4 is $t_4 = n^2O(n^6+n^4L(d)^2) = O(n^8+n^6L(d)^2)$.

The maximum computing time is dominated by the time for the rational zero calculation in step 2. □

By another resultant calculation in step (1), also "rotated" factors of A and B can be detected such that $\alpha_i = r\beta_j$, for some rational r, and in general, for any Gaussian rationals a, b, c, d factors with $\alpha_i = (a\beta_j+b)/(c\beta_j+d)$ can be found.

Acknowledgment. The author thanks G. Collins and G. Goos for improvements of an earlier draft of this paper.

REFERENCES

[1] G. E. COLLINS, *Infallible calculation of polynomial zeros to specified precision*, in Proc. Math. Software Conference, Madison, WI, March 1977.

[2] D. R. MUSSER, *Multivariate polynomial factorization*, J. Assoc. Comput. Mach., 22 (1975), pp. 291–308.

[3] R. W. GOSPER, JR., *Decision procedure for indefinite hypergeometric summation*, Proc. Nat. Acad. Sci. U.S.A., 75 (1978), pp. 40–42.

[4] G. E. COLLINS, *Computer algebra of polynomials and rational functions*, Amer. Math. Monthly, 80 (1973), pp. 725–755.

[5] N. M. GÜNTER AND R. O. KUSMIN, *Aufgabensammlung zur Höheren Mathematik*, I, Translation from the Russian, Berlin, 1968.

[6] D. J. LEWIS, *Diophantine equations: p-adic methods* in Studies in Number Theory, W. J. LeVeque, ed., Math. Assoc. Amer., 1969, pp. 25–75.

[7] D. E. KNUTH, *The Art of Computer Programming, Vol.* II, *Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.

[8] R. LOOS, *Computing rational zeros of integral polynomials by p-adic expansion*, Bericht 11/78, Fakultät für Informatik, Universität Karlsruhe, Dec. 1978.

[9] M. KARR, *Summation in finite terms*, J. Assoc. Comput. Mach., 28 (1981), pp. 305–350.

[10] M. LAUER, *Generalized p-adic constructions*, this Journal, this issue, pp. 395–410.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0006 $01.25/0

MULTIPROCESSOR SCHEDULING OF UNIT-TIME JOBS WITH ARBITRARY RELEASE TIMES AND DEADLINES*

BARBARA SIMONS†

Abstract. We present a polynomial time algorithm for constructing an optimal schedule, if a feasible schedule exists, for tthe following multimachine scheduling problem. There are n unit-time jobs, with arbitrary release times and deadlines, and m identical parallel machines. A feasible schedule is one in which no job is started before it is released, each job is completed by its deadline, and no job is interrupted once it begins to run.

Key words. scheduling, release time, deadline, computational complexity

1. Introduction. The problem we shall discuss is that of scheduling n unit-time jobs, with release times and deadlines which are arbitrary rational numbers, on m identical parallel machines. If we relax the constraints by allowing different jobs to have different running times, then the problem becomes strongly NP-complete, even in the one-machine case [7]. The problem remains NP-complete even if only two different values for the release times and two different values for the deadlines are allowed [3].

In general we shall assume that there is no partial order. If. however, there is only one machine, then by modifying the release times and deadlines to reflect the partial order, it is possible to produce a feasible schedule with minimum completion time which does not violate the partial order, if one exists [5]. For the two-machine case with integer release times, arbitrary deadlines and a partial order, there is a polynomial time algorithm [4], and for an arbitrary number of machines the problem is NP-complete even with all $r(i)=0$ and all $d(i)=D$ [10].

A polynomial time algorithm with time complexity $O(n^2 \log n)$ was first obtained for the single machine unit-running-time case by Simons [9]. An alternative algorithm with the same time complexity was subsequently obtained by Carlier [2]. Finally, an algorithm with time complexity $O(n \log n)$ for the single machine case was presented by Garey, Johnson, Simons and Tarjan [5]. Since the problem of sorting can be reduced to the problem of producing a feasible schedule, this algorithm is optimal to within a constant factor in the order of the number of comparisons.

In the multimachine case, the best previously known result was produced by Carlier, whose algorithm runs in $O(n^{m+2} \log n)$, where m is the number of machines [2]. However, the question of whether an algorithm exists which solves the problem in time and which is polynomial in *both* m and n remained unanswered and was presented as an open problem in the paper by Garey, Johnson, Simons and Tarjan [5]. We describe below an algorithm with running $O(n^3 \log \log n)$ which schedules all the jobs while minimizing both the maximum completion time and the sum of all the completion times.

2. Definitions. We shall assume that there is a set of n *jobs*: $J(1), J(2), \cdots, J(n)$ and m *processors* or *machines*: $M(0), M(1), \cdots, M(m-1)$. Each job $J(i)$ has a *release time*, $r(i)$, a *deadline*, $d(i)$ and unit processing time. When we speak of a job $J(i)$ being *released* by time t, we mean that $r(i) \leqq t$. The *start time*, $s(i)$ of job $J(i)$ is the time at which $J(i)$ is scheduled to begin in a given schedule. If at some time t there is no job scheduled to be running on a given processor, then we say that the time between the completion of the last job to be scheduled before t and the first job to be scheduled after t on that processor is called *idle time*.

* Received by the editors January 4, 1982, and in revised form June 24, 1982.
† IBM Research K51-1BN, San Jose, California 95193.

One normally thinks of a *schedule* as being an assignment for each job $J(i)$ of a start time, $s(i)$ and a machine, $h(i)$. More formally, a *feasible schedule* is a mapping $g:\{1,\cdots,n\}\to Q^+\times\{1,\cdots,m\}$, where Q^+ is the nonnegative rationals, and where $g(i)=(s(i),h(i))$, $1\leqq i\leqq n$, such that:

1. $r(i)\leqq s(i)\leqq d(i)-1$ (every job is run between its release time and its deadline),

2. if $h(i)=h(j)$ for $i\neq j$, then $s(i)\leqq s(j)\Rightarrow s(j)-s(i)\geqq 1$ (each machine can run at most one job at a time).

If there are some jobs which have not been assigned start times, then we have a *partial schedule*. Finally, in none of the problems will *preemption* be allowed, that is, once a job has begun execution it cannot be interrupted and consequently must run until it is completed.

The scheduling techniques used in the following lemma is the basis for the approach we use in scheduling on parallel machines. It is also interesting to note the correspondence between schedules and permutations.

LEMMA 1. *An arbitrary feasible schedule for a set of n jobs with release times, deadlines, and unit processing times can be mapped into a unique permutation. Conversely, given a permutation obtained as described below from a feasible schedule, a unique feasible schedule can be constructed. Futhermore, each job in the constructed schedule will finish no later than it did in the original schedule.*

Proof. Suppose we are given a feasible schedule for a set of n jobs as in the lemma. From this schedule we derive a permutation by representing the job with the minimum start time as the first element in the permutation, the job with the next smallest start time as the second, and so on. In the case of a tie, the job on the machine with the smallest index is chosen first.

From a permutation so obtained we construct a feasible schedule in the following circular fashion. The ith job is scheduled on the machine i mod m and has as its start time either the completion time of the job most recently scheduled on machine i mod m or its release time, whichever is larger. Clearly the first job to be scheduled in the permutation schedule starts no later than the time at which the first job in the original feasible schedule was started. By induction, the ith job scheduled in the permutation schedule starts no later than the ith job in the original schedule. Since all of the jobs are completed by their deadlines in the original schedule, they will all be completed by their deadlines in the permutation schedule. By definition, none of the jobs in the permutation schedule is begun before its release time. So the schedule obtained from the permutation is indeed a feasible schedule. □

3. Strategy. Given that the algorithm already has constructed a partial feasible schedule, the schedule is extended as follows. The earliest time, say t, at which a new job can be scheduled is determined and the next job to be scheduled is selected. The job selection is done by choosing the unscheduled job with the earliest deadline which has been released by time t. This technique was first described in [6].

If a job, say $J(i)$, is discovered which cannot be completed by its deadline, then a call is made to the *crisis* subroutine. The crisis subroutine determines a point in the schedule, say t, at which it is necessary to delay scheduling some job if $J(i)$ is to be completed by its deadline. Then all the jobs which have been examined by the crisis subroutine are removed from the partial schedule. The information obtained from the crisis subroutine is incorporated into the partial schedule, and the process continues from the smaller partial schedule.

The algorithm creates a schedule with the minimum possible completion time if one exists, or determines that the problem instance has no feasible schedule otherwise.

Note that the problem in which all the jobs have the same running time, say p, can be made into a version of our problem by dividing all release times, deadlines and running times by p.

We shall first illustrate the algorithm by example, beginning with the following two machine examples. The release time is the first number in the ordered pair and the deadline is the second. So A has a release time of 0 and a deadline of 2:

$$A\colon (0, 2),\quad B\colon (0.2, 3),\quad C\colon (0.5, 2),\quad D\colon (0.5, 3).$$

In the example, which is illustrated in Fig. 1*a*, we begin scheduling starting at time 0 on machine 0. Since A is the only job which has been released at time 0, A is scheduled to begin at time 0. Machine 1 is available to run a job at time 0, but there are no remaining unscheduled jobs which have been released by time 0. The algorithm then determines the job with the minimum release time from among the unscheduled jobs. In our example, this results in job B being scheduled at its release time of 0.2 on machine 1. Now, machine 0 is available to run a job at time 1, by which time jobs C and D have both been released. Again selecting the job with the earlier deadline, C is started on machine 0 at time 1. Machine 1 is available at time 1.2, and therefore D is scheduled on machine 1 at time 1.2.

In the above example, the jobs were scheduled in one iteration with no complications. Suppose, however, that we change C's deadline from 2 to 1.9. In this case, C could not be scheduled to begin at time 1, since it could not be completed by its deadline. But if, instead of scheduling B to begin at time 0.2, we were to wait until C was released at time 0.5 and schedule C on machine 1, then we could construct a feasible schedule. This schedule is shown in Fig. 1b.

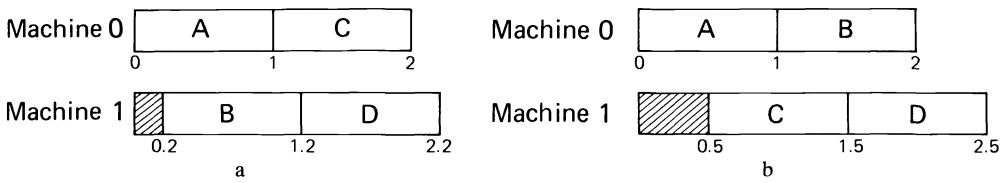


FIG. 1. *A 2-machine example.*

The basic idea behind the algorithm is that *there are times when it is necessary to wait for a crucial job to be released, even though there may be other jobs which are available to be run. But if we do start a job, it should be the available job with the earliest deadline.* This theme also appears in the single machine algorithms, although in the case of the recursive $O(n^2 \log n)$ algorithm of Simons [9], the enforced waiting time is only implicit in the algorithm.

4. The barriers algorithm. We define a v-partial schedule to be a feasible schedule of a set of v jobs, $v \leqq n$. (When the size of the v-partial schedule is not of interest, we may speak simply of the partial schedule.) The algorithm proceeds by trying to increase the current v-partial schedule to a $(v+1)$-partial schedule. The next job to be scheduled is said to occupy *slot* $v+1$. In fact, given any partial schedule obtained by any algorithm, we say that the job occupying the $(v+1)$st slot is the job which is the $(v+1)$st job scheduled to begin running. So the first job to be scheduled by the algorithm occupies slot 1, the second occupies slot 2, and so forth. A v-partial schedule can be viewed as a permutation of v elements and a slot as a location in that permutation. If at some point in the running of the algorithm the jobs are all removed

from the partial schedule, then we begin again with scheduling the first job in slot 1. Note that the job occupying a particular slot can vary from partial schedule to partial schedule. But there will be at most n slots in any partial schedule, and if the nth slot is ever filled in a feasible schedule, then we have succeeded in scheduling the entire set of jobs.

The fundamental notion making this algorithm polynomial is that of "barriers." A *barrier* is an ordered pair (j, r), where j is a slot number and r is a release time. (The manner in which barriers are created is described below.) A barrier (j, r), represents the constraint that the jth slot can begin no earlier than time r.

The *earliest deadline with barriers* (EDB) *subroutine* is called to create a $(v+1)$-partial schedule from the current v-partial schedule. It first computes h, the number of the machine on which the job in slot $v+1$ is to be scheduled, by setting $h = (v+1) \bmod m$. It then determines the "earliest allowed time" t at which slot $v+1$ can begin as follows. The function f mapping slots to start times in the current v-partial schedule is defined by setting $f(i)$ to be the start time of slot i, $1 \leqq i \leqq v$. To determine the earliest possible start time for the next slot, we set t_1 to be 0 if $v+1 \leqq m$ or $f(v+1-m)+1$ otherwise. (Slot $v+1-m$ is the slot most recently scheduled on machine $M(h)$). Set t_2 to be the minimum release time of the unscheduled jobs, and set t_3 to be $\max\{\{r\colon (j, r) \text{ is a barrier, } 1 \leqq j \leqq v+1\} \cup \{0\}\}$. Let $t = \max\{t_1, t_2, t_3\}$. We say that t is the *earliest allowed time respecting barriers* (for slot $v+1$).

The subroutine calls the earliest deadline algorithm to schedule the job to occupy slot $v+1$ with a start time of t. Suppose job $J(i)$ has been so selected. If $d(i) \geqq t+1$, then $J(i)$ is scheduled in slot $v+1$, and the subroutine returns to the main routine. Note that if $t > t_1$, there will be idle time preceding slot $v+1$.

The subroutine will *fail* if $d(i) < t+1$. In this case we say that a *crisis* has occurred and that $J(i)$ is a *crisis job*. The *crisis subroutine* is then called.

The crisis subroutine backtracks over the current partial schedule, searching for the job with the highest slot number that has a deadline greater than that of the crisis job. If it does not find such a job, it concludes that there is no feasible schedule and halts. Otherwise, the first such job which it encounters is called the *pull job*. Let j be the slot number of the pull job. The set of all jobs in the v-partial schedule with slot number $j+1$ or greater together with the crisis job called a *restricted set*. (The pull job is not a member of the restricted set.) The subroutine then determines the minimum release time of all the jobs in the restricted set, call this time r, creates a new barrier defined to be the order pair (j, r) and returns.

The main routine adds the new barrier to the previous list of barriers, removes all jobs occupying slots j through v in the partial schedule, and calls the EDB subroutine, starting with a $(j-1)$-partial schedule. The algorithm halts when either it has succeeded in creating an n-partial schedule or it discovers a crisis job for which there is no pull job. In the former case the output consists of a feasible schedule, and in the latter, a statement that there is no feasible schedule.

5. Correctness proof. A barrier (j, r) is *correct* for a problem instance if in all feasible schedules the jth slot can start no earlier than r. To demonstrate correctness of the algorithm we shall prove that each barrier is correct and that if the algorithm does not produce a feasible schedule, then none exists.

First consider the more general problem of "scheduling with barriers." This problem is identical to the originally stated problem except that it has the additional constraint of a (possibly empty) set of barriers. For each barrier (j, r) we require that slot j be started no earlier than time r.

LEMMA 2. *Assume we are given a problem instance of "scheduling with barriers" and that successive calls to the* EDB *subroutine, starting with an empty partial schedule, produce a v-partial schedule without requiring any calls to the crisis subroutine. Then there is no feasible schedule in which any of the v slots has an earlier start time. Furthermore, slot $v+1$ can be started no earlier (in any feasible schedule) than the earliest allowed time respecting barriers as computed by the algorithm.*

Proof. Assume we are given a v-partial schedule obtained as in the statement of the lemma and assume by induction that the lemma is true for all slots up to and including slot v. Let t = earliest allowed time respecting barriers for slot $v+1$. If $t = t_1$, then either $t=0$ or $t=f(v+1-m)+1$. The former case is trivially optimal. In the latter case $f(v+1-m)$ is by the induction assumption the earliest possible time at which slot $v+1-m$ can start. Since none of the slots in the partial schedule with numbers greater than $v+1-m$ can start earlier than the time at which slot $v+1-m$ starts, and since every machine has scheduled on it a slot which is numbered between $v+1-m$ and v, there is no machine available for running the $(v+1)$st slot any earlier than the machine on which slot $v+1-m$ had been scheduled. Therefore, $f(v+1-m)+1$ is the earliest possible time for starting slot $v+1$. If $t=t_2$, then slot $v+1$ is started at the minimum release time of the unscheduled jobs. Consequently, there are only v jobs with release times less than or equal to t_2. So there is no feasible schedule in which slot $v+1$ can start earlier than t_2. If $t=t_3$ and $t\neq 0$, then a barrier constraint is preventing slot $v+1$ from beginning earlier. □

LEMMA 3. *The jobs in the restricted set all have release times strictly greater than the start time of the pull job.*

Proof. The pull job has a deadline strictly greater than that of any of the jobs in the restricted set. Therefore, if any of them had been released by the time at which the pull job was scheduled, it would have been scheduled at that time. □

THEOREM 1. *Each barrier that the algorithm creates is correct.*

Proof. Assume the first $k-1$ barriers are correct, that the algorithm has constructed a v-partial schedule at the time of the kth crisis, and that (j, r) is the kth barrier to be created. Suppose to the contrary that there is a feasible schedule in which the jth slot is scheduled before time r. Since r is the minimum release time of all the jobs in the restricted set, by the pigeon-hole principle some job in the restricted set must occupy a slot $\geqq v+1$. By applying Lemma 2 to the "scheduling with barriers" problem obtained using the first $k-1$ barriers, it follows that slot $v+1$ can be started no earlier than the earliest allowed time respecting barriers computed by the EDB subroutine. Since all of the jobs in the restricted set have deadlines less than or equal to that of the crisis job, and since the completion time of slot $v+1$ must exceed the deadline of the crisis job in any feasible schedule, none of the jobs in the restricted set can be scheduled in slot $v+1$. Hence in no feasible schedule can slot j begin before time r. □

LEMMA 4. *If the crisis subroutine does not find a pull job, then there is no feasible schedule.*

Proof. Let $J(i)$ be a crisis job for which no pull job is found. It follows from the definition of a pull job that all jobs occupying slots 1 through v in the current v-partial schedule have deadlines no greater than $d(i)$, the deadline of the current crisis job. Therefore, they must all be completed by $d(i)$. Using the same argument as in Lemma 2, slot $v+1$ can begin no earlier than the time computed by the EDB subroutine. Consequently, any job occupying slot $v+1$ will have a completion time greater than $d(i)$. So either the crisis job or some job which had been scheduled in the v-partial schedule will fail to meet its deadline. □

Since the only way in which the barriers algorithm fails to produce a schedule is if the crisis subroutine does not find a pull job, we have the following theorem.

THEROREM 2. *If the barriers algorithm fails to produce a schedule, then there is no feasible schedule.*

We also note that by Lemma 2 if the algorithm produces a schedule, then each slot in the schedule, in particular the last, is started at the earliest possible time.

THEOREM 3. *If the barriers algorithm produces a schedule, then no feasible schedule has an earlier completion time. Furthermore, the schedule produced by the barriers algorithm minimizes the sum of the completion times.*

6. Complexity.

THEOREM 4. *The algorithm runs in time* $O(n^3 \log\log n)$.

Proof. There are at most n distinct release times and n slots. So n^2 is an upper bound on the number of barriers which the algorithm can create. If one uses a standard priority queue, the cost of scheduling each job is $O(\log n)$, but by using a stratified binary tree the cost is only $O(\log\log n)$ [1]. Since at most n jobs can be scheduled before a new barrier is created, we get the running time of $O(n^3 \log\log n)$. □

Acknowledgment. I would like to thank Ron Fagin for having generously given of his time as I struggled through the details of the correctness proof.

REFERENCES

[1] P. V. E. BOAS, *Preserving order in a forest in less than logarithmic time*, 16th Annual Symposium of Foundations of Computer Science, IEEE Computer Society, Long Beach, CA, 1978, pp. 75–84.

[2] J. CARLIER, *Problème à une machine dans le cas où les tâches ont des durées égales*, Technical Report, Institut de Programmation, Université Paris VI, 1979.

[3] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*, W. H. Freeman, San Francisco, 1978.

[4] ——— *Two-processor scheduling with start-times and deadlines*, this Journal, 6 (1977), pp. 416–426.

[5] M. R. GAREY, D. S. JOHNSON, B. B. SIMONS AND R. E. TARJAN, *Scheduling unit-time tasks with arbitrary release times and deadlines*, this Journal, 10 (1981), pp. 256–269.

[6] J. R. JACKSON, *Scheduling a production line to minimize maximum tardiness*, Research Report 43, Management Science Research Project, Univ. of California, Los Angeles, 1955.

[7] J. K. LENSTRA, A. G. H. RINNOOY KAN AND P. BRUCKER, *Complexity of machine scheduling problems*, Ann. Discrete Math., 1 (1977), pp. 343–362.

[8] B. B. SIMONS, *A fast algorithm for multiprocessor scheduling*, 21st Annual Symposium of Foundations of Computer Science, IEEE Computer Society, Long Beach, CA, 1980, pp. 50–53.

[9] ——— *A fast algorithm for single processor scheduling*, 19th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Long Beach, CA, 1978, pp. 246–252.

[10] J. D. ULLMAN, NP-*complete scheduling problems*, J. Comput. System Sci., 10 (1975), pp. 384–393.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0007 $01.25/0

PROPERTIES OF FINITE AND PUSHDOWN TRANSDUCERS*

CHRISTIAN CHOFFRUT† AND KAREL CULIK II‡

Abstract. We consider the subfamilies of rational and pushdown transducers and corresponding translations (relations) which are most frequently encountered in the literature. We survey some of the known results on the characterization, factorization, closure properties, decision problems and comparisons of classes and give new results on these properties using either direct proofs or results from other theories such as homomorphic equivalence.

Key words. classes of transducers, rational relations, pushdown translations, generalized sequential machines, sequential transducers, subsequential functions, equivalence problem

1. Introduction. One of the most natural ways of defining a *relation*, or *translation*, or *transduction*, of a free monoid Σ^* into another free monoid Δ^* is to use some type of *transducer*, i.e. some type of automaton provided with outputs. At each step of the recognition procedure of a word $w \in \Sigma^*$, one of the available outputs in Δ^* is chosen. The concatenation of these outputs in the order they are produced during the computation defines a word $z \in \Delta^*$ which is in relation with w. Thus, finite automata yield the notion of *rational relation* (also known as *rational transduction* or *finite translation*), while more generally, pushdown automata define *pushdown translations* (*relations*) (cf. e.g. [1, p. 228]).

Rational relations are definitely the best known family of relations, due to their nice properties and to the role they play in the AFL theory where they prove to be a useful tool for studying context-free languages (cf. e.g. [3] and [14]). Unfortunately, this very success has probably caused the cessation of their study for their own sake (with some very few exceptions, see e.g. [18] and [19]). In our opinion, the next step towards a better knowledge of rational relations should be a systematical study of the "simplest" subfamilies of rational relations: rational partial functions, sequential and subsequential partial functions.

As far as pushdown translations are concerned, they have received less attention than they deserve. Indeed, since their general properties have been established, very little work has been done. Yet, apart from providing a model of compilation (cf. [1]), and, from a strictly mathematical point of view, posing some challenging problems, ignoring these translations would exclude some natural functions such as the function which reverses words, and the characteristic function of context-free languages.

In this paper we consider the subfamilies of pushdown relations which are most frequently encountered in the literature. Since the authors assume the reader is familiar with rational relations, the emphasis will be put on other subfamilies of relations. We shall survey the properties which are known so far—characterizations, factorizations, closure properties, equivalence decision problems—and give some new properties using either direct proofs or results from other theories, such as homomorphic equivalence (cf. [8]).

The definitions of the different families are given in the preliminaries in terms of transducers. Yet, when it comes to proving results, one wishes to use an alternative more algebraic definition. Such a characterization was first stated for rational relations

* Received by the editors February 18, 1981, and in revised form May 28, 1982. This research was supported by the Natural Sciences and Engineering Research Council of Canada under the grant No. A7403.

† Laboratoire d'Informatique Théorique et Programmation, Université Paris VII, Paris, France.

‡ Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

by Nivat [17], and is easily extended to pushdown relations (translations). It uses the notion of "bimorphism" and has given way to what in Eilenberg's terminology [9] is called "the first factorization theorem". In § 3, we establish a similar characterization for unambiguous pushdown functions, which helps in proving results in the following sections.

Section 4 is concerned with properties of the closure under composition. We consider more specifically subfamilies of pushdown functions. Our results enable us to define a hierarchy of these subfamilies.

Finally, equivalence decision problems are considered in § 5. Some new results are shown.

2. Preliminaries.

2.1. Free monoid relations. We denote by Σ^* the free monoid generated by the finite nonempty set (or *alphabet*) Σ, by ε its unit or *empty* word and by $\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$ the free semigroup. The *length* of a word $w \in \Sigma^*$ is denoted by $|w|$. Let $w = a_1 \cdots a_n$ be a word, where $n > 0$ and $a_1, \cdots, a_n \in \Sigma$. Then the *reverse* of w is the word $w^R = a_n \cdots a_1$. We define the reverse of the empty word to be the empty word.

In order to simplify notation, all relations considered in this paper, unless otherwise stated, are from the fixed free monoid Σ^* into the fixed free monoid Δ^*. We write such a relation as $f : \Sigma^* \to \Delta^*$ and view it as a function of Σ^* into the power set 2^{Δ^*}. It is *length-preserving* if $v \in f(u)$ implies $|u| = |v|$. If $f(u)$ possesses at most one element for each $u \in \Sigma^*$, we say that f is a *partial function.*

The *domain* of a relation $f : \Sigma^* \to \Delta^*$, denoted by Dom f is the subset of Σ^* defined by: $\{u \in \Sigma^* | f(u) \neq \phi\}$. Its *image* is the subset $\text{Im}\, f = \{v \in \Delta^* | v \in f(u)$ for some $u \in \Sigma^*\}$. Its *graph* is the subset $\# f = \{(u, v) \in \Sigma^* \times \Delta^* | v \in f(u)\}$. Finally the *reverse* of f is the relation $f^R : \Sigma^* \to \Delta^*$ whose graph is $\# f^R = \{(u^R, v^R) \in \Sigma^* \times \Delta^* | (u, v) \in \# f\}$.

Given a family $\mathscr{F}$ of relations, $\mathscr{F}^R$ denotes the family of all f^R where $f \in \mathscr{F}$.

The union of a family of relations $f_i : \Sigma^* \to \Delta^* (i \in I)$ is the relation $f : \Sigma^* \to \Delta^*$ defined by: $\# f = \bigcup_{i \in I} \# f_i$.

2.2. Pushdown relations. We refer to [1, p. 228] for all notions not explicitly defined in the sequel. We recall that a *pushdown transducer*, abbreviated a PDT, is an 8-tuple $\mathscr{T} = (Q, \Sigma, \Delta, X, \delta, q_0, Z_0, F)$ where:

- Q is the set of *states*, $q_0 \in Q$ is the *initial* state and $F \subseteq Q$ is the subset of *final* states;
- Σ is the *input* alphabet;
- Δ is the *output* alphabet;
- X is the *pushdown* alphabet and $Z_0 \in X$ is the *start* symbol;
- δ is a function which maps $Q \times (\Sigma \cup \{\varepsilon\}) \times X$ into finite subsets of $Q \times X^* \times \Delta^*$.

A relation $f : \Sigma^* \to \Delta^*$ is a *pushdown relation* (translation), abbreviated PD relation, if it is *defined* by some PDT.

We denote by PDR the family of all pushdown relations and by PDF the family of all pushdown partial functions.

Let $f : \Sigma^* \to \Delta^*$ be a partial function. Then it is *unambiguous* if it can be defined by some PDT whose underlying pushdown automaton is unambiguous (cf. e.g. [15, p. 142]). It is *left deterministic* (or simply *deterministic*) if it can be defined by some PDT whose underlying pushdown automaton is deterministic (cf. e.g. [15, p. 139]). It is *right deterministic* if the partial function f^R is deterministic. Finally it is *bideterministic* if it is both left and right deterministic.

We denote by UPDF, DET, DET^R and BIDET respectively, the families of unambiguous, deterministic, right deterministic and bideterministic partial functions.

Let $L \subseteq \Sigma^*$ be a context-free language recognized by the pushdown automaton $A = (Q, \Sigma, X, \delta, q_0, Z_0, F)$. We denote by $I_L : \Sigma^* \to \Sigma^*$ the restriction of identity to L, i.e., the relation whose graph is $\{(u, u) \in \Sigma^* \times \Sigma^* | u \in L\}$. The restriction of identity to L is a pushdown function since it is defined by the PDT $\tau = (Q, \Sigma, \Sigma, X, \delta', q_0, Z_0, F)$ where: $(p, u, a) \in \delta'(q, a, v)$ iff $(p, u) \in \delta(q, a, v)$.

An important subfamily of PDR is the family RAT of all *rational* relations, i.e. of all relations $f : \Sigma^* \to \Delta^*$ whose graph is a rational subset of the product monoid $\Sigma^* \times \Delta^*$ (cf. e.g. [9, p. 236]). The family of rational partial functions is denoted by RATF. The well-known facts that rational relations are particular pushdown relations and that $\mathrm{PDR} = \mathrm{PDR}^{\mathrm{R}}$, $\mathrm{PDF} = \mathrm{PDF}^{\mathrm{R}}$, $\mathrm{UPDF} = \mathrm{UPDF}^{\mathrm{R}}$, $\mathrm{RAT} = \mathrm{RAT}^{\mathrm{R}}$, $\mathrm{RATF} = \mathrm{RATF}^{\mathrm{R}}$, follows from Theorem 3.1.

2.3. Sequential and subsequential partial functions. We now turn to the crucial notion of sequential partial functions. Since there exist in the literature all kinds of "sequential" functions, we will present in detail the notion we will use which corresponds to Eilenberg's generalized sequential partial functions (cf. [9, Chap. XI]).

We will make use of the following convention. All partial functions f of a set X into the free monoid Σ^* shall be considered as a *total* (i.e. everywhere defined) function of X into the semiring 2^{Σ^*}. Therefore we have $f(u) = \varnothing$ whenever f is undefined for the value $u \in X$. Further, the product $f(u)f(v)$ equals $\varnothing$, i.e. is undefined if and only if $f(u)$ or $f(v)$ is equal to $\varnothing$.

A *sequential* transducer is a sextuple $\mathcal{S} = (Q, \Sigma, \Delta, \lambda, \theta, q_0)$ where Q, Σ, Δ and q_0 are the same as in the definition of a pushdown transducer, and where $\lambda : Q \times \Sigma \to Q$ is a partial function, called the *transition* function, and $\theta : Q \times \Sigma \to \Delta^*$ is a partial function, called the *output* function.

It is assumed that λ and θ have the same domain.

The partial functions λ and θ are extended to $Q \times \Sigma^*$ in the usual way (see e.g. [9, p. 297]). For all $q \in Q$ and $u \in \Sigma^*$ we write $q \cdot u$ and $q * u$ instead of $\lambda(q, u)$ and $\theta(q, u)$ respectively. This enables us, as long as no confusion arises, to denote $\mathcal{S}$ by the quadruple (Q, Σ, Δ, q_0), the partial functions λ and θ being understood.

A partial function $f : \Sigma^* \to \Delta^*$ is *left sequential* or simply *sequential* if there exists some sequential transducer $\mathcal{S} = (Q, \Sigma, \Delta, q_0)$ satisfying for all $u \in \Sigma^*$: $f(u) = q_0 * u$. It is *right sequential* if the reverse partial function f^R is left sequential. It is *bisequential* if it is both left and right sequential.

We shall denote by SEQ, $\mathrm{SEQ}^{\mathrm{R}}$ and BISEQ respectively the families of sequential, right sequential and bisequential partial functions.

Sequential partial functions are particular rational partial functions as it is easily seen (cf. e.g. [9, Prop. XI, 3.1]). More precisely we have the following crucial result due to Elgot and Mezei (cf. e.g. [3, Thm. IV.5.2]):

THEOREM 2.1. *A partial function $f : \Sigma^* \to \Delta^*$ is rational, if and only if there exist a finite set Γ, a left (resp. right) sequential partial function $g : \Sigma^* \to \Gamma^*$ and a right (resp. left) sequential partial function $h : \Gamma^* \to \Delta^*$ such that $f(u) = h(g(u))$ holds for all $u \in \Sigma^+$.*

Now, we compare several different "sequential" functions: Ginsburg's generalized sequential machine mappings—or GSM mappings—as defined in [13, p. 93], are sequential partial functions which are total, i.e. everywhere defined. DGSM mappings, considered by several authors (cf. e.g. [16, p. 172]), are restrictions of "Ginsburg's GSM mappings" to some arbitrary rational subset (not just the complement of a rational right ideal as for our sequential partial functions). DGSM are thus sequential transducers with a distinguished subset of "final" states.

We will show (Corollary 2.3): Assuming that every word has an endmarker, DGSM mappings are exactly the subsequential functions [5]. The latter have a simple algebraic characterization [19].

We turn now to the notion of subsequential partial function.

A *subsequential* transducer is a pair $(\mathcal{S}, \varphi)$ where $\mathcal{S} = (Q, \Sigma, \Delta, \lambda, \theta, q_0)$ is a sequential transducer and $\varphi : Q \to \Delta^*$ is a partial function (see [5, p. 109]).

A partial function $f : \Sigma^* \to \Delta^*$ is *left subsequential* or simply *subsequential* if there exists a subsequential transducer $(\mathcal{S}, \varphi)$ such that $f(u) = q_0 * u \cdot \varphi(q_0 \cdot u)$ holds for all $u \in \Sigma^+$ (notice that there exists no condition on $f(\varepsilon)$). It is *right subsequential* if the reversed partial function $f^R : \Sigma^* \to \Delta^*$ is left subsequential. It is *bisubsequential* if it is both left and right subsequential. Subsequential partial functions have been considered in [11] where they are called "augmented versions of DGSM mappings".

Intuitively a subsequential transducer is a sequential transducer capable of guessing the end of the input word. Formally we have the following connection between sequential and subsequential partial functions:

THEOREM 2.2. *Let $f : \Sigma^* \to \Delta^*$ be a partial function. Then f is a subsequential if and only if its domain $R = f^{-1}(\Delta^*)$ is a rational set and if there exist a new symbol $\$ \notin \Sigma$ and a sequential total function $g : (\Sigma \cup \{\$\})^* \to \Delta^*$ such that $f(u) = g(u\$)$ holds for all $u \in R \setminus \{\varepsilon\}$.*

Proof. Only if. Let $(\mathcal{S}, \varphi)$, where $\mathcal{S} = (Q, \Sigma, \Delta, q_0)$ be a subsequential transducer defining the subsequential partial function $f : \Sigma^* \to \Delta^*$. Let $F = \operatorname{Dom} \varphi \subseteq Q$, $Q' = Q \cup \{q_t\}$ where $q_t \notin Q$ is a new symbol and $\Sigma' = \Sigma \cup \{\$\}$. The domain $R = f^{-1}(\Delta^*)$ is rational since it consists in all words $u \in \Sigma^*$ such that $q_0 \cdot u \in F$. Define the sequential transducer $\mathcal{S}' = (Q', \Sigma', \Delta, \lambda, \theta, q_0)$ where λ and θ satisfy:

$$\lambda(q, a) = \begin{cases} q \cdot a & \text{if } a \in \Sigma, q \in Q \text{ and } q \cdot u \neq \varnothing, \\ q_t & \text{otherwise;} \end{cases}$$

$$\theta(q, a) = \begin{cases} q * a & \text{if } a \in \Sigma \text{ and } q \in Q, \\ \varphi(q) & \text{if } a = \$ \text{ and } q \in F, \\ \varepsilon & \text{otherwise.} \end{cases}$$

Let $g : \Sigma'^* \to \Delta^*$ be the sequential total function defined by $\mathcal{S}'$. Then for all $u \in R$ we have:

$$f(u) = (q_0 * u)\varphi(q_0 \cdot u) = \theta(q_0, u)\varphi(q_0 \cdot u) = \theta(q_0, u)\theta(q_0 \cdot u, \$) = \theta(q_0, u\$) = g(u\$).$$

If. Let $g : (\Sigma \cup \{\$\})^* \to \Delta^*$ be a sequential total function defined by the sequential transducer $\mathcal{S}' = (Q', \Sigma', \Delta, q_0)$ where $\Sigma' = \Sigma \cup \{\$\}$. Without loss of generality we may assume that the automaton underlying $\mathcal{S}$ recognizes the rational subset R, i.e. that there exists a subset $F \subseteq Q$ such that $R = \{u \in \Sigma^* | q_0 \cdot u \in F\}$. Denote by $\varphi : Q \to \Delta^*$ the partial function defined by:

$$\varphi(q) = \begin{cases} q * \$ & \text{if } q \in F, \\ \varnothing & \text{otherwise.} \end{cases}$$

Let $\mathcal{S} = (Q, \Sigma, \Delta, q_0)$ be the sequential transducer obtained by restriction to Σ of the transition and output functions of $\mathcal{S}'$. For all $u \in \Sigma^*$ we have:

$$f(u) = g(u\$) = (q_0 * u)(q_0 \cdot u * \$) = (q_0 * u)\varphi(q_0 \cdot u),$$

which proves that f is subsequential. □

COROLLARY 2.3. *Let* $\$ \notin \Sigma^*$ *and* $f: \Sigma^*\$ \to \Delta^*$. *Then* f *is a* DGSM *mapping if and only if* f *is a subsequential function.*

COROLLARY 2.4. *The family of* DGSM *mappings is properly between the families of total sequential functions and subsequential functions.*

We refer to [5] for a systematic study of subsequential partial functions. We shall denote by SUBSEQ, SUBSEQ^R and BISUBSEQ respectively, the families of subsequential, right subsequential and bisubsequential partial functions.

The diagram in Fig. 1 shows the various strict inclusions among the families considered in this section.

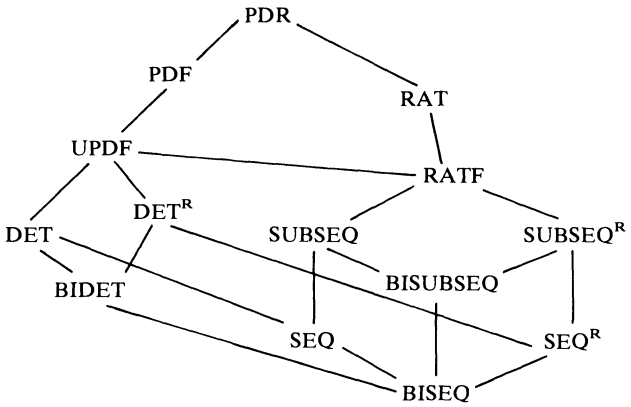


FIG. 1

3. Factorization properties.

3.1. A useful factorization of unambiguous functions. Many proofs on relations involving constructions of transducers are intricate and therefore unreliable. It is desirable to use, as much as possible, an alternative, more manageable definition. This is luckily the case for rational relations which, according to what in Eilenberg's terminology is the "First Factorization Theorem", can be obtained as a composition of simpler relations: morphisms, inverse morphisms and intersections with a rational language. A similar result is true for pushdown relations, where "context-free language" has to be substituted for "rational language". We summarize in a single statement these two well-known results (for the "rational" part of Theorem 3.1 see for instance [17] or [9, Thm. IX, 2.2], and for the "pushdown" part see [12, p. 122] or [1, Thm. 3.4]). We recall that a morphism $f: \Sigma^* \to \Delta^*$ is *alphabetic* if the image of every letter of Σ by f, is either a letter of Δ or the empty word: $f(\Sigma) \subseteq \Delta \cup \{\varepsilon\}$. It is *strictly alphabetic* if $f(\Sigma) \subseteq \Delta$.

THEOREM 3.1. *Let* $f: \Sigma^* \to \Delta^*$ *be a relation. Then it is a rational (resp. pushdown) relation if and only if there exist a finite set* Γ, *a rational (resp. context-free) language* $R \subseteq \Gamma^*$ *and two alphabetic morphisms* $h: \Gamma^* \to \Sigma^*$ *and* $g: \Gamma^* \to \Delta^*$ *such that for all* $u \in \Sigma^*$, $f(u) = g(h^{-1}(u) \cap R)$ *holds.*

When $f: \Sigma^* \to \Delta^*$ is a partial function—not any relation—the preceding proposition can be made more precise (see e.g. [9, Thm. IX, 8.1]):

THEOREM 3.2. *Let* $f: \Sigma^* \to \Delta^*$ *be a partial function. Then it is rational if and only if there exist a finite set* Γ, *a rational subset* $R \subseteq \Gamma^*$, *an alphabetic morphism* $h: \Gamma^* \to \Sigma^*$ *which maps bijectively* R *onto* $h(R)$, *and an alphabetic morphism* $g: \Gamma^* \to \Delta^*$ *such that* $f(u) = g(h^{-1}(u) \cap R)$ *holds for all* $u \in \Sigma^*$.

As a consequence, all rational partial functions are unambiguous. This is not the case for all pushdown partial functions, since the restriction of identity to an inherently ambiguous context-free language is clearly not unambiguous. However, unambiguous pushdown partial functions can be characterized in a similar way:

THEOREM 3.3. *Let $f: \Sigma^* \to \Delta^*$ be a partial function. Then it is unambiguous if and only if there exist a finite set Ξ, a bideterministic context-free language $L \subseteq \Xi^*$, an alphabetic morphism $g: \Xi^* \to \Delta^*$, and an alphabetic morphism $h: \Xi^* \to \Sigma^*$ which maps bijectively L onto $h(L)$ such that for all $u \in \Sigma^*$, $f(u) = g(h^{-1}(u) \cap L)$ holds.*

Proof. Only if. Assume we have proved that f admits the factorization

$$f = g \circ I_L \circ h^{-1} \tag{3.1}$$

where g and h are as in Theorem 3.3 and where L is a deterministic (not necessarily bideterministic) context-free language. By Chomsky–Schützenberger's theorem (see e.g. [3, Thm. 3.10]), since L is a deterministic, and therefore an unambiguous language, there exist a bideterministic language D (actually the intersection of a rational language with a Dyck language over n letters, for some integer $n > 0$) and an alphabetic morphism k which bijectively maps D over $k(D) = L$. Therefore the equality $I_L = k \circ I_D \circ k^{-1}$ holds. Substituting in (3.1) we have:

$$f = g \circ k \circ I_D \circ k^{-1} \circ h^{-1} = (g \circ k) \circ I_D \circ (h \circ k)^{-1}.$$

The conditions of the theorem are thus satisfied.

Assume now we have proved that f admits the factorization

$$f = g \circ I_L \circ h^{-1} \tag{3.2}$$

where L is deterministic, h is as in the theorem, but there is no assumption on the morphism g. We will show that f admits a factorization of the type (3.1).

First, the morphism $g: \Xi^* \to \Delta^*$ can be factorized as $g = g_2 \circ g_1$, where g_1 is an injective morphism of Ξ^* into a new free monoid Γ^*, and $g_2: \Gamma^* \to \Delta^*$ is an alphabetic morphism. Indeed letting:

$$\Gamma = \{(a, i) \in \Xi \times N \mid 1 \leqq i \leqq |g(a)| \text{ or } g(a) = \varepsilon \text{ and } i = 0\},$$

it suffices to define g_1 and g_2 by the following conditions: for all $a \in \Xi$, if $g(a) = \varepsilon$ then $g_1(a) = (a, 0)$ and $g_2(a, 0) = \varepsilon$; otherwise, if $g(a) = b_1, \cdots, b_n$ (where $n > 0$ and $b_1, \cdots, b_n \in \Delta$) then $g_1(a) = (a, 1) \cdots (a, n)$ and $g_2(a, i) = b_i$ $(1 \leqq i \leqq n)$.

Substituting $g = g_2 \circ g_1$ in (2) and observing that $g_1 \circ I_L = I_{g_1(L)} \circ g_1$, we obtain: $f = g \circ I_L \circ h^{-1} = g_2 \circ g_1 \circ I_L \circ h^{-1} = g_2 \circ I_{g_1(L)} \circ g_1 \circ h^{-1}$. Since g_1^{-1} is a DGSM mapping, then according to [16, Thm. 12.3], $g_1(L) = (g_1^{-1})^{-1}(L)$ is a deterministic language. It suffices to verify that the partial function $h \circ g_1^{-1}$ is the restriction to its domain $D = (g_1(\Xi))^* \subseteq \Gamma^*$ of an alphabetic morphism $g': \Gamma^* \to \Sigma^*$ which is injective on $g_1(L) \subseteq D$.

Let g' be the morphism defined, for all $(a, i) \in \Gamma$ by:

$$g'(a, i) = \begin{cases} \varepsilon & \text{if } i \neq 1, \\ h(a) & \text{otherwise.} \end{cases}$$

Clearly, g' is alphabetic and for all $a \in \Xi$, we have: $g' \circ g_1(a) = h(a)$. Therefore, for all $u = g_1(a_1 \cdots a_n) \in D$, we obtain: $h \circ g_1^{-1}(u) = h(a_1 \cdots a_n) = g'(u)$, which shows that $h \circ g_1^{-1}$ is the restriction of g' to D. Furthermore, if $u, v \in g_1(L)$ verify $h \circ g_1^{-1}(u) = h \circ g_1^{-1}(v)$ then $g_1^{-1}(u) = g_1^{-1}(v)$, i.e., $u = v$, which completes the verification.

Therefore, given any unambiguous partial function f, it suffices to prove that f admits a factorization of type (3.2), which we do.

Let $\tau = (Q, \Sigma, \Delta, X, \delta, q_0, Z_0, F)$ be an unambiguous pushdown transducer defining f. For each $(q, a, x, q', x', u) \in Q \times (\Sigma \cup \{\varepsilon\}) \times X \times Q \times X^* \times \Delta^*$ such that $(q', x', u) \in \delta(q, a, x)$ we define a new symbol $[q, a, x, q', x', u]$ and we denote by W the finite set thus obtained. Let $\mathscr{A} = (Q, W, X, \delta', q_0, Z_0, F)$ be the pushdown automaton defined for all $w = [q, a, x, q', x'u] \in W$, $p \in Q$ and $y \in X$ by: $(q', x') \in \delta'(p, [q, a, x, q', x', u], y)$ if and only if $p = q$ and $y = x$. Then the language L recognized by $\mathscr{A}$ is deterministic, and the morphism $h : W^* \to \Sigma^*$ which to each $[q, a, x, q', x', u]$ assigns its second component $a \in \Sigma \cup \{\varepsilon\}$ is alphabetic and injective on L since two distinct $w_1, w_2 \in L$ such that $h(w_1) = h(w_2)$ would correspond to two distinct computations of $w \in \operatorname{Dom} f$. Then it suffices to define $g : W^* \to \Delta^*$ as the morphism which to $[q, a, x, q', x', u] \in W$ assigns its last component u.

If. Since the relations $g \circ I_L \circ h^{-1}$ and $I_L \circ h^{-1}$ can be defined by two PDT having the same underlying automaton, it suffices to show that $I_L \circ h^{-1}$ is an unambiguous relation.

Let $\mathscr{A} = (Q, \Xi, X, \delta, q_0, Z_0, F)$ be a bideterministic pushdown automaton accepting L. Define a PDT $\tau = (Q, \Sigma, \Xi, X, \delta', q_0, Z_0, F)$ as follows. For all $(q, a, x, q', x') \in Q \times (\Xi \cup \{\varepsilon\}) \times X \times Q \times X^*$ we have: $(q', x', a) \in \delta'(q, h(a), x)$ iff $(q', x') \in \delta(q, a, x)$.

Clearly, $I_L \circ h^{-1} : \Sigma^* \to \Xi^*$ is the relation defined by τ. Finally, the pushdown automaton underlying τ is unambiguous since to each $w \in \Sigma^*$ corresponds a unique factorization $h(a_1 \cdots a_n) = w$ ($n > 0$ and $a_i \in \Xi$ for $1 \leqq i \leqq n$) and for such a word $a_1 \cdots a_n$ there is a unique computation in $\mathscr{A}$. This completes the proof of Theorem 3.3. □

3.2. A factorization using length-preserving partial functions. The following is equivalent, for a certain family of deterministic partial functions, to Eilenberg's [9, Thm. IX, 8.4], which states that every rational partial function is the composition of a length-preserving rational partial function and a morphism. We think it is quite unlikely that there exists a result of this type for many reasonable families of pushdown partial functions. We recall that a subset $X \subseteq \Sigma^*$ is *prefix-free* if for all $u, v \in \Sigma^*$ we have: $u, uv \in X$ implies $v = \varepsilon$.

THEOREM 3.4. *Let $f : \Sigma^* \to \Delta^*$ be a deterministic partial function whose domain is prefix-free. Then there exist a finite set Ω, a length-preserving deterministic partial function $g : \Sigma^* \to \Omega^*$ and a morphism $h : \Omega^* \to \Delta^*$ such that: $f = h \circ g$.*

Notice that the assumptions of our theorem cannot be weakened. Indeed let $\Sigma = \Delta = \{a, b\}$ and consider the deterministic partial function $f : \Sigma^* \to \Delta^*$ defined by:

$$\text{for all } r > 0, \quad n \geqq m > 0, \quad f(b^r a^n b^m) = \begin{cases} b^r a^n b^m & \text{if } m < n, \\ b^r a^n b^m a^r & \text{if } m = n; \end{cases}$$

$$\text{otherwise } f(u) = \varnothing.$$

Assume $f = h \circ g$ where $g : \Sigma^* \to \Omega^*$ is a length-preserving deterministic partial function and $h : \Omega^* \to \Delta^*$ is a morphism. Set $M = \max \{|h(x)| \mid x \in \Omega\}$, $g(b^M a^2 b) = u$ and $g(b^M a^2 b^2) = uv$ where by the hypothesis $|u| = M + 3$ and $|v| = 1$. We obtain:

$$f(b^M a^2 b^2) = h(uv) = h(u)h(v) = f(b^M a^2 b) h(v).$$

Since $f(b^M a^2 b^2) = b^M a^2 b^2 a^M$ and $f(b^M a^2 b) = b^M a^2 b$, we have $|h(v)| = M + 1$, which yields a contradiction.

Proof of Theorem 3.4. Let us note that it suffices to prove that $f = h \circ g$ where h is a morphism and g is a deterministic partial function such that $|g(u)| = 2|u|$ whenever

$u \in \mathrm{Dom}\, f$. Indeed, denote by Γ a copy of Ω^2 and by $[w_1w_2]$ the copy of $w_1w_2 \in \Omega^2$. Let $\gamma : \Gamma^* \to \Omega^*$ be the monomorphism defined by $\gamma([w_1w_2]) = w_1w_2$. Then γ^{-1} is a DGSM mapping and $\gamma \circ \gamma^{-1}$ is the identity over $(\Omega^2)^* \supseteq \mathrm{Im}\, g$. Therefore we have the factorization $f = h\gamma \circ (\gamma^{-1}g)$ where $h\gamma$ is a morphism and $\gamma^{-1}g$ is, by Theorem 4.3, a length-preserving deterministic partial function.

Let $\tau = (Q, \Sigma, \Delta, X, \delta, q_0, z_0)$ be a deterministic PDT defining f. Without loss of generality, we may assume that the underlying pushdown automaton recognizes Dom f by empty stack (cf. [15, Thm. 11.5.2]). Furthermore, we may suppose that τ satisfies the following conditions:

1) All ε-moves are erasing moves (cf. [15, Exer. 5.6.6]).
2) For all $q, p \in Q$, $a \in \Sigma$, $x \in X$, $y \in X^*$ and $u \in \Delta^*$ we have: $\delta(q, a, x) = (p, y, u)$ implies $|y| \leqq 2$ (cf. [15, Thm. 5.4.2]).
3) For all $a \in \Sigma^+$, $q \in Q$, $u \in X^*$ and $x \in X$ we have: $(q_0, a, Z_0) \overset{*}{\vdash} (q, \varepsilon, ux)$ implies $q \neq q_0$ and $x \neq Z_0$ (this can be done, if necessary, by creating a new initial state q_0', and a new start symbol Z_0'). Furthermore, for all $a \in \Sigma$, $\delta(q_0, a, Z_0)$ is of the form (q, x, u) where x is a letter.

Let Ω be the set consisting of a copy of Dom δ and of a new element σ. We shall denote by $[q, a, x]$ the copy of the element $(q, a, x) \in \mathrm{Dom}\, \delta$. The morphism $h : \Omega^* \to \Delta^*$ of the factorization we seek is defined by

$$h(\sigma) = \varepsilon,$$

and for all $q, p \in Q$, $a \in \Sigma \cup \{\varepsilon\}$, $x \in X$, and $u \in \Delta^*$,

$$h([q, a, x]) = u \quad \text{iff} \quad \delta(q, a, x) = (p, y, u)$$

for some y in X^*.

We now give an informal explanation of how the deterministic partial function $g : \Sigma^* \to \Omega^*$ assigns to each word $u \in \mathrm{Dom}\, f \subseteq \Sigma^*$ a word of length $2|u|$ containing all information about $f(u)$. A deterministic transducer τ' defining g will be obtained by modifying τ.

Assume that, at a given moment of the computation for a word $u \in \mathrm{Dom}\, f$ by the PDT τ, the current letter is $a \in \Sigma$. If this occurrence preserves the height of the stack, then the output assigned to the move in the PDT τ' will be a word of length 2. If it increases the height of the stack (by 1 according to assumption 2) then the output in τ' is a word of length 1, i.e. a letter. In this latter case, the new stack symbol will eventually be removed either by an ε-move to which an output of length 1 will be assigned, or by a non-ε-move, i.e. a move involving a letter $b \in \Sigma$, to which an output of length 3 will be assigned. In all cases the average length of an output assigned to one letter is $(1+3)/2$ or $1+1$ that is 2.

Formally $\tau' = (Q, \Sigma, \Delta, X, \delta', q_0, Z_0)$ is a deterministic PDT recognizing Dom g = Dom f by empty stack, and defined by:

i) For all $q, p \in Q$, $x \in X$ we have

$$\delta'(q, \varepsilon, x) = (p, \varepsilon, [q, \varepsilon, x]) \quad \text{iff } \delta(q, \varepsilon, x) = (p, \varepsilon, u) \quad \text{for some } u \in \Delta^*;$$

ii) For all $p, q \neq q_0 \in Q$, $a \in \Sigma$, $x \in X$ and $y \in X^*$ we have

$$\delta'(q, a, x) = (p, y, [q, a, x])\sigma^{2-|y|} \quad \text{iff } \delta(q, a, x) = (p, y, u) \quad \text{for some } u \in \Delta^*;$$

iii) For all $q \in Q$, $a \in \Sigma$ and $y \in X^*$ we have:

$$\delta'(q_0, a, Z_0) = (p, y, [q_0, a, Z_0]) \quad \text{iff } \delta(q_0, a, Z_0) = (p, y, u) \quad \text{for some } u \in \Delta^*.$$

It is left to the reader to verify that τ' works as claimed. □

4. Closure under composition.

4.1. General results. It is well known that rational relations are closed under composition (see e.g. [9, Thm. IX, 4.1]), while pushdown relations are not (if $L, M \subseteq \Sigma^*$ are two context-free languages, then I_L and I_M are pushdown relations, but $I_L \circ I_M = I_{L \cap M}$ might not be). Yet we have (see e.g. [12, p. 115]):

THEOREM 4.1. *Let $f: \Sigma_1^* \to \Sigma_2^*$ be a pushdown relation and $g: \Delta_1^* \to \Delta_2^*$ be a rational relation. If $\Sigma_1 = \Delta_2$ (resp. $\Sigma_2 = \Delta_1$) then $f \circ g$ (resp. $g \circ f$) is a pushdown relation.*

Obviously, Theorem 4.1 holds when f and g are partial functions. Using the factorization properties established in the preceding section, we will see that it still holds in the following case:

THEOREM 4.2. *Let $f: \Sigma_1^* \to \Sigma_2^*$ be an unambiguous partial function and $g: \Delta_1^* \to \Delta_2^*$ a rational partial function. If $\Sigma_1 = \Delta_2$ (resp. $\Sigma_2 = \Delta_1$), then $f \circ g$ (resp. $g \circ f$) is an unambiguous partial function.*

Proof. By Theorems 3.3 and 3.2, we have the factorizations

$$f = f_2 \circ I_L \circ f_1^{-1} \quad \text{and} \quad g = g_2 \circ I_R \circ g_1^{-1}$$

where $L \subseteq \Sigma^*$ is a bideterministic language, $R \subseteq \Delta^*$ is a rational language, $f_i: \Sigma^* \to \Sigma_i^*$ $i = 1, 2$ and $g_i: \Delta^* \to \Delta_i^*$ $i = 1, 2$ are alphabetic morphisms such that f_1 and g_1 are injective on L and R respectively.

Case 1. $\Sigma_1 = \Delta_2$. Consider the relation $h = f_1^{-1} \circ g_2: \Delta^* \to \Sigma^*$. By [9, Lemma IX, 4.2], there exist a finite set Γ, a rational subset $K \subseteq \Gamma^*$ and two alphabetic morphisms $h_1: \Gamma^* \to \Delta^*$ and $h_2: \Gamma^* \to \Sigma^*$ such that $h = h_2 \circ I_K \circ h_1^{-1}$. Furthermore for all $(u, v) \in \# h$, there exists exactly one element $w \in K$ such that $h_1(w) = u$ and $h_2(w) = v$.

We have:

$$\begin{aligned} f \circ g &= (f_2 \circ I_L \circ f_1^{-1}) \circ (g_2 \circ I_R \circ g_1^{-1}) \\ &= f_2 \circ I_L \circ (h_2 \circ I_K \circ h_1^{-1}) \circ I_R \circ g_1^{-1} \\ &= f_2 \circ h_2 \circ I_{h_2^{-1}(L)} \circ I_K \circ I_{h_1^{-1}(R)} \circ h_1^{-1} \circ g_1^{-1} \\ &= f_2 \circ h_2 \circ I_M \circ (g_1 \circ h_1)^{-1} \end{aligned}$$

where $M = h_1^{-1}(R) \cap K \cap h_2^{-1}(L)$ is, according to [16, Thms 12.2 and 12.3], a bideterministic language.

It now suffices to show that h_1 is injective on M. Consider $z, t \in M$ such that $h_1(z) = h_1(t)$. Since $h_2 \circ I_M \circ h_1^{-1} = I_L \circ f_1^{-1} \circ g_2 \circ I_R$ is a partial function, we have $h_2(z) = h_2(t)$ and therefore $z = t$, which proves that h_1 is injective on M as claimed.

Case 2. $\Sigma_2 = \Delta_1$. We have $g \circ f = g_2 \circ I_R \circ g_1 \circ f_2 \circ I_L \circ f_1^{-1}$. Since $I_R \circ g_1 \circ f_2: \Sigma^* \to \Delta^*$ is a rational partial function, there exist a finite set Γ, a rational language $K \subseteq \Gamma^*$, an alphabetic morphism $h_1: \Gamma^* \to \Sigma^*$ which is injective on K and an alphabetic morphism $h_2: \Gamma^* \to \Delta^*$ such that $I_R \circ g_1 \circ f_2 = h_2 \circ I_K \circ h_1^{-1}$. Thus we obtain

$$\begin{aligned} g \circ f &= g_2 \circ (h_2 \circ I_K \circ h_1^{-1}) \circ I_L \circ f_1^{-1} \\ &= (g_2 \circ h_2) \circ I_K \circ I_{h_1^{-1}(L)} \circ h_1^{-1} \circ f_1^{-1} \\ &= (g_2 \circ h_2) \circ I_{K \cap h_1^{-1}(L)} \circ (f_1 \circ h_1)^{-1}. \end{aligned}$$

Since $K \cap h_1^{-1}(L)$ is bideterministic, the proof is complete. □

The families of (left) deterministic partial functions and of right sequential partial functions are incomparable. Therefore, if we compose a deterministic partial function with a right sequential partial function (or more generally with a rational partial

function), the result is not necessarily a deterministic partial function. However, we have:

THEOREM 4.3. *Let* $f:\Sigma_1^*\to\Sigma_2^*$ *be a deterministic partial function and* $g:\Delta_1^*\to\Delta_2^*$ *a* DGSM *mapping. If* $\Sigma_1=\Delta_2$ *(resp.* $\Sigma_2=\Delta_1$*), then* $f\circ g$ *(resp.* $g\circ f$*) is a deterministic partial function.*

Proof. Let $\mathcal{T}=(Q,\Sigma_1,\Sigma_2,X,\delta,q_0,Z_0,F)$ be a deterministic pushdown transducer defining f, and $\mathcal{S}=(P,\Delta_1,\Delta_2,\lambda,\theta,p_0,H)$ a DGSM (sequential transducer with final states) defining g. We recall our convention from § 3.3 that for all $q\in P$ and $u\in\Delta_1^*$ we write $q\cdot u$ and $q*u$ instead of $\lambda(q,u)$ and $\theta(q,u)$, respectively.

Case 1. $\Sigma_2=\Delta_1$. It suffices to notice that the partial function $g\circ f:\Sigma_1^*\to\Delta_2^*$ is defined by the deterministic pushdown $\tau'=(Q\times P,\Sigma_1,\Delta_2,X,\delta',(q_0,p_0),Z_0,F\times H)$ where δ' is defined, for all $q,q'\in Q$, $p\in P$, $a\in\Sigma_1\cup\{\varepsilon\}$, $u\in\Delta_2^*$, $x\in X$ and $x'\in X^*$ by:

$$((q',p\cdot u),x',p*u)\in\delta'((q,p),a,x)\quad\text{iff }(q',x',u)\in\delta(q,a,x).$$

Case 2. $\Sigma_1=\Delta_2$. We shall first consider three particular subcases.

Subcase i. g is length preserving (which is equivalent to saying that for all $p\in P$ and $a\in\Delta_1$, we have $p*a\subseteq\Delta_2$). Then $f\circ g$ is defined by the deterministic pushdown transducer $\tau_1=(Q\times P,\Delta_1,\Sigma_2,X,\delta',(q_0,p_0),Z_0,F\times H)$ where for all $q,q'\in Q$, $p\in P$, $a\in\Delta_1\cup\{\varepsilon\}$, $x\in X$, $x'\in X^*$ and $u\in\Sigma_2^*$ we have:

$$((q',p\cdot a),x',u)\in\delta'((q,p),a,x)\quad\text{iff }(q',x',u)\in\delta(q,p*a,x).$$

Subcase ii. For all $a\in\Delta_1$, we have $g(a)=(a,1)\cdots(a,n)$ for some integer $n>0$ (in particular $|g(a)|=n$). Consider the set $R=\{(q,a,i)\in Q\times\Delta_1\times\mathbb{N}\mid 1\leqq i\leqq|g(a)|\}$ and define the pushdown transducer

$$\tau'=(R,\Delta_1,\Sigma_2,X,\delta',(q_0,a_0,|g(a_0)|),Z_0,F)$$

(where a_0 is a fixed arbitrary letter) in the following way:

- For all $q,q'\in Q$, $a,b\in\Delta_1$, $x\in X$, $x'\in X^*$ and $u\in\Sigma_2^*$ we have

$$((q',a,1),x',u)\in\delta'((q,b,|g(b)|),a,x)\quad\text{iff}\quad(q',x',u)\in\delta(q,(a,1),x);$$

- For all $q,q'\in Q$, $a,b\in\Delta_1$, $x\in X$, $x'\in X^*$, $u\in\Sigma_2^*$ and $1<i<|g(a)|$ we have

$$((q',a,i+1),x',u)\in\delta'((q,a,i),\varepsilon,x)\quad\text{iff}\quad(q',x',u)\in\delta(q,(a,i),x);$$

- For all $q,q'\in Q$, $a\in\Delta_1$, $x\in X$, $x'\in X^*$, $u\in\Sigma_2^*$ and $1\leqq i\leqq|g(a)|$ we have

$$((q',a,i),x',u)\in\delta'((q,a,i),\varepsilon,x)\quad\text{iff}\quad(q',x',u)\in\delta(q,\varepsilon,x).$$

It suffices to note that τ' is deterministic and defines $f\circ g$.

Subcase iii. g is an alphabetic morphism. Let $Q'\subseteq Q$ be the subset of all states defining ε-moves. Define the pushdown transducer $\tau'=(Q,\Delta_1,\Sigma_2,X,\delta',q_0,Z_0,F)$ in the following way:

- For all $q\in Q'$, $q'\in Q$, $x\in X$, $x'\in X^*$ and $u\in\Sigma_2^*$ we have

$$(q',x',u)\in\delta'(q,\varepsilon,x)\quad\text{iff}\quad(q',x',u)\in\delta(q,\varepsilon,x);$$

- For all $q\in Q'$, $q'\in Q$, $a\in\Delta_1$, $x\in X$, $x'\in X^*$ and $u\in\Delta_2^*$ we have

$$\text{if } g(a)\neq\varepsilon,\text{ then }(q',x',u)\in\delta'(q,a,x)\text{ iff }(q',x',u)\in\delta(q,g(a),x),$$

$$\text{if } g(a)=\varepsilon,\text{ then }(q,x,\varepsilon)\in\delta'(q,a,x).$$

It is easy to check that τ' is deterministic and that it defines $f\circ g$.

To complete the proof, it suffices to note that every sequential partial function g can be factorized as $g = g_2 \circ g_1$, where g_1 is a length-preserving sequential partial function and g_2 is a morphism. But g_2 can be factorized as $g_2 = g_2'' \circ g_2'$, where g_2' is as in Subcase ii and g_2'' is alphabetic. □

Let $R \subseteq \Sigma^*$ be a rational subset and $f : \Sigma^* \to \Delta^*$ an unambiguous partial function. Since $I_R : \Sigma^* \to \Sigma^*$ is a rational partial function, by Theorem 4.2, the restriction $f \circ I_R$ of f to the subset R is an unambiguous partial function. We can deduce, from the next theorem, a similar result for deterministic partial functions.

We recall that a subset $R \subseteq \Sigma^* \times \Delta^*$ is *recognizable* (cf. [9, p. 68]) if there exist a morphism φ of $\Sigma^* \times \Delta^*$ into a finite monoid M and a subset $N \subseteq M$ such that $R = \varphi^{-1}(N)$.

THEOREM 4.4. *For each unambiguous (resp. deterministic) partial function $f : \Sigma^* \to \Delta^*$ and each recognizable subset $R \subseteq \Sigma^* \times \Delta^*$, the partial function $g : \Sigma^* \to \Delta^*$, whose graph is $\# g = \# f \cap R$, is an unambiguous (resp. deterministic) partial function.*

Proof. Let $\mathcal{T} = (Q, \Sigma, \Delta, X, \delta, q_0, Z_0, F)$ be an unambiguous (resp. deterministic) PDT defining f and let $R = \varphi^{-1}(N)$ be as in the above definition. Then it suffices to observe that g is defined by the unambiguous (resp. deterministic) PDT $\mathcal{T}' = (Q \times M, \Sigma, \Delta, X, \delta', q_0 \times \{1\}, Z_0, F \times N)$ where we have denoted by 1 the unit of the monoid M and where δ' is defined by:

- For all $q, q' \in Q$, $m \in M$, $a \in \Sigma \cup \{\varepsilon\}$, $x \in X$, $y \in X^*$ and $u \in \Delta^*$, we have:

$$((q', m \cdot \varphi(a, u)), y, u) \in \delta'((q, m), a, x) \quad \text{iff} \quad (q', y, u) \in \delta(q, a, x). \qquad \square$$

As a consequence we have:

COROLLARY 4.5. *For each unambiguous (resp. deterministic) partial function $f : \Sigma^* \to \Delta^*$ and each rational subset $R \subseteq \Sigma^*$, the restriction of f to R is an unambiguous (resp. deterministic) partial function.*

Proof. Indeed we have $\# (f \circ I_R) = \# f \cap (R \times \Delta^*)$. □

COROLLARY 4.6. *For each unambiguous (resp. deterministic) partial function $f : \Sigma^* \to \Delta^*$ and each rational subset $R \subseteq \Delta^*$, $f^{-1}(R)$ is an unambiguous (resp. deterministic) context-free language.*

Proof. Indeed, $f^{-1}(R)$ is the domain of the partial function $g : \Sigma^* \to \Delta^*$ whose graph is: $\# g = \# f \cap (\Sigma^* \times R)$. □

Finally, we recall that sequential and subsequential partial functions are closed under composition (see e.g. [3, Prop. IV, 2.5]).

4.2. A hierarchy of pushdown partial functions. Given two families of partial functions $\mathcal{F}_1$ and $\mathcal{F}_2$, we shall use the customary notation $\mathcal{F}_2 \circ \mathcal{F}_1$ to indicate the family of all partial functions of the form $f_2 \circ f_1$ with $f_1 \in \mathcal{F}_1$ and $f_2 \in \mathcal{F}_2$.

Theorem 4.2 shows that: UPDF $\circ$ RATF = RATF $\circ$ UPDF = UPDF. This implies in particular, that DET, DET $\circ$ RATF, RATF $\circ$ DET and RAT $\circ$ DET $\circ$ RAT are subfamilies of UPDF. We will see that all inclusions among these families are strict.

THEOREM 4.7. *The strict inclusions in Fig. 2 hold.*

Proof. For obvious reasons of symmetry it suffices to consider the left part of the diagram.

Strict inclusions (1) and (7) can be established arguing on the domain of the different partial functions. The same holds for strict inclusion (2). Indeed, by Theorems 2.1 and 4.3, we have RATF $\circ$ DET $\circ$ RATF = $\text{SEQ}^{\text{R}} \circ$ DET $\circ$ SEQ^{R} which shows that the domain of a partial function $f \in$ RATF $\circ$ DET $\circ$ RATF is the inverse image of a (left) deterministic language by a right sequential partial function. But we do not in this way get all unambiguous languages (cf. e.g. [21, p. 40]).

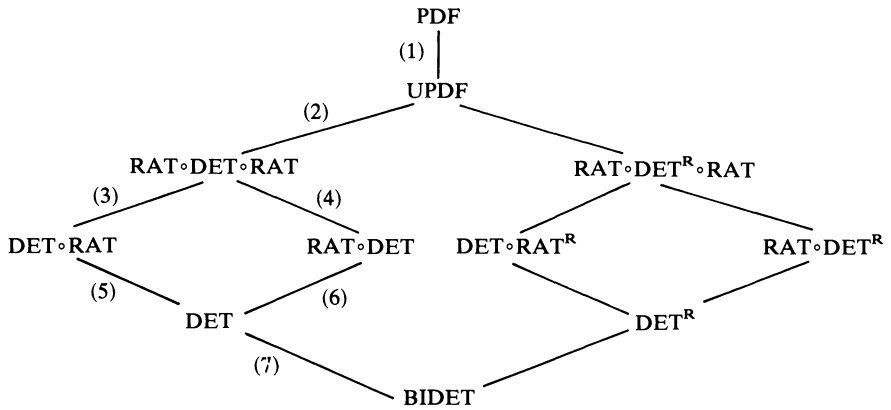


FIG. 2

Strict inclusions (5) and (6) are consequences of the fact that RATF and DET are incomparable.

Strict inclusions (3) and (4) follow from the fact that RATF ∘ DET and DET ∘ RATF are incomparable, which we will now prove.

In order to show that DET ∘ RATF $\not\subseteq$ RATF ∘ DET, let $\Sigma = \{a, b, c, d\}$ and set $L = \{a^n b^n c \mid n > 0\} \cup \{a^n b^{2n} d \mid n > 0\}$. Then $I_L : \Sigma^* \to \Sigma^*$ belongs to DET ∘ RATF as is easily seen, but does not belong to RATF ∘ DET, since its domain is not a deterministic context free language.

We are now left with proving RATF ∘ DET $\not\subseteq$ DET ∘ RATF. This is done in two steps.

LEMMA 4.8. *Let* $\Sigma = \{a, b\}$ *and* $\Delta = \{c, d\}$. *The partial function* $f : \Sigma^* \to \Delta^*$, *defined by*:

$$\text{for all } m, n \geqq 0, \quad f(a^n b a^m b) = \begin{cases} c^n & \text{if } n \geqq m, \\ d & \text{otherwise,} \end{cases}$$

and

$$f(u) = \varnothing \quad \text{if } u \notin a^* b a^* b,$$

is not deterministic.

Proof. Assume f is defined by a deterministic pushdown transducer $\mathcal{T} = (Q, \Sigma, \Delta, \delta, q_0, Z_0, F)$. We shall assign to $\mathcal{T}$ a deterministic pushdown automaton $\mathcal{A}$ recognizing a language $\mathcal{L} \subseteq (\Sigma \cup \Delta)^*$ consisting of words belonging to the shuffle of a word $u \in \Sigma^*$ with its image $f(u) \in \Delta^*$.

Formally, we consider the pushdown automaton $\mathcal{A} = (Q \cup Q', \Sigma \cup \Delta, X, \delta', q_0, Z_0, F)$ where the set Q' of new states disjoint from Q and δ' is defined by replacing each relation $(q', y, u) \in \delta(q, a, x)$ by a set of relations according to the following rules:

i) If $u = \varepsilon$, then the collection is reduced to:

$$(q', y) \in \delta'(q, a, x);$$

ii) If $u = u_1 \cdots u_p$ $(u_i \in \Delta, 1 \leqq i \leqq p)$ then the collection is:

$$(q'_1, x) \in \delta'(q, a, x),$$
$$(q'_{i+1}, x) \in \delta'(q'_i, u_i, x), \qquad 1 \leqq i < p,$$
$$(q', y) \in \delta'(q'_1, u_p, x),$$

where $q'_1, \cdots, q'_p$ are new states.

If we denote by $\mathscr{L}$ the (deterministic) context-free language recognized by $\mathscr{A}$, then $\mathscr{L} \cap \{a, b, c\}^* \subseteq \{a, b\}^* c^*$ since the occurrence of c cannot be output before $\mathscr{A}$ knows for sure that $n \geqq m$, i.e., before it reads the second occurrence of b. More precisely $\mathscr{L} \cap \{a, b, c\}^* = \{a^n b a^m b c^n \mid n \geqq m\}$, which contradicts the context-freeness of $\mathscr{L}$. □

LEMMA 4.9. *Let* $\Sigma = \{a, b\}$, $\Delta = \{c, d\}$ *and consider the partial function* $f : \Sigma^* \to \Delta^*$ *defined by*

$$f(u) = \varnothing \quad \text{if } u \notin a^* b^*$$

and

$$f(a^n b^m) = \begin{cases} c^n & \text{if } n \geqq m \geqq 0, \\ d & \text{otherwise.} \end{cases}$$

Then $f \in (\mathrm{RATF} \circ \mathrm{DET}) \setminus (\mathrm{DET} \circ \mathrm{RATF})$.

Proof. i) $f \in \mathrm{RATF} \circ \mathrm{DET}$. Let $g : \Sigma^* \to \Delta^*$ be the deterministic partial function defined by

$$g(u) = \varnothing \quad \text{if } u \notin a^* b^*,$$

$$g(a^n b^m) = \begin{cases} c^n & \text{if } n \geqq m > 0, \\ c^n d^{m-n} & \text{otherwise.} \end{cases}$$

Now consider the right sequential partial function $h : \Delta^* \to \Delta^*$ satisfying for all $u \in \Delta^*$, $h(uc) = uc$ and $h(ud) = d$. Then it suffices to note that $f = h \circ g$.

ii) $f \notin \mathrm{DET} \circ \mathrm{RATF}$. By Theorems 2.1 and 4.3, it suffices to verify that f cannot be factorized as $f = h \circ g$ where $g : \Sigma^* \to \Gamma^*$ is a right sequential partial function and $h : \Gamma^* \to \Delta^*$ is a deterministic partial function.

Assume this is the case and let φ be the canonical morphism of Σ^* onto the transition monoid of a sequential transducer defining g (cf. e.g. [10, p. 157]).

Choose an integer $n > 0$ satisfying the conditions:

$$\varphi(a^n) = \varphi(a^{2n}) \quad \text{and} \quad \varphi(b^n) = \varphi(b^{2n}). \tag{4.1}$$

Then there exist four words u, v, w, $z \in \Gamma^*$ such that for all r, $s \geqq 0$ we have $g(a^{n(r+1)} b^{n(s+1)}) = u^r v w^s z$. This implies the following:

$$h(u^r v w^s z) = \begin{cases} c^{n(r+1)} & \text{if } r \geqq s, \\ d & \text{otherwise.} \end{cases}$$

Consider the sequential partial functions $h_1 : \Sigma^* \to \Gamma^*$ and $h_2 : \Delta^* \to \Delta^*$ defined by:

$$h_1(t) = \begin{cases} u^r v w^s z & \text{if } t = a^r b a^s b, \ r, s \geqq 0, \\ \varnothing & \text{otherwise,} \end{cases}$$

and

$$h_2(t) = \begin{cases} c^r & \text{if } t = c^{n(r+1)}, \\ t & \text{otherwise.} \end{cases}$$

Then by Theorem 4.3, the partial function $h_2 \circ h \circ h_1 : \Sigma^* \to \Delta^*$ is deterministic. But this contradicts Lemma 4.8, since this partial function is precisely the one defined in that lemma. □

5. Decision problems. Let $\mathscr{F}_1$ and $\mathscr{F}_2$ be two subfamilies of pushdown relations and consider $f_1 \in \mathscr{F}_1$ and $f_2 \in \mathscr{F}_2$. We are concerned in this section with the two following

decision problems:

Problem 1. $f_1 \in \mathscr{F}_2$?

Problem 2. $f_1 = f_2$?

Problem 1 is known to be undecidable in case $\mathscr{F}_1 = \mathscr{F}_2 = \text{RAT}$ (see e.g. [3, Thm. III, 8.4]. Several authors have independently proved that Problem 1 is decidable when $\mathscr{F}_1 = \text{RAT}$ and $\mathscr{F}_2 = \text{RATF}$ (cf. [4] and [20]).

THEOREM 5.1. *Given an arbitrary rational relation $f: \Sigma^* \to \Delta^*$, it is decidable whether or not it is a partial function, i.e., whether $f(u)$ contains at most one element for all $u \in \Sigma^*$.*

In particular this proves that Problem 2, under the same assumption, is decidable. Indeed, $f_1 = f_2$ if and only if the two rational languages $\text{Dom}\, f_1$ and $\text{Dom}\, f_2$ are equal and if the union of f_1 and f_2 is again a partial function. Therefore we have (cf. the same references):

THEOREM 5.2. *Given two arbitrary rational partial functions $f_i: \Sigma^* \to \Delta^*$, $i = 1, 2$, it is decidable whether $f_1 = f_2$, i.e., whether $f_1(u) = f_2(u)$ holds for all $u \in \Sigma^*$.*

Assume $\mathscr{F}_1 = \text{RAT}$ and $\mathscr{F}_2$ is any of the families SEQ, SEQ^R, BISEQ, SUBSEQ, SUBSEQ^R, BISUBSEQ. For each of these cases Problem 1 can be decided (cf. [6]):

THEOREM 5.3. *Given an arbitrary rational relation $f: \Sigma^* \to \Delta^*$, each of the following problems is decidable:*

(5.1) $f \in \text{SEQ}$?

(5.2) $f \in \text{SEQ}^R$?

(5.3) $f \in \text{BISEQ}$?

(5.4) $f \in \text{SUBSEQ}$?

(5.5) $f \in \text{SUBSEQ}^R$?

(5.6) $f \in \text{BISUBSEQ}$?

A stronger result than Theorem 5.2 has been shown in [7, Thm. 7], namely that Problem 2 is decidable for $\mathscr{F}_1 = \text{RATF}$ and $\mathscr{F}_2 = \text{UPDT}$. We will further strengthen this result in two ways.

THEOREM 5.4. *Given an arbitrary rational relation $f_1: \Sigma^* \to \Delta^*$ and an unambiguous pushdown function $f_2: \Sigma^* \to \Delta^*$, it is decidable whether $f_1 = f_2$.*

Proof. By [7, Thms. 1 and 7]. □

THEOREM 5.5. *Let $f_1: \Sigma^* \to \Delta^*$ be a pushdown relation and $f_2: \Sigma^* \to \Delta^*$ a rational partial function such that $\text{Dom}\, f_2 \subseteq \text{Dom}\, f_1$. Then it is decidable whether f_1 and f_2 are equal.*

Proof. Observe that under the assumptions of the theorem, $\text{Dom}\, f_2 = \text{Dom}\, f_1$ if and only if the context-free language $\text{Dom}\, f_1 \backslash \text{Dom}\, f_2$ is empty. This is known to be decidable. We shall from now on assume that $\text{Dom}\, f_1 = \text{Dom}\, f_2$.

Let Δ_1 and Δ_2 be two disjoint copies of Δ such that $\Delta_1 \cap \Sigma = \Delta_2 \cap \Sigma = \varnothing$. The idea of the proof is to define a pushdown relation $f: \Sigma^* \to (\Delta_1 \cup \Delta_2)^*$ with the following property. For each $u \in \Sigma^*$, every word in $f(u)$ belongs to the shuffle of the copy (in Δ_1^*) of some word of $f_1(u)$ with the copy (in Δ_2^*) of some word of $f_2(u)$. In particular, if p_1 and p_2 are the canonical projections of $(\Delta_1 \cup \Delta_2)^*$ over Δ^*, we will have for all $u \in \Sigma^*$: $f_1(u) = p_1 \circ f(u)$ and $f_2(u) = p_2 \circ f(u)$. Therefore the initial decision problem will be reduced to testing whether the two homomorphisms p_1 and p_2 agree over the context-free language $\text{Im}\, f = f(\Sigma^*)$.

Denote by $j_1 : \Delta^* \to \Delta_1^*$ and $j_2 : \Delta^* \to \Delta_2^*$ the canonical isomorphisms. By Theorem 3.1 the pushdown relation $j_1 \circ f_1 : \Sigma^* \to \Delta_1^*$ and the rational partial function $j_2 \circ f_2 : \Sigma^* \to \Delta_2^*$ admit the factorizations:

$$j_1 \circ f_1 = g_1 \circ I_L \circ h_1^{-1} \quad \text{and} \quad j_2 \circ f_2 = g_2 \circ I_R \circ h_2^{-1}$$

where $L \subseteq \Xi_1^*$ is a context-free and $R \subseteq \Xi_2^*$ a rational language, and where $h_1 : \Xi_1^* \to \Sigma^*$, $g_1 : \Xi_1^* \to \Delta_1^*$, $h_2 : \Xi_2^* \to \Sigma^*$ and $g_2 : \Xi_2^* \to \Delta_2^*$ are alphabetic morphisms. We will assume, without loss of generality, that Ξ_1, Ξ_2, Σ, Δ_1 and Δ_2 are pairwise disjoint.

Let $g_1' : \Xi_1^* \to (\Sigma \cup \Delta_1)^*$ be the morphism defined for all $x \in \Xi_1$ by $g_1'(x) = h_1(x) g_1(x)$, and let $f_1' = g_1' \circ I_L \circ h_1^{-1}$.

Consider next the rational subset $R' \subseteq (\Xi_2 \cup \Delta_1)^*$ which is the shuffle of $R \subseteq \Xi_2^*$ with Δ_1^* (cf. e.g. [9, Prop. II, 3.4]). Define the alphabetic morphisms $h_2' : (\Xi_2 \cup \Delta_1)^* \to (\Sigma \cup \Delta_1)^*$ and $g_2' : (\Xi_2 \cup \Delta_1)^* \to (\Delta_2 \cup \Delta_1)^*$ by letting

$$h_2'(x) = \begin{cases} x & \text{if } x \in \Delta_1, \\ h_2(x) & \text{if } x \in \Xi_2, \end{cases} \qquad g_2'(x) = \begin{cases} x & \text{if } x \in \Delta_1, \\ g_2(x) & \text{if } x \in \Xi_2. \end{cases}$$

Then $f_2' = g_2' \circ I_{R'} \circ h_2^{-1} : (\Sigma \cup \Delta_1)^* \to (\Delta_2 \cup \Delta_1)^*$ is a rational relation, and we leave it to the reader to verify that the pushdown relation $f = f_2' \circ f_1' : \Sigma^* \to (\Delta_1 \cup \Delta_2)^*$ satisfies the condition: $f_1 = p_1 \circ f$ and $f_2 = p_2 \circ f$ as claimed.

Assume $f_1 = f_2$ and consider $x \in f(\Sigma^*)$. Then we have $x \in f(u)$ for some $u \in \Sigma^*$. This yields:

$$p_1(x) \in p_1 f(u) = f_1(u) \quad \text{and} \quad p_2(x) \in p_2 f(u) = f_2(u).$$

Since $f_1(u)$ and $f_2(u)$ are two equal singletons, we obtain $p_1(x) = p_2(x)$.

Conversely, assume that $p_1(x) = p_2(x)$ holds for all $x \in f(\Sigma^*)$. Then for all $u \in \Sigma^*$ we obtain:

$$f_1(u) = p_1 f(u) = p_2 f(u) = f_2(u).$$

In other words, $f_1 = f_2$ if and only if the two morphisms are equivalent over the subset $f(\Sigma^*)$. Since this subset is context-free (cf. e.g. [12, p. 116]), this last problem is decidable, by [8] or [2]. □

REFERENCES

[1] A. V. AHO AND J. D. ULLMAN, *The Theory of Parsing, Translation and Compiling*, Vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[2] J. ALBERT AND K. CULIK II, *Test sets for homomorphism equivalence on context-free languages*, in Proc. of the 7th ICALP, Lecture Notes in Computer Science, 85, Springer, New York, 1980, pp. 12–18.

[3] J. BERSTEL, *Transductions of Context-Free Language*, Teubner Verlag, Berlin, 1979.

[4] M. BLATTNER AND T. HEAD, *Single valued a-transducers*, J. Comput. System Sci., 15 (1977), pp. 310–327.

[5] C. CHOFFRUT, *Contribution a l'étude de quelques families remarquables de fonctions rationnelles*, Thèse de Doctorat d'Etat, Université Paris 7, Paris, 1978.

[6] ———, *Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles*, Theoret. Comput. Sci., 5 (1977), pp. 325–337.

[7] K. CULIK II, *Some decidability results about regular and pushdown translations*, Inform. Process. Lett., 8 (1979), pp. 5–8.

[8] K. CULIK II AND A. SALOMAA, *On the decidability of homomorphism equivalence for languages*, J. Comput. System Sci., 17 (1978), pp. 43–51.

[9] S. EILENBERG, *Automata, Languages and Machines*, Vol. A, Academic Press, New York, 1974.

[10] ———, *Automata, Languages and Machines*, Vol. B, Academic Press, New York, 1976.

[11] J. ENGELFRIET AND G. ROZENBERG, *Equality languages and fixed point languages*, Inform. and Control, 39 (1978), pp. 20–49.
[12] M. FLIESS, *Transductions algébriques*, R.A.I.R.O., R1, (1970), pp. 109–125.
[13] S. GINSBURG, *The Mathematical Theory of Context-Free Languages*, McGraw-Hill, New York, 1966.
[14] S. GINSBURG, S. GREIBACH AND J. HOPCROFT, *Studies in abstract families of languages*, Mem. Amer. Math. Soc., 113 (1966), pp. 285–296.
[15] M. A. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.
[16] J. E. HOPCROFT AND J. D. ULLMAN, *Formal Languages and Their Relations to Automata*, Addison-Wesley, Reading, MA, 1969.
[17] M. NIVAT, *Transductions des langages de Chomsky*, Ann. Inst. Fourier, 18 (1968), pp. 339–456.
[18] M. P. SCHÜTZENBERGER, *Sur les relations fonctionnelles entre monoides libres*, Theoret. Comput. Sci., 3 (1976), pp. 243–260.
[19] ———, *Sur une variante des fonctions séquentielles*, Theoret. Comput. Sci., 4 (1977), pp. 47–57.
[20] ———, *Sur les relations rationnelles*, in Automata Theory and Formal Languages, H. Brakhage, ed., 2nd GI Conference, Lecture Notes in Computer Science, 33, Springer, Berlin, 1975, pp. 209–213.
[21] M. SORIA, *Langages quasidéterministes*, Thèse de 3ème Cycle, Université Paris VII, Paris, France, 1978.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0008 $01.25/0

QUADRATIC ALGORITHMS FOR MINIMIZING JOINS IN RESTRICTED RELATIONAL EXPRESSIONS*

YEHOSHUA SAGIV†

Abstract. An important step in the optimization of queries in relational databases is minimizing the number of join operations in the evaluation of a query. It is shown that three subclasses of tableaux (including the subclass of simple tableaux) have $O(n^2)$ time equivalence and minimization algorithms. Since tableaux are nonprocedural representations of relational expressions over select, project and join, these minimization algorithms can be used to minimize the number of join operators in expressions whose tableaux belong to one of these subclasses.

Key words. relational database, relational algebra, query optimization, equivalence of queries, conjunctive query, tableau, NP-complete

CR categories. 4.33, 5.25

1. Introduction. The relational model for databases features two high-level query languages: the relational algebra and the relational calculus [9], [10]. The relational algebra is a procedural language that uses operators defined on relations, and a query is usually translated to a relational expression before being evaluated. However, the efficiency with which a query can be answered depends on the relational expression that has been chosen to represent this query. Consequently, a number of papers (e.g., [12], [13], [14], [15], [17], [18]) have considered transformations that reduce the cost of evaluating a query. However, these transformations do not necessarily produce an equivalent query of least cost. Chandra and Merlin [8] show how to perform global optimization on a large class of queries, but their algorithm is exponential in the size of the query.

The most commonly used operators of the relational algebra are select, project and join, and a polynomial time algorithm for optimizing a subclass of expressions with these operators is given in [4], [5]. This optimization technique uses tableaux [3] as a nonprocedural representation of queries. Tableaux are similar to the conjunctive queries of [8], and resemble Zloof's "query-by-example" language [20]. Relational expressions over select, project, and join can be represented by tableaux [3]. In [8] it is shown that every tableau has an equivalent minimal tableau. The importance of this result follows from the fact that an expression with a minimum number of joins corresponds to a minimal tableau. A polynomial minimization algorithm for the class of simple tableaux is given in [4] (although the problem is NP-complete in the general case [3]). In [5] it is shown how to obtain in polynomial time an optimal expression from a minimal simple tableau (if such an expression exists).

This optimization technique is machine independent. It is capable of minimizing the number of join operators (note that join is the most expensive operator to implement), eliminating redundant subexpressions, and applying select and project as early as possible. This type of optimization should be augmented with machine dependent optimization that takes into consideration the size of the relations, sorted columns, etc.

* Received by the editors November 21, 1979, and in revised form May 28, 1982. This work was supported in part by the National Science Foundation under grants MCS-76-15255 and MCS-80-03308.

† Institute of Mathematics and Computer Science, The Hebrew University, Givat Ram 91904, Jerusalem, Israel.

In this paper we describe new minimization algorithms for two subclasses of tableaux. We also show how to improve the running time of the minimization algorithm for the class of simple tableaux. All these algorithms have an $O(n^2)$ running time. It is shown that each one of the three subclasses of tableaux discussed in this paper also has an $O(n^2)$ time equivalence algorithm. Finally, in §§ 7 and 8 we touch upon the problems of minimizing tableaux in the general case and obtaining optimal expressions from minimal tableaux.

2. Basic definitions.

2.1. The relational model. The relational model for databases [9] assumes that the data is stored in tables called *relations*. The columns of a table correspond to *attributes*, and the rows to *records* or *tuples*. Each attribute has an associated *domain* of values. A tuple is viewed as a mapping from the attributes to their domains, since no canonical ordering of the attributes is needed in this way. If r is a relation with a column corresponding to the attribute A, and μ is a tuple in r, then $\mu(A)$ is the value of the A-component of μ. In this paper we usually denote a relation as a set of tuples.

A *relation scheme* is a set of attributes labeling the columns of a table, and it is usually written as a string of attributes. We often use the relation scheme itself as the name of the table. A relation is just the "current value" of a relation scheme.

2.2. The relational algebra and relational expressions. The relational algebra [9], [10], is a set of operators defined on relations. In this paper we consider the operators select, project, and join.

Let r be a relation defined on a set of attributes X, A an attribute in X and c a value from the domain of A. The *selection* $A = c$, written $\sigma_{A=c}(r)$, is

$$\sigma_{A=c}(r) = \{\mu \mid \mu \in r \text{ and } \mu(A) = c\}.$$

Let Y be a subset of X, the *projection* of r onto Y, written $\pi_Y(r)$, is

$$\pi_Y(r) = \{\mu \mid \mu \text{ is a mapping on } Y, \text{ and there is a } \nu \text{ in } r \text{ such that } \mu(Y) = \nu(Y)\}.$$

Let r_1 and r_2 be relations defined on the relation schemes R_1 and R_2, respectively. The (natural) *join* of r_1 and r_2, written $r_1 \bowtie r_2$, is

$$r_1 \bowtie r_2 = \{\mu \mid \mu \text{ is a mapping on } R_1 \cup R_2, \text{ and there is } \nu_1 \text{ in } r_1 \text{ and } \nu_2 \text{ in } r_2 \text{ such that } \mu(R_1) = \nu_1(R_1) \text{ and } \mu(R_2) = \nu_2(R_2)\}.$$

Note that the join includes intersection (when R_1 and R_2 are the same) and cartesian product (when R_1 and R_2 are disjoint) as special cases.

The relational algebra operators are used to formulate queries in terms of relational expressions. A *restricted relational expression* consists of select, project and join as operators, and relation schemes as operands.

2.3. Expression values and equivalence of expressions. An underlying assumption in many papers (e.g., [1], [6], [7]) is the existence of a single universal relation at each instant of time. This relation is defined on the set of all the attributes, and it is called a *universal instance* or just an *instance*. If I is an instance and E is a relational expression, then each relation scheme R_i in E is assigned the relation $\pi_{R_i}(I)$. The value of E with respect to I, written $v_I(E)$, is computed by applying operators to operands in the following natural way:

(1) If E is a single relation scheme R_i, then $v_I(E) = \pi_{R_i}(I)$.

(2) (a) If $E = \sigma_{A=c}(E_1)$, then $v_I(E) = \sigma_{A=c}(v_I(E_1))$.

(b) If $E = \pi_X(E_1)$, then $v_I(E) = \pi_X(v_I(E_1))$.

(c) If $E = E_1 \bowtie E_2$, then $v_I(E) = v_I(E_1) \bowtie v_I(E_2)$.

We may regard a relational expression as a mapping from instances to relations, i.e., the expression E maps the instance I to the relation $v_I(E)$. Two expressions E_1 and E_2 are *equivalent* if they define the same mapping. That is, if for all instances I, $v_I(E_1) = v_I(E_2)$.

The algorithms given in this paper do not depend upon the universal instance assumption. These algorithm can be applied in $O(n^2)$ time even if the relations assigned to the relation schemes do not necessarily come from an instance (see [3] for details on how to test equivalence in this case).

3. Tableaux. Tableaux are just another way of representing mappings from instances to relations. Unlike relational expressions, tableaux are nonprocedural representation of queries in exactly the sense that relational calculus [9], [10] is nonprocedural.

A *tableau* is a matrix consisting of a *summary* and a set of *rows*. The columns of a tableau correspond to the attributes of the universe in a fixed order. The symbols appearing in a tableaux are chosen from:

(1) distinguished variables, usually denoted by subscripted a's.
(2) nondistinguished variables, usually denoted by subscripted b's.
(3) constants, which are drawn from the domains of the attributes
(4) blank.

Each row may contain constants, distinguished and nondistinguished variables. The summary has constants, distinguished variables and blanks. A variable cannot appear in more than one column, and a distinguished variable may appear in a particular column only if it appears in the summary. Further, if a constant or a distinguished variable appears in some column A of the summary, then it must also appear in column A of at least one row.

Let T be a tableau with a summary w_0 and rows $w_1, \cdots, w_n$, and let S be the set of all the nonblank symbols in T. A *valuation* ρ for T maps each symbol in S to a constant, such that if c is a constant in S, then $\rho(c) = c$. The valuation ρ is extended to the rows and summary of T by defining $\rho(w_i)$ to be the result of substituting $\rho(v)$ for all variables v in w_i.

A tableau T defines a mapping from instances to relations on its *target relation scheme*, which is the set of all the attributes corresponding to columns that have a nonblank symbol in the summary. Given an instance I, the value of T, written $T(I)$, is

$$T(I) = \{\rho(w_0) | \text{ for some valuation } \rho, \text{ we have } \rho(w_i) \text{ in } I \text{ for } 1 \leqq i \leqq n\}.$$

Conventionally, we also regard ϕ as a tableau. This tableau represents the function that maps every instance to the empty relation.

Example 1. Consider the following tableau:

$T =$

A	B	C
a_1		a_3
a_1	2	b_3
b_1	b_2	a_3
a_1	b_2	b_4

The summary is shown first, with a line below it, and integers are used as constants.

Tableau T defines a relation on the relation scheme AC. For example, suppose that I is the instance {211, 121, 122}.

Consider the valuation ρ that assigns 2 to b_2 and 1 to every other variable in T. Under this valuation, each row of T becomes 121, which is a member of I. Therefore, $\rho(a_1a_3) = 11$ is in $T(I)$.

If ρ assigns 1 to a_1, b_1, b_3 and b_4, and 2 to b_2 and a_3, the first and third rows become 121 and the second row becomes 122; both are members of I, and so 12 is in $T(I)$.

Since no valuation for T produces a tuple other than 11 or 12, $T(I) = \{11, 12\}$.

Given a restricted relational expression E, we can construct in polynomial time a corresponding tableau that represents the same mapping [3]. However, there are tableaux that do not correspond to any relational expression [3].

3.1. Equivalence of tableaux. Two tableaux T_1 and T_2 are *equivalent*, written $T_1 \equiv T_2$, if for all instances I, $T_1(I) = T_2(I)$. We say that T_1 is *contained in* T_2, written $T_1 \subseteq_T T_2$, if for all I, $T_1(I) \subseteq T_2(I)$.

Let T_1 and T_2 be tableaux with the same target relation scheme, and let S_1 and S_2 be the sets of symbols of T_1 and T_2, respectively. A *homomorphism* is a mapping $\xi : S_1 \to S_2$ such that:

(a) If c is a constant, then $\xi(c) = c$.

(b) If s is the summary of T_1, then $\xi(s)$ is the summary of T_2.

(c) If w is any row of T_1, then $\xi(w)$ is a row of T_2.

The following theorem is proved in [3], [8].

THEOREM 1. *$T_2 \subseteq_T T_1$ if and only if T_1 and T_2 have the same target relation scheme, and there is a homomorphism $\xi : S_1 \to S_2$.*

By condition (c), a homomorphism ξ corresponds to a mapping θ from the rows of T_1 to the rows of T_2. The mapping θ is called a *containment mapping*, and it satisfies the following conditions:[1]

(1) If row w of T_1 has a constant in some column A, then $\theta(w)$ has the same constant in column A.

(2) If row w of T_1 has a distinguished variable in column A, then $\theta(w)$ has a distinguished variable in column A.

(3) If rows w and v have the same nondistinguished variable in column A, then rows $\theta(w)$ and $\theta(v)$ have the same symbol in column A.

COROLLARY 2. [3]. *Tableaux T_1 and T_2 are equivalent if and only if they have identical summaries up to renaming of distinguished variables, and there are containment mappings in both directions.*

A tableau T is *minimal* if T is not equivalent to any tableau with fewer rows. If an expression E corresponds to a minimal tableau, then E is not equivalent to any expression with fewer joins. Since it has been shown that each tableau has a unique (up to renaming of variables[2]) equivalent minimal tableau [3], [8], it follows that one can minimize the number of joins in a restricted relational expression E by minimizing its corresponding tableau T. Further, the minimal tableau equivalent to T can be obtained by deleting some of the rows of T. Therefore, if E is a nonminimal expression,

[1] In this paper we consider only equivalence (and not proper containment) of tableaux, and therefore the definition of a containment mapping is more restricted than the original definition given in [3].

[2] Let T and T' be tableaux with sets of symbols S and S', respectively. T' is a renaming of T if there is a one-to-one homomorphism ξ from S onto S' such that the simultaneous replacement of each variable v in T with $\xi(v)$ produces T'.

then E has a proper subexpression (corresponding to its unique minimal tableau) whose value is the same as E and, so, the evaluation of a nonminimal expression can never be more efficient than the evaluation of its minimal form.

A *core* of T is a subset of the rows of T that forms a minimal tableau equivalent to T. (Note that all the cores of T are the same up to renaming of variables.) A *folding* is a containment mapping from the rows of a tableau T to the rows of a core of T, such that every row in the core of T is mapped to itself. Since a tableau is equivalent to its core, every tableau T has a folding. Note that a homomorphism that corresponds to a folding maps every variable in the core of T to itself. The following corollary is an immediate consequence of the results stated so far.

COROLLARY 3. *Let T_1 and T_2 be tableaux with the same summary. If the rows of T_1 are a subset of the rows of T_2, then*

(1) $T_2 \subseteq_T T_1$, *and*

(2) $T_2 \equiv T_1$ *if and only if a core of T_2 is contained in T_1.*

Let w and x be rows of tableaux over the same set of attributes. Row w *covers* row x, if for all columns A,

(1) if x has a constant in column A, then w has the same constant in column A, and

(2) if x has a distinguished variable in column A, then w has a distinguished variable in column A.

If x is mapped to w, and w covers x, then the first two conditions of a containment mapping are satisfied. Let R and S be sets of rows over the same set of attributes. The set S covers the set R, if every row of R is covered by some row of S. We say that rows w and x are *equivalent* if w covers x and x covers w.

A symbol (i.e., a variable or a constant) of a tableau T is *repeated* in some column A, if it appears in that column in more than one row. A tableau T is *simple* if whenever T has a repeated nondistinguished variable b in some column A, then b is the only repeated symbol in that column. The class of simple tableaux has an $O(n^3)$ equivalence algorithm [3], and an $O(n^4)$ minimization algorithm [4]. Other equivalence algorithms can be obtained from the containment algorithms of [16]. In particular, testing whether $T_2 \equiv T_1$ can be done in $O(n^2)$ in the following two cases:

(1) Both T_1 and T_2 have at most one repeated nondistinguished variable in each row.

(2) Every row of T_1 is covered by at most two rows of T_2, and every row of T_2 is covered by at most two rows of T_1.

4. Polynomial equivalence algorithms. In this section we consider the following classes of tableaux.

(1) The class of all tableaux T, such that T has at most one repeated nondistinguished variable in each row.

(2) The class of all tableaux T, such that every row of T is covered by at most one row of T besides itself.

Deciding whether a tableau belongs to Class 1 (or whether a tableau is simple) can be done in $O(n)$ time. Deciding whether a tableau belongs to Class 2 can be done in $O(n^2)$.

THEOREM 4. *Each one of the above classes has an $O(n^2)$ time equivalence algorithm.*

Proof. If tableaux T_1 and T_2 belong to Class 1, then the first algorithm of [16] can be used to test whether $T_1 \equiv T_2$. If T_1 and T_2 belong to Class 2, the second algorithm of [16] cannot be applied directly to T_1 and T_2 (since it is not necessarily

true that every row of T_1 is covered by at most two rows of T_2). However, we can still use it in the following way. First, we remove from T_1 every row that does not have an equivalent row in T_2. Let T_1' be the tableau consisting of the remaining rows of T_1. Similarly, T_2' is obtained from T_2 by removing every row that does not have an equivalent row in T_1. We claim that $T_1 \equiv T_2$ if and only if

$$(*) \qquad T_1 \equiv T_1', \quad T_2 \equiv T_2' \quad \text{and} \quad T_1' \equiv T_2'.$$

Further, condition (*) can be tested using the second algorithm of [16]. Clearly, this algorithm can be used to test the first and second equivalences, since the rows of T_1' are a subset of the rows of T_1 and T_1 belongs to Class 2 (and similarly for T_2' and T_2). A row w of T_1' cannot be covered by more than two rows of T_2', since w has an equivalent row x of T_2' and so every row of T_2' that covers w also covers x. Therefore, the second algorithm of [16] can also be used to test the third equivalence of (*).

It remains to be shown that $T_1 \equiv T_2$ if and only if condition (*) is satisfied. Clearly, if condition (*) is true, then $T_1 \equiv T_2$. The "only if" direction follows from the fact that if $T_1 \equiv T_2$, then T_1 and T_2 have identical cores, and so every row in a core of one of them has an equivalent row in every core of the other. □

5. Obtaining minimization algorithms from equivalence algorithms. Let S be a class of tableaux. We say that S is *closed under row deletion* if whenever a tableau T is in S, and T' is obtained by deleting some of the rows of T, then T' is also in S.

THEOREM 5. *Let S be a class of tableaux closed under row deletion. If there is an equivalence algorithm for S that runs in $F(n)$ time ($F(n) \geqq cn$ for some constant c), then there is a minimization algorithm for S that runs in $nF(n)$ time.*

Proof. Suppose that a tableau T in S is not minimal. Then there is a row x such that $T \equiv T - x$, where $T - x$ is T with the row x deleted. Thus, for each row x we have to test whether $T \equiv T - x$, and if so, delete x and continue the process with the remaining rows. This can be done in $nF(n)$ time, since no row has to be considered more than once (because if $T \not\equiv T - x$, then $T' \not\equiv T' - x$ for every T' obtained by deleting some redundant rows of T). □

The classes of tableaux described in § 4 and the class of simple tableaux both have polynomial time equivalence algorithms. Each one of these classes is closed under row deletion and, therefore, Theorem 5 can be applied to obtain polynomial minimization algorithms for these classes. However, the algorithms obtained by applying Theorem 5 are not the most efficient minimization algorithms for these classes. In the following sections we will give for each one of these classes a minimization algorithm that runs in $O(n^2)$ time.

6. Polynomial minimization algorithms.

6.1. Quasi-minimal tableaux. In this section we will show that a tableau, in which a folding does not eliminate any repeated nondistinguished variable, can be minimized in $O(n^2)$ time. This fact is used in developing the minimization algorithms of the following sections.

Let b be a repeated nondistinguished variable of a tableau T. The variable b is *essential* if it appears in every core of T. A tableau T is *quasi-minimal* if all repeated nondistinguished variables of T are essential.

LEMMA 6. *A quasi-minimal tableau can be minimized in $O(n^2)$ time.*

Proof. Consider the following rule for row deletion. Delete row x of T if T has a row w such that for all columns A, if $x(A) \neq w(A)$, then $x(A)$ is a nondistinguished variable that appears nowhere else in T. By [3, Cor. 2], $T \equiv T - x$. We claim that a

quasi-minimal tableau can be minimized by repeatedly deleting rows according to the above rule until no more rows can be deleted. (Clearly, this can be done in $O(n^2)$ time.) Let T be a quasi-minimal tableau. Every repeated nondistinguished variable of T is essential and, therefore, must appear in every core of T. Consider a folding from T onto its core. The corresponding homomorphism maps every repeated nondistinguished variable of T to itself, since the folding maps every row in the core to itself. Therefore, if row x of T is mapped to some other row w in the core of T, then in every column on which x and w disagree, x has a nondistinguished variable that appears nowhere else in T. Thus, x can be deleted by the above rule. □

Each one of the following algorithms for minimizing a tableau T has two steps. In the first step some rows of T are deleted in order to obtain an equivalent quasi-minimal tableau $\bar{T}$. In the second step the algorithm for minimizing quasi-minimal tableau is applied to $\bar{T}$.

6.2. Minimizing tableaux of class 1. Let T be a tableau that has at most one repeated nondistinguished variable in each row. For each repeated nondistinguished variable b, let $W(b)$ be the set of all the rows that contain b. Suppose that some repeated nondistinguished variable b_0 is not essential, and let A be the column in which b_0 appears. Let θ be a folding from the rows of T to a core that does not contain b_0. Obviously, all the rows of $\theta(W(b_0))$ have the same symbol d $(d \neq b_0)$ in column A, and $W(b_0)$ is covered by $\theta(W(b_0))$. The following lemma shows that these conditions are also sufficient for the elimination of b_0.

LEMMA 7. *Suppose that ψ is a mapping from T to itself such that for all $x \notin W(b_0)$, $\psi(x) = x$. Then ψ is a containment mapping if and only if*

(1) *all the rows of $\psi(W(b_0))$ have the same symbol in column A, and*

(2) *for all $x \in W(b_0)$, x is covered by $\psi(x)$.*

Proof. By the above discussion, conditions (1) and (2) are satisfied if ψ is a containment mapping. Conversely, if ψ satisfies condition (2), then ψ satisfies the first two conditions of a containment mapping. If two rows x and w of T have the same nondistinguished variable b in some column B, then either both x and w are in $W(b_0)$ and b is b_0 (because T is the Class 1), or both x and w are not in $W(b_0)$. In either case, $\psi(x)$ and $\psi(w)$ have the same symbol in column B. Thus, ψ is a containment mapping. □

By Lemma 7, if b_0 is not essential, then we can always delete all the rows containing b_0 by finding a set of rows S, such that S covers $W(b_0)$ and all the rows of S have the same symbol in column A. By comparing each row with every other row, i.e., in $O(n^2)$ time, all repeated nondistinguished variables that are not essential can be deleted. The result is a quasi-minimal tableau that can be minimized in $O(n^2)$ time. Thus, we have the following theorem.

THEOREM 8. *A tableau T that has at most one repeated nondistinguished variable in each row can be minimized in $O(n^2)$ time.*

6.3. Minimizing tableaux of class 2. Let T be a tableau such that every row of T is covered by at most one row of T besides itself. For each row w of T, let $c(w)$ be the other row that covers w (if no other row of T covers w, then $c(w) = w$). The function c can be computed in $O(n^2)$ time. Note that every folding maps each row w to either $c(w)$ or itself.

Let b_0 be a repeated nondistinguished variable of T, and let $W(b_0)$ be the set of all the rows containing b_0. Suppose that there exists a folding ψ that eliminates b_0. Let $\bar{M}$ be the set of all rows w such that $\psi(w) = c(w)$ and $w \neq c(w)$. Clearly, $W(b_0) \subseteq \bar{M}$, since ψ is a folding onto a core that does not have b_0. We can simultaneously compute

$\bar{M}$ and the homomorphism h corresponding to ψ as follows. We initialize a set M to $W(b_0)$. Then the following rules are applied repeatedly to rows of M:

1. If a nondistinguished variable b appears in column A of some row $w \in M$, then $h(b)$ is the symbol appearing in column A of $c(w)$.

2. If b appears in some row $c(w)$, where $w \in M$, then $h(b) = b$.

3. If w is a row of T such that $w \notin M$ and w has a variable b for which $h(b)$ has already been determined and $h(b) \neq b$, then add w to M.

The first rule follows from the fact that every $w \in M$ is mapped to $c(w)$. The second rule is valid, since $c(w)$ is in the image of a folding and, hence, $c(w)$ must be mapped to itself. The third rule follows from the fact that if w contains a variable b such that $h(b) \neq b$, then $\psi(w) = c(w)$.

The above rules can be applied even if a folding that eliminates b_0 is not known to exist. The rules should be applied repeatedly until one of the following happens:

(i) The rules imply that $h(b_0) = b_0$ and, hence, b_0 is essential.

(ii) A contradiction is derived (i.e., for some b, the rules imply that $h(b)$ must be equal to two distinct variables) and, so, b_0 is essential.

(iii) The rules cannot be applied any more (and neither (i) nor (ii) has occurred). We will show that in this case b_0 can be eliminated.

Suppose that Case (iii) has occurred, and let $\bar{M}$ be the final value of M. Define a mapping θ as follows:

$$\theta(w) = \begin{cases} c(w) & \text{if } w \in \bar{M}, \\ w & \text{if } w \notin \bar{M}. \end{cases}$$

LEMMA 9. *The mapping θ is a containment mapping, and no row of $\bar{M}$ is in the image of θ (provided that neither Case* (i) *nor* (ii) *has occurred).*

Proof. For all rows w of T, row $c(w)$ covers w and, hence, θ satisfies the first two conditions of a containment mapping. Suppose that rows w and x of T have the same nondistinguished variable b in some column A. If both w and x are in $\bar{M}$, then $\theta(w)$ and $\theta(x)$ have the same symbol in column A (because the rules have not implied a contradiction). If both w and x are not in $\bar{M}$, then $\theta(w) = w$ and $\theta(x) = x$ and, so, $\theta(w)$ and $\theta(x)$ agree in column A. Suppose that exactly one of w and x is in $\bar{M}$. Therefore, $h(b) = b$ (otherwise, Rule 3 implies that all the rows containing b should be in $\bar{M}$) and, hence, $\theta(w)$ and $\theta(x)$ have b in column A. Thus, θ satisfies also the third condition of a containment mapping.

Suppose that a row w of $\bar{M}$ is mapped to some other row u of $\bar{M}$. Since u is in $\bar{M}$, it must have some repeated nondistinguished variable b such that $h(b) \neq b$ (otherwise, u neither contains b_0 nor could have been added to $\bar{M}$ by Rule 3). But since w is mapped to u, Rule 2 implies that $h(b) = b$. However, it is assumed that no contradiction has occurred and, hence, w cannot be mapped to u. □

It follows that if the rules imply neither a contradiction nor $h(b_0) = b_0$, then all the rows of $\bar{M}$ (and hence all the rows containing b_0) can be eliminated from T. If a contradiction is derived or $h(b_0) = b_0$, then b_0 is essential. Thus, by repeating this process for every nondistinguished variable in T, we obtain a quasi-minimal tableau $\bar{T}$ equivalent to T.

THEOREM 10. *A tableau T, in which each row is covered by at most one row of T besides itself, can be minimized in time $O(n^2)$.*

Proof. For each repeated nondistinguished variable b_0, we can test in linear time whether b_0 is essential. If it is not, then all the rows of $\bar{M}$ can be deleted from T. In $O(n^2)$ time we can repeat the process for every repeated nondistinguished variable. The result is a quasi-minimal tableau that can be minimized in $O(n^2)$ time by Lemma 6. □

6.4. Minimizing simple tableaux. Suppose that T is a simple tableau. Let S be a set of rows, and let w be a row of T. The *closure* of S with respect to w, denoted $\mathrm{CL}_w(S)$, is the minimal set of rows such that

(1) $S \subseteq \mathrm{CL}_w(S)$, and

(2) if x is a row in $\mathrm{CL}_w(S)$ such that x has a repeated nondistinguished variable b in some column A, and w has some other symbol in this column, then all the rows containing b are in $\mathrm{CL}_w(S)$.

In [3] it is shown that if w covers $\mathrm{CL}_w(S)$, then the tableau obtained by deleting all the rows of $\mathrm{CL}_w(S) - w$ is equivalent to T (this is true even if T is not simple). Further, if some repeated nondistinguished variable b of T is not essential, then there is a row w (that does not contain b) such that w covers $\mathrm{CL}_w(W(b))$. (Note that $w \notin \mathrm{CL}_w(W(b))$, since $w \notin W(b)$.)

These results can be used to obtain a quasi-minimal tableau equivalent to T as follows. Compute $\mathrm{CL}_w(W(b))$ for each repeated nondistinguished variable b and each row w that does not contain b. If w covers $\mathrm{CL}_w(W(b))$, then delete all the rows of $\mathrm{CL}_w(W(b))$ from T. The resulting tableau is equivalent to T, and it is quasi-minimal [3]. In this section we describe an implementation of this algorithm that runs in $O(n^2)$ time.

LEMMA 11. *Let w be a row of a simple tableau T. Suppose that b_1 and b_2 are two repeated nondistinguished variables of T that do not occur in w. Then $\mathrm{CL}_w(W(b_1))$ and $\mathrm{CL}_w(W(b_2))$ are either equal or disjoint.*

Proof. Suppose that $x \in \mathrm{CL}_w(W(b_1))$. By the definition of $\mathrm{CL}_w(W(b_1))$, row x has a repeated nondistinguished variable b that does not appear in w. Further, part (2) of the definition is reversible, i.e., if y in $\mathrm{CL}_w(W(b_1))$ implies that z is also in $\mathrm{CL}_w(W(b_1))$, then z in $\mathrm{CL}_w(W(b_1))$ implies that y is also in $\mathrm{CL}_w(W(b_1))$. Therefore, $\mathrm{CL}_w(W(b_1)) = \mathrm{CL}_w(W(b))$. It follows that if $\mathrm{CL}_w(W(b_1))$ and $\mathrm{CL}_w(W(b_2))$ have a row x in common, then they are equal. □

Lemma 11 implies that if $\mathrm{CL}_w(W(b))$ has been computed, and $\bar{b}$ is a repeated nondistinguished variable that appears in some row of $\mathrm{CL}_w(W(b))$, then there is no need to compute $\mathrm{CL}_w(W(\bar{b}))$ (it is assumed that neither b nor $\bar{b}$ occurs in w). Thus, for each row w we do the following. At first all repeated nondistinguished variables that do not appear in w are marked "unconsidered". The next step is to compute $\mathrm{CL}_w(W(b))$ for some repeated nondistinguished variable that is marked "unconsidered". During this step all repeated nondistinguished variables that occur in some row of $\mathrm{CL}_w(W(b))$ are marked "considered". If $\mathrm{CL}_w(W(b))$ is covered by w, then all the rows of $\mathrm{CL}_w(W(b))$ are deleted. This step is repeated for some other variable that is marked "unconsidered", until all the variables are marked "considered". The complete procedure is described in Algorithm 1.

ALGORITHM 1

```
procedure CLOSURE (b, w):
begin
(1)  S := φ;
(2)  make QUEUE empty;
(3)  mark b "considered";
(4)  add all the rows containing b to QUEUE;
(5)  while QUEUE is not empty do
       begin
(6)      let v be the first row on QUEUE;
(7)      move v from QUEUE to S;
(8)      for every column A do
```

```
(9)          if v and w disagree in column A, v has a repeated
               nondistinguished variable d in this column,
               and d is marked "unconsidered" then
               begin
(10)             mark d "considered";
(11)             for every row x containing d do
(12)               if x is neither in S nor on QUEUE
(13)                 then add x to QUEUE;
               end;
        end;
(14)  return S;
    end
    begin/* main procedure */
(15)  for every row w do
        begin
(16)      mark all repeated nondistinguished variables "unconsidered";
(17)      for every repeated nondistinguished variable d
            that occurs in w do
(18)        mark d "considered";
(19)      while there is a repeated nondistinguished variable b
              marked "unconsidered" do
            begin
(20)          R := CLOSURE (b, w);
(21)          if w covers R then delete all the rows of R from T;
            end;
        end;
(22)  return T;
    end
```

THEOREM 12. *A simple tableau T can be minimized in $O(n^2)$ time.*

Proof. By Lemma 11 and [3], Algorithm 1 returns a quasi-minimal tableau $\bar{T}$ equivalent to T. Consider the time complexity of this algorithm. Each call CLOSURE (b, w) is done in $O(|\mathrm{CL}_w\ (W(b))|)$ time and, hence, the cost of executing the loop of lines (15)–(21) once is $O(n)$. Thus, the total cost of the algorithm is $O(n^2)$. By Lemma 6, $\bar{T}$ can be minimized in $O(n^2)$ time. □

THEOREM 13. *If T_1 and T_2 are simple tableaux, then testing whether T_1 is equivalent to T_2 can be done in $O(n^2)$ time.*

Proof. By using Algorithm 1, we compute quasi-minimal tableaux $\bar{T}_1$ and $\bar{T}_2$ equivalent to T_1 and T_2, respectively. It follows from [3] that testing whether $\bar{T}_1$ is equivalent to $\bar{T}_2$ can be done in $O(n^2)$ time, where n is the size of T_1 and T_2. □

7. Decomposition of tableaux. Let T be a tableau that does not necessarily belong to one of the classes we have discussed so far. None of the three minimization algorithms can minimize T in polynomial time, but they can be used as heuristics. The minimization algorithm for simple tableaux can be applied to any tableau, and the result is an equivalent tableau possibly with fewer rows. The idea behind the algorithm of § 6.2 can be used to reduce the number of rows in a tableau T as follows. Let b be a repeated nondistinguished variable of T, and suppose that the set $W(b)$ of all the rows containing b does not have any other repeated nondistinguished variable. Then all the rows of $W(b)$ can be deleted if they are covered by a set of rows S that have the same symbol in the column of b.

The algorithm of § 6.3 is not only good as a heuristic, but can also be used as an exponential time minimization algorithm for tableaux. Let T be a tableau. For every row i of T, we define $C(i)$ to be the set of all the rows that cover row i, i.e.,

$$C(i) = \{j | j \neq i \text{ and row } j \text{ covers row } i\}.$$

Suppose we construct a function c such that for all i, $c(i) \in C(i)$ (it is understood that $c(i) = i$ if $C(i) = \phi$). Using c we can apply the algorithm of § 6.3 as a heuristic. The number of all possible c's is exponential only in the number of rows i for which $C(i)$ has more than one element. If we execute the algorithm once for each possible c we are guaranteed to minimize T, since at least one of these c's corresponds to a folding from T to its core.

The following approach can be used to further reduce the exponential factor in the running time of this algorithm. We define a relation R on the rows of a tableau T as follows. For rows x and y of T, xRw if and only if x and w have the same nondistinguished variable in some column. Obviously, R is symmetric and reflexive. Let $P_1, P_2, \cdots, P_q$ be the equivalence classes of the transitive closure of R.

LEMMA 14. *Let θ be a containment mapping from a tableau T to itself, and let k $(1 \leqq k \leqq q)$ be a fixed integer. Define*

$$\xi(x) = \begin{cases} \theta(x) & \text{if } x \in P_k, \\ x & \text{otherwise.} \end{cases}$$

Then ξ is a containment mapping.

Proof. Obviously, for all rows x, $\xi(x)$ covers x. Suppose that rows x and w have the same nondistinguished variable in column A. Thus, either both w and x are in P_k or both are not in P_k. In either case, $\xi(x)$ and $\xi(w)$ have the same symbol in column A. □

Lemma 14 implies that when the algorithm of § 6.3 is applied as a heuristic, we have to consider only c's such that for some fixed k, $c(i) \in C(i)$ if $i \in P_k$; otherwise $c(i) = i$. If among all the P_j's, P_{j_0} has a maximum number, say m, of rows for which $C(i)$ contains more than one element, then the number of all possible c's is exponential only in m. For each P_j there is at least one possible c that maps all redundant rows in P_j to the core of T. Thus, no c has to be considered more than once.

8. Synthesis of expressions from tableaux. Given an expression E over select, project and join, we can construct an equivalent tableau T in polynomial time [3]. However, deciding whether a tableau T has an equivalent expression E is NP-complete [19] (the problem is NP-complete even if T is minimal).

The optimization of a relational query is carried out in two steps. First, we minimize the number of joins (by minimizing the corresponding tableau). This step has the effect of eliminating redundant subexpressions. In the second step, we produce from the minimal tableau an equivalent expression in which select and project are applied as early as possible. The approach taken in the second step was found useful by previous workers (e.g., [12], [17]) in the field of expression optimization, and can be viewed as our "cost function." In the case of simple tableau, the second step can also be done in $O(n^2)$ time [5].

In order to obtain an optimal expression for a minimal tableau T, [5] uses the following approach. At first all constants are deleted from the summary of T. Then an optimal expression E is synthesized. Finally a new operator *augment*, defined by $\alpha_{A=c}(r) = \{\mu | \mu(A) = c$ and there exists ν in r such that for all attributes B in the relation scheme of r, $\mu(B) = \nu(B)\}$, is applied to E to introduce the constants that were deleted from the summary of T.

Suppose we decompose a tableau T into equivalence classes $P_1, P_2, \cdots, P_q$ as described in § 7, and let s be the summary of T. For every j, we can define a tableau T_j that has the same rows as P_j and a summary as follows. For each column A, if s has a nonblank symbol in column A and that symbol appears also in column A of P_j, then the summary of T_j has the same symbol in column A; otherwise, it has a blank. It follows that $T = \bowtie_{j=1}^{q} T_j$. Thus, T comes from an expression if and only if each T_j comes from an expression. Suppose that each T_j comes from the expression E_j. Then T corresponds to the expression $E = \bowtie_{j=1}^{q} E_j$. If T is a minimal tableau, we can get an optimal expression for T as follows. At first all constants are deleted from the summary of T. Then we decompose T and find an optimal expression E_j for each T_j. (Note that each T_j is minimal.) Let $E = \bowtie_{j=1}^{q} E_j$. By applying augmentation to E we get an optimal expression for T.

If T is a tableau with at most one repeated nondistinguished variable in each row, then each T_j is a simple tableau (because it has at most one repeated nondistinguished variable). Thus, we can synthesize an optimal expression for T in polynomial time. However, if T is a tableau in which every row is covered by at most one row besides itself, then testing whether T comes from an expression is NP-complete.[3]

THEOREM 15. *Let T be a tableau in which every row is covered by at most one row besides itself. The problem whether T corresponds to a restricted relational expression is* NP-*complete.*

Proof. The problem is in NP, since we only have to guess an expression E, construct its corresponding tableau as in [3], and check that it is identical to T (after possibly renaming some variables).

In order to show that the problem is NP-complete, the problem of testing whether a tableau T corresponds to an expression (shown NP-complete in [19]) is reduced to this problem as follows. Let T be a tableau, and let G be a new attribute. We construct a tableau T' by adding to T a new column that corresponds to G. The summary of T' has a blank, and row i has the constant i in this column. We claim that T corresponds to an expression if and only if T' corresponds to an expression.

If. Let E' be an expression for T'. Delete G from each relation scheme in E', and delete every selection operator that is applied to the attribute G. Each projection operator π_X that appears in E' is replaced with $\pi_{X-\{G\}}$. Let E be the resulting expression. It is easy to show that E corresponds to T.

Only if. Let E be an expression for T. Let R_i be the relation scheme that corresponds to row i of T, and define $R_i' = R_i \cup \{G\}$ for every i. An expression E' for T' is obtained from E by replacing each R_i with $\pi_{R_i}(\sigma_{G=i}(R_i'))$.

Since each row of T' is covered by no other row of T' besides itself, the proof is complete. □

Acknowledgment. The author is grateful to an anonymous referee for useful suggestions.

REFERENCES

[1] A. V. AHO, C. BEERI AND J. D. ULLMAN, *The theory of joins in relational databases*, ACM Trans. Database Systems, 4 (1979), pp. 297–314.

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

[3] A. V. AHO, Y. SAGIV AND J. D. ULLMAN, *Equivalences among relational expressions*, this Journal, 8 (1979), pp. 218–246.

[3] See [2], [11] for an exposition of NP-completeness and related topics.

[4] A. V. AHO, Y. SAGIV AND J. D. ULLMAN, *Efficient optimization of a class of relational expressions*, ACM Trans. Database Systems, 4 (1979), pp. 435–454.

[5] A. V. AHO, Y. SAGIV, T. G. SZYMANSKI AND J. D. ULLMAN, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions*, this Journal, 10 (1981), pp. 405–421.

[6] W. W. ARMSTRONG, *Dependency structures of data base relationship*, in Proc. 1974 International Federation for Information Processing Congress, North-Holland, Amsterdam, 1974, pp. 580–583.

[7] C. BEERI, R. FAGIN AND J. H. HOWARD, *A complete axiomatization for functional and multivalued dependencies*, in Proc. ACM-SIGMOD International Conference on the Management of Data, Toronto, Canada, August, 1977, pp. 47–61.

[8] A. K. CHANDRA AND P. M. MERLIN, *Optimal implementation of conjunctive queries in relational data bases*, in Proc. Ninth Annual ACM Symposium on Theory of Computing, Boulder, Colorado, May, 1977, pp. 77–90.

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

[10] ———, *Relational completeness of data base sublanguages*, in Data Base Systems, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65–98.

[11] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1978.

[12] P. A. V. HALL, *Optimization of a single relational expression in a relational database system*, IBM J. Res. Develop. 20 (1976), pp. 244–257.

[13] J. MINKER, *Performing inferences over relational databases*, in Proc. ACM-SIGMOD International Conference on Management of Data, San Jose, CA, May, 1975, pp. 79–91.

[14] F. P. PALERMO, *A database search problem*, in Information Systems COINS IV, J. T. Tou, ed., Plenum Press, New York, 1974.

[15] R. M. PECHERER, *Efficient evaluation of expressions in a relational algebra, in* Proc. ACM Pacific Conference, San Francisco, April, 1975, pp. 44–49.

[16] Y. SAGIV AND M. YANNAKAKIS, *Equivalences among relational expressions with the union and difference operators*, J. Assoc. Comput. Mach., 27 (1980), pp. 633–655.

[17] J. M. SMITH AND P. Y.-T. CHANG, *Optimizing the performance of a relational algebra database interface*, Comm. ACM, 18 (1975), pp. 468–579.

[18] E. WONG AND K. YOUSSEFI, *Decomposition—A strategy for query processing*, ACM Trans. Database Systems, 1 (1976), pp. 223–241.

[19] M. YANNAKAKIS AND C. H. PAPADIMITRIOU, *Algebraic dependencies*, in Proc. 21st Symposium on Foundations of Computer Science, Syracuse, N.Y., October, 1980, pp. 328–332.

[20] M. M. ZLOOF, *Query-by-example: The invocation and definition of tables and forms*, in Proc. ACM International Conference on Very Large Data Bases, New York, September, 1975, pp. 1–24.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0009 $01.25/0

THE WORST AND THE MOST PROBABLE PERFORMANCE OF A CLASS OF SET-COVERING ALGORITHMS*

V. LIFSCHITZ† AND B. PITTEL‡

In memory of Y. D. Burtin

Abstract. Let $I=(1,\cdots,m)$, $J=(1,\cdots,n)$ and $\Delta=(D_i)_{i\in I}$ be a family of subsets D_i of J. A class of algorithms which find a minimum number of D_i's covering $D=\cup_{i\in I}D_i$ is studied. A measure $T(\Delta)$ of the computation time is shown to grow exponentially with the size of the problem in the worst case, namely $\max_\Delta T(\Delta)>(4^{1/5})^{\min(m,n')}>1.319^{\min(m,n')}$, $n'=|D|$. For $m=n'$ and a large subclass of algorithms, an estimate $\max_\Delta T(\Delta)<(3/4^{1/3})^m<(1.890)^m$ is established, so they always perform better than the obvious trivial procedure. Let, on the other hand, Δ be chosen at random. Under condition $\ln n/\ln m\to\gamma\in(0,\infty)$, it is proven that

$$P(m^{c_1(\gamma)\ln m}\leqq T(\Delta)\leqq m^{c_2(\gamma)\ln m})\to 1.$$

Hence, asymptotically almost certainly, the computation time is of a considerably lower order than that in the worst case, but it is still far from being polynomially bounded.

Key words. algorithmic analysis, complexity, worst case, probable behavior, random trees, asymptotical estimates

1. Introduction.

Consider a finite family $\Delta=(D_i)_{i\in I}$ of finite sets, and let $D=\cup_{i\in I}D_i$. A Δ-*cover* is any set $X\subset I$ such that

$$\bigcup_{i\in X} D_i = D.$$

The *minimum cover problem* calls for finding a Δ-cover of minimum cardinality. It is known to be NP-complete [8], [7], unless all D_i's have $|D_i|\leqq 2$, in which case it can be solved in polynomial time by matching techniques [15], [4], [9]. Hence, it is highly unlikely that there will be found a polynomial time algorithm for the general case. Still, many attempts have been made to develop practically working algorithms. One of them [5] is based on the reduction of a given problem to one or two problems of a smaller size. By iterating this procedure, every problem can be reduced to trivial problems with $D=\varnothing$. A branching type reduction is performed only if other kinds of reductions are not applicable, and this restriction actually defines a whole class of set covering algorithms, each of them to be specified by additional conditions.

We are interested in the computation time of these algorithms, which can be defined as the total number $T(\Delta)$ of reduction steps needed to find a minimum cover for Δ. We show that the worst case computation time of the algorithms in question grows *exponentially* with the size of the problem: if $|I|=m$, $|D|=n$ then

$$\max_\Delta T(\Delta)\geqq(4^{1/5})^{\min(m,n)}>1.319^{\min(m,n)}.$$

Avis [1] studied more elaborate set covering algorithms due to Chvátal [3]. For a special Steiner type of subsets D_i suggested first by Fulkerson et al. [6], he proved that these algorithms have a computation time of the order $2^{\sqrt{2n/3}}$.

* Received by the editors July 9, 1981, and in final revised form July 7, 1982.

† Department of Mathematical Sciences, University of Texas at El Paso, El Paso, Texas 79968. The research of this author was supported by the National Science Foundation under grant MCS 8002442.

‡ Department of Mathematics, Ohio State University, Columbus, Ohio 43210. The research of this author was supported by the National Science Foundation under grant MCS 8002966.

To brighten things up, we prove that, for $m = n$ and a large subclass of the algorithms,

$$\max_{\Delta} T(\Delta) < (3/4^{1/3})^m < (1.890)^m.$$

Thus, these algorithms always perform better than the trivial procedure based on the examination of all 2^m subsets of Δ. (Another result of this kind which we are aware of was obtained by Tarjan and Trojanowski [17] in connection with the problem of finding a maximal independent set of a graph.) We conjecture that the same nice property holds when n is allowed to grow not faster than a polynomial function of m. If no restriction is imposed on n then the situation is considerably worse: for every m there is a Δ with

$$n = \binom{m}{1+[(m-1)/2]}$$

for which

$$T(\Delta) = \binom{m-1}{[(m-1)/2]} \sim cm^{-1/2} \cdot 2^m.$$

Suppose now $D_1, \cdots, D_m$ are subsets chosen at random, and independently of each other, from the whole collection of 2^n subsets of a given n-element set. (Thus, there is a positive probability that not all of $D_1, \cdots, D_m$ are distinct or (and) that some of them are empty. Under conditions given below, this probability tends to zero as $m, n \to \infty$.)

Then, for each set covering algorithm in question, the computation time $T(\Delta)$ becomes a random variable. What are its typical values? How do they compare with that of the worst case? Let $m, n \to \infty$, so that $\ln n (\ln m)^{-1} \to \gamma \in (0, \infty)$. Then there exist two constants $c_1(\gamma)$, $c_2(\gamma)$ such that, for each algorithm,

$$P(m^{c_1(\gamma) \ln m} \leqq T(\Delta) \leqq m^{c_2(\gamma) \ln m}) \to 1.$$

Hence, asymptotically almost certainly, the computation time is of a considerably lower order than that of the worst case, but is still far from being polynomially bounded. Our conjecture is that, under the above condition, $\ln T(\Delta)/\ln^2 m \to C$ in probability, where C is a constant possibly dependent on the choice of the algorithm.

This paper contributes to the investigation of the worst and stochastic behavior of algorithms for exponentially hard to solve problems [2], [10], [11], [12], [13], [16], [17]. A common difficulty is the necessity to analyze the distribution of the computation time whose typical values rapidly grow with the size of the problem. It is our hope that the methods of this paper can be used for other problems of this class.

2. A class of set covering algorithms and their worst-case behavior. Our problem is to find a set $X \subset I$ of minimum cardinality such that

$$\bigcup_{i \in X} D_i = D, \qquad \left(D = \bigcup_{i \in I} D_i\right). \tag{2.1}$$

If $D = \varnothing$ then the only solution is $X = \varnothing$. Otherwise a reduction of one of the following four types can be performed.

1. *Elimination of a redundant subset.* If $D_{i_1} \subset D_{i_2}$ for some $i_1 \neq i_2$ then every Δ-cover which includes i_1 can have i_2 instead, so that D_{i_1} can be eliminated from consideration. In other words, Δ can be replaced by

$$\Delta' = (D_i)_{i \in I \setminus \{i_1\}}. \tag{2.2}$$

2. *Elimination of a redundant element.* Let $j_1 \neq j_2 \in D$; if j_2 majorizes j_1, i.e., $j_1 \in D_i$ implies $j_2 \in D_i$ for every i, then every subfamily of Δ covering j_1 covers j_2, so Δ can be replaced by

$$\Delta' = (D_i \backslash \{j_2\})_{i \in I}. \tag{2.3}$$

3. *Inclusion of a necessary subset.* If for some $j_1 \in D$ and $i_1 \in I$

$$j_1 \notin \bigcup_{i \neq i_1} D_i, \tag{2.4}$$

then every Δ-cover includes i_1. Thus it suffices to find a minimum cover for

$$\Delta' = (D_i \backslash D_{i_1})_{I \in I \backslash \{i_1\}}; \tag{2.5}$$

if X is a minimum Δ'-cover then $X \cup \{i_1\}$ is a minimum Δ-cover.

4. *Branching.* If reductions of types 1–3 do not apply then we choose an $i_1 \in I$ and examine separately X's with $i_1 \notin X$ and with $i_1 \in X$. To this end, we find solutions X', X'' of the problems

$$\Delta' = (D_i)_{i \in I \backslash \{i_1\}}, \qquad \Delta'' = (D_i \backslash D_{i_1})_{i \in I \backslash \{i_1\}}. \tag{2.6}$$

Notice that the union of Δ' is D, since otherwise D_{i_1} would be necessary and branching would not apply. Hence X' is a Δ-cover not including i_1 which has the minimum possible cardinality. The union of Δ'' is $D \backslash D_{i_1}$ and $X'' \cup \{i_1\}$ is a Δ-cover including i_1 with the minimum possible cardinality. Set $X = X'$ or $X = X'' \cup \{i_1\}$ depending on which of these sets has fewer elements. It is convenient to represent Δ by the Boolean matrix with the columns corresponding to the elements of D, the rows corresponding to the D_i's, and the elements in each subset specified by the 1's in the proper column. This matrix will have no zero columns. It is easy to see that every reduction step corresponds to the elimination of some rows and columns.

Example. The family $\Delta = (\{1, 2\}, \{1, 3\}, \{2, 3\})$ can be represented by the matrix

$$\begin{array}{c} \\ D_1 \\ D_2 \\ D_3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{array}.$$

No elements or subsets can be eliminated and no subset is necessary, so that branching applies. If $i_1 = 1$ then we get these two matrices:

$$\begin{array}{c} \\ D_2 \\ D_3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ D_2 \\ D_3 \end{array} \begin{array}{c} \begin{array}{c} 3 \end{array} \\ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{array}.$$

The second matrix leads, after eliminating D_2 and including D_3, to $X'' = \{3\}$. In the first, the third column is redundant, and we get

$$\begin{array}{c} \\ D_2 \\ D_3 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{array}.$$

Here D_2 is necessary, and in the resulting matrix

$$\begin{array}{c} \\ D_3 \end{array} \begin{array}{c} \begin{array}{c} 2 \end{array} \\ [1] \end{array}$$

D_3 is necessary, so that $X' = \{2, 3\}$. Both X' and $X'' \cup \{i_1\} = \{1, 3\}$ are two-element,

therefore each of them is a minimum Δ-cover. (We would find the third minimum Δ-cover $\{1, 2\}$ if, when solving the second of the two problems obtained by branching, we eliminated D_3 instead of D_2.)

Our only restriction on the order of reductions (viz., that branching is not used unless necessary) does not define the choice of reductions completely. Every particular order of reductions can be described by a tree with a submatrix of Δ assigned to each vertex; for instance, the example given above can be depicted as in Fig. 1.

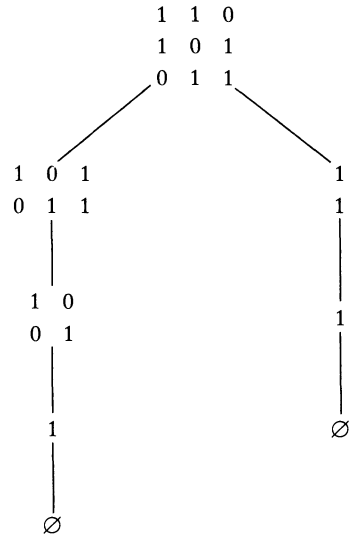


FIG. 1

A tree of this type will be called a Δ-*tree*. In every Δ-tree τ, Δ is assigned to the root, and the 0×0 matrix to every leaf. No matrix in τ has zero columns. Every edge of τ leads to a matrix obtained by eliminating at least one row and/or at least one column. Of the two matrices obtained by branching, we shall show the one in which a row is eliminated on the left, and the one in which, in addition, columns are eliminated, on the right.

We shall study the size of Δ-trees, which characterizes the computation time of corresponding computational processes. For instance, the number of vertices $|\tau|$ of a Δ-tree τ equals the number of recursive calls of the corresponding procedure. Another quantity describing the size of τ is the number of leaves (or branches) $L(\tau)$, which also equals the number of branchings plus 1. If $|I| = m$, $|D| = n$ (so that Δ is an $m \times n$ matrix) then

$$L(\tau) \leqq |\tau| \leqq (m+n+1)L(\tau),$$

because every branch consists of at most $m+n$ edges.

The following theorem shows that, in the absence of restrictions on the size of D, $L(\tau)$ may grow about as fast as 2^m, so that the computation time of the algorithm in question is as large, in the worst case, as that of the trivial procedure based on the examination of all 2^m subsets of Δ.

THEOREM 1. (a) *For every* Δ-*tree* τ *with* $m = |I| \geqq 2$,

$$L(\tau) \leqq 2^{m-2}. \tag{2.7}$$

(b) *For every m there is Δ with $|I| = m$ such that for every Δ-tree τ*

$$L(\tau) = \left\lceil \begin{pmatrix} m-1 \\ \lceil m-1 \rceil \\ 2 \end{pmatrix} \right\rceil. \tag{2.8}$$

Proof. Part (a) is proved by induction on m. If $m \leqq 2$, then every column either is necessary or majorizes another column, so that branching does not apply, and $L(\tau) = 1$. If $m > 2$ then the uppermost branching leads to two matrices with at most $m-1$ rows, so that the number of branches at most doubles.

To prove part (b), consider the matrix $\varphi_{m,k}$ $(1 \leqq k \leqq m)$ whose columns are all Boolean vectors of length m with exactly k nonzero elements. For instance, Δ of the example is $\varphi_{3,2}$. It suffices to prove the following lemma:

LEMMA 1. *For every $\varphi_{m,k}$-tree τ,*

$$L(\tau) = \binom{m-1}{k-1}.$$

Proof. By induction on m. *Basis*: $m = 1$, $k = 1$; for the only $\varphi_{1,1}$-tree τ, $L(\tau) = 1$.

Induction step: $m > 1$. If $k = 1$ or $k = m$ then, for every $\varphi_{m,k}$-tree τ, $L(\tau) = 1$. Let $1 < k < m$. Then in any $\varphi_{m,k}$-tree τ, the first reduction is a branching. Assume for definiteness that $i_1 = 1$. After a rearrangement of columns, $\varphi_{m,k}$ may be represented in the form

$$\Delta = \begin{array}{|c|c|} \hline 0 \cdots 0 & 1 \cdots 1 \\ \hline \varphi_{m-1,k} & \varphi_{m-1,k-1} \\ \hline \end{array}.$$

Consider the result of the first branching. The left edge leads to

$$\Delta' = \begin{array}{|c|c|} \hline \varphi_{m-1,k} & \varphi_{m-1,k-1} \\ \hline \end{array},$$

the right edge to $\varphi_{m-1,k}$. By the hypothesis of induction, the latter gives $\binom{m-2}{k-1}$ branches. To describe the tree generated by the former, consider two cases.

Case 1. $k = 2$. Then $\varphi_{m-1,k-1}$ is the $(m-1) \times (m-1)$ unity matrix, and each column of $\varphi_{m-1,k}$ majorizes some of the columns of $\varphi_{m-1,k-1}$. Thus two types of reductions are possible: the elimination of a column of $\varphi_{m-1,k}$ and the inclusion of a row of Δ'. Row eliminations are impossible and will be impossible at every stage of the process, because what remains of $\varphi_{m-1,k-1}$ is always a unity matrix. Thus the matrix in question gives one branch, in which all columns of $\varphi_{m-1,k}$ are eliminated and all rows of Δ' included.

Case 2. $k > 2$. Then the only possible reductions are the eliminations of the columns of $\varphi_{m-1,k}$, so that the left branch leads to $\varphi_{m-1,k-1}$, which, by the hypothesis of induction, gives $\binom{m-2}{k-2}$ branches. This formula clearly works in Case 1 too, so in both cases the total number of branches of τ is

$$\binom{m-2}{k-1} + \binom{m-2}{k-2} = \binom{m-1}{k-1}.$$

This concludes the proof of Lemma 1 and of Theorem 1. □

Theorem 1 estimates the maximum possible size of Δ-trees under the assumption that n, the number of columns of Δ, is not restricted. In fact, the number of columns of the matrix used in the proof of Theorem 1(b) grows exponentially with m. What can be said about the size of Δ-trees if $n = O(m)$, e.g., if Δ is a square matrix?

We begin by constructing examples of square matrices with large Δ-trees.

LEMMA 2. *Let*

$$\Delta = \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2 \end{bmatrix}.$$

If $L(\tau_1) = a$, $L(\tau_2) = b$ for every Δ_1-tree τ_1 and every Δ_2-tree τ_2, then $L(\tau) = a \cdot b$ for every Δ-tree τ.

Proof. By induction on $|\tau|$. If $|\tau| = 1$ then each of Δ_1, Δ_2, Δ is the 0×0 matrix, and $a = b = a \cdot b = 1$. Let $|\tau| > 1$. Consider the first reduction in τ.

Case 1. This is not branching. The result Δ' of that reduction has the form

$$\begin{bmatrix} \Delta_1' & 0 \\ 0 & \Delta_2 \end{bmatrix} \text{ or } \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2' \end{bmatrix},$$

where Δ_i' can be obtained from Δ_i by a reduction of the same type. Assume for definiteness that $i = 1$. For every Δ_1'-tree τ_1', $L(\tau_1') = a$. If τ' is the Δ'-tree obtained from Δ by removing the root and the uppermost edge then $L(\tau) = L(\tau')$, and, by the induction hypothesis, $L(\tau') = a \cdot b$.

Case 2. The first reduction is branching. Then the matrices Δ', Δ'' which result from this branching have the form

$$\begin{bmatrix} \Delta_1' & 0 \\ 0 & \Delta_2 \end{bmatrix}, \quad \begin{bmatrix} \Delta_1'' & 0 \\ 0 & \Delta_2 \end{bmatrix}$$

or

$$\begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2' \end{bmatrix}, \quad \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2'' \end{bmatrix},$$

where Δ_i', Δ_i'' can be obtained from Δ_i by branching. Assume again $i = 1$. For all Δ_1'-trees τ_1', $L(\tau_1')$ has the same value a' (otherwise there would exist Δ_1–trees τ_1 with different values of $L(\tau_1)$). Similarly, for all Δ_1''-trees τ_1'' $L(\tau_1'')$ has the same value a'', and $a' + a'' = a$. If τ' and τ'' are the Δ'-tree and the Δ''-tree obtained from Δ by removing the root and the uppermost edges, then $L(\tau) = L(\tau') + L(\tau'')$, and, by the induction hypothesis, $L(\tau') = a' \cdot b$, $L(\tau'') = a'' \cdot b$. Hence $L(\tau) = a' \cdot b + a'' \cdot b = (a' + a'')b = a \cdot b$. □

Lemma 2 suggests a way to construct a sequence of square matrices $\Delta_1, \Delta_2, \cdots$ such that the sizes of Δ_i-trees grow rapidly with the size of Δ_i. Let Δ_1 be an $m_1 \times m_1$ matrix, and $L(\tau_1) = a$ for every Δ_1-tree τ_1. Define Δ_i to be the $m \times m$ matrix, $m = im_1$, consisting of i blocks equal to Δ_1. By Lemma 2, for every Δ_i-tree τ

$$L(\tau) = a^i = (a^{1/m_1})^m.$$

Hence Δ_1 should be chosen in such a way that a^{1/m_1} is as large as possible.

Since $\varphi_{m_1, m_1 - 1}$ is an $m_1 \times m_1$ matrix, we can use it as Δ_1 in this construction. In this case

$$a = \binom{m_1 - 1}{m_1 - 2} = m_1 - 1, \qquad a^{1/m_1} = (m_1 - 1)^{1/m_1}.$$

The fastest growing $L(\tau)$ corresponds to the value $m_1 = 5$, for which $(m-1)^{1/m}$ is maximum. Thus we have proved:

THEOREM 2. *Let $m = 5i$, $i \geqq 0$. There exists an $m \times m$ matrix Δ_m such that for every Δ_m-tree τ*

$$L(\tau) = (4^{1/5})^m. \tag{2.9}$$

COROLLARY. *For m, $n \geqq 5$, there exists an $m \times n$ matrix Δ such that for every Δ-tree τ*

$$L(\tau) \geqq (4^{1/5})^{\min(m,n)}. \tag{2.10}$$

Proof. Consider the problem Δ represented by the $m \times n$ matrix

$$\left[\begin{array}{c|c} 1 & 0 \\ \hline & \Delta_\nu \end{array}\right]$$

where $\nu = [\min(m, n)/5]$, Δ_ν is introduced in the proof of Theorem 2 and 1 and 0 are matrices with one and zero entries. Clearly, every Δ-tree consists of the $(m+n-2\nu)$-long vertical branch with a Δ_ν-tree attached to the endpoint of the branch. Thus, every Δ-tree has $(4^{1/5})^\nu$ leaves. □

The constant $4^{1/5}$ can be replaced by a larger one if we want *at least one* Δ-tree to be large, not necessarily *all* of them (in other words, if we want to give an example of Δ for which at least one order of reductions leads to a long computation).

THEOREM 3. *Let $m = 7i$, $i \geqq 0$. There exists an $m \times m$ matrix Δ such that for some Δ-tree τ*

$$L(\tau) = (10^{1/7})^m. \tag{2.11}$$

Proof. Let

$$\Delta_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

There exists a Δ_1-tree τ_1 with ten branches as shown in Fig. 2. (In this tree we always use the first row for branching, and the rows and columns that are eliminated are marked by asterisks.)

Let now Δ_i be the $m \times m$ matrix consisting of i blocks equal to Δ_1. Construct the Δ_i-tree τ_i as follows: apply the reductions shown in the Δ_1-tree above to the first block, thus arriving at ten matrices equal to Δ_{i-1}, and then continue reducing each of those matrices in a similar fashion. Clearly,

$$L(\tau_i) = 10 \cdot L(\tau_{i-1}).$$

Since $L(\tau_1) = 10$, it follows that

$$L(\tau_i) = 10^i = 10^{m/7} = (10^{1/7})^m.$$

Theorem 3 is proved. □

Since

$$4^{1/5} \approx 1.3195, \qquad 10^{1/7} \approx 1.3895,$$

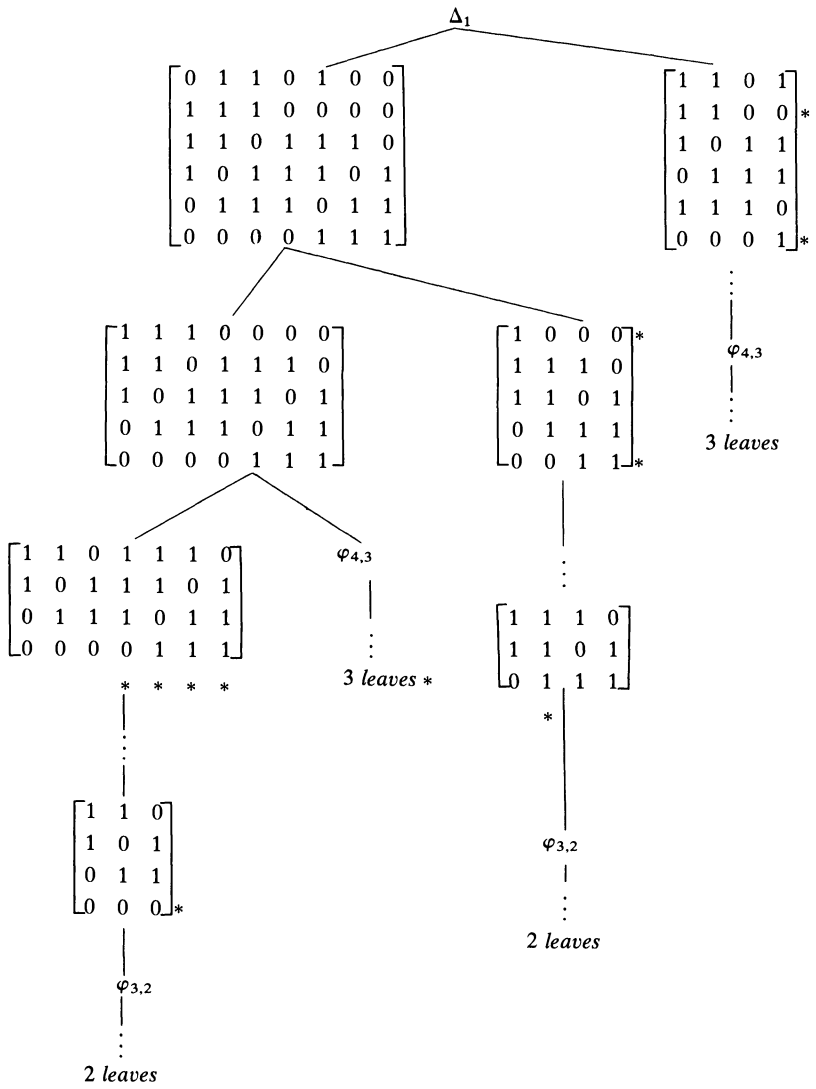


FIG. 2

Theorems 2 and 3 show the following: there exist large $m \times m$ matrices Δ such that, for every Δ-tree τ, $L(\tau) > 1.319^m$, there exist large $m \times m$ matrices Δ such that, for some Δ-tree τ, $L(\tau) > 1.389^m$.

Our next goal is to show that for every sufficiently large $m \times m$ matrix Δ there exists a Δ-tree τ with $L(\tau) < 1.9^m$ so that the base 1.319 in the estimate above cannot be replaced by 1.9.

To this end, we shall estimate first the size of Δ-tree for matrices Δ with at most 2 nonzero elements in any row (in terms of families of sets, with $|D_i| \leqq 2$ for each $i \in I$). The minimum cover problem for such matrices has a simple graph-theoretic interpretation. Without restricting generality, we can assume that the D_i's are distinct, and for every i $|D_i| = 2$. Then the D_i's can be considered as the edges of a graph with the set of vertices D. The minimum cover problem can be stated as the problem of finding a minimum edge cover of that graph.

LEMMA 3. *Let every row of* Δ *contain at most two nonzero elements. For every* Δ*-tree* τ,

$$L(\tau) \leqq \lambda^m, \tag{2.12}$$

where $m = |I|$, *and* λ *is the root of the equation* $\lambda^5 - \lambda^4 - 1 = 0$.

Proof. By induction on the size of τ. If $|\tau| = 1$ then $m = 0$, and $L(\tau) = \lambda^0 = 1$. If $|\tau| > 1$ then consider two cases.

Case 1. The first reduction is something other than branching. Let τ' be obtained from τ by deleting the root and the uppermost edge. Then $L(\tau) = L(\tau')$ and by the induction hypothesis, $L(\tau') \leqq \lambda^m$.

Case 2. The first reduction is branching. Then every row of Δ has exactly two nonzero elements (a row with <2 nonzero elements either is necessary or is majorized by another row), and every column of Δ has at least two nonzero elements (otherwise Δ would have a necessary row). Let τ', τ'' be obtained from τ by deleting the root and the uppermost edges, and let Δ', Δ'' be their roots. Assume for definiteness that for the first reduction $i_1 = 1$, and that the nonzero elements in the first row of Δ are Δ_{11} and Δ_{12}, so that

$$\Delta = \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 \\ \Delta_{21} & \Delta_{22} & & & \\ \vdots & \vdots & & \Delta'' & \\ \Delta_{m1} & \Delta_{m2} & & & \end{pmatrix},$$

$$\Delta' = \begin{pmatrix} \Delta_{21} & \Delta_{22} & & \\ \vdots & \vdots & & \\ & & \Delta'' & \\ \Delta_{m1} & \Delta_{m2} & & \end{pmatrix}.$$

Case 2.1. Each of the first two columns of Δ' has at least two nonzero elements. These four nonzero elements belong to four different rows, for otherwise one of the rows of Δ' would majorize the first row of Δ, and branching would not apply. Then Δ'' contains at least four rows with one nonzero element in each. Each of these rows is either eliminated or included in τ'' prior to the first reduction, so that this reduction is applied to a matrix with at most $m-5$ rows. On the other hand, Δ' has $m-1$ rows. By the induction hypothesis, $L(\tau') \leqq \lambda^{m-1}$, $L(\tau'') \leqq \lambda^{m-5}$. Hence

$$L(\tau) = L(\tau') + L(\tau'') \leqq \lambda^{m-1} + \lambda^{m-5} = \lambda^{m-5}(\lambda^4 + 1) = \lambda^m.$$

Case 2.2. One of the first two columns of Δ' has at least two nonzero elements, and the other has one nonzero element. Arguing as in Case 2.1, we see that Δ'' has at least three rows with one nonzero element in each, so that the first reduction in τ'' is applied to a matrix with at most $m-4$ rows. On the other hand, Δ' has at least one necessary row, so that the first reduction in τ' is applied to a matrix with at most $m-2$ rows. By the induction hypothesis, $L(\tau') \leqq \lambda^{m-2}$, $L(\tau'') \leqq \lambda^{m-4}$. Hence $L(\tau) \leqq \lambda^{m-2} + \lambda^{m-4} = \lambda^{m-4}(\lambda^2 + 1) < \lambda^m$ (calculation shows that $\lambda^2 + 1 < \lambda^4$).

Case 2.3. Each of the first two columns of Δ' has one nonzero element. Then Δ'' has at least two rows with one nonzero element in each, and Δ' has at least two necessary rows, so that both in Δ' and in Δ'' the first reduction is applied to a matrix with at most $m-3$ rows. By the induction hypothesis, $L(\tau')$, $L(\tau'') \leqq \lambda^{m-3}$. Hence

$$L(\tau) \leqq 2\lambda^{m-3} < \lambda^m$$

(calculation shows that $\lambda^3 > 2$).

Lemma 3 is proved. □

Since $\lambda \approx 1.3247$, Lemma 3 shows that the computation time of the algorithm in question applied to finding minimum edge covers is estimated from above by 1.325^m, where m is the number of edges. This estimate will be used in Theorem 4 below. (As for the minimum edge cover problem itself, it can be solved in polynomial time by matching methods; see the Introduction).

An estimate from below can be obtained as in the proof of Theorem 2 with $\varphi_{3,2}$ instead of $\varphi_{5,4}$. In other words, for $m = 3i$, there exists an $m \times m$ matrix Δ with at most two nonzero elements in every row, such that for every Δ-tree τ

$$L(\tau) > (2^{1/3})^m > 1.259^m.$$

This matrix Δ consists of i blocks equal to $\varphi_{3,2}$ each. The corresponding graph consists of i triangles not connected with each other (incidentally, this graph is known to have the maximum possible number of maximal independent sets [14]).

Now we shall estimate the size of Δ-trees under a natural additional restriction on their structure.

A Δ-tree τ is *regular* if branching is always applied in τ to a row with the maximum number of nonzero elements. For instance, the proof of Lemma 1 shows that every $\varphi_{m,k}$-tree τ is regular, since branching is applied in τ only to matrices with the same number of nonzero elements in every line. The tree constructed in the proof of Theorem 3 is not regular. A regular branching minimizes the number of columns in the second of the two matrices it leads to, and thus minimizes the total size of the matrices.

THEOREM 4. *Let* $\mu > 3/4^{1/3} \approx 1.8899$. *For every sufficiently large* m, *every* $m \times m$ *matrix* Δ *and every regular* Δ*-tree* τ,

$$L(\tau) < \mu^m. \tag{2.13}$$

Proof. A vertex x of τ is *critical* if the submatrix Δ_x of Δ assigned to x has at most two nonzero elements in every row, and no matrix between Δ_x and the root of τ has this property. Every branch of τ contains exactly one critical vertex. To estimate $L(\tau)$, we shall estimate the number of critical vertices and, using Lemma 3, the number of leaves under each critical vertex.

The position of a critical vertex x in τ can be described as follows. Let $\nu_1, \cdots, \nu_k$ be the branchings on the way from the root of τ to x. Let ε_i $(i = 1, \cdots, k)$ be zero if the way from ν_i to x passes through the left of the two descendants of ν_i, and one, otherwise. Define

$$\mathscr{E}(x) = (\varepsilon_1, \cdots, \varepsilon_k).$$

Since no two critical vertices belong to the same branch, the Boolean vector $\mathscr{E}(x)$ completely defines x.

The size of Δ_x can be estimated in terms of $\mathscr{E}(x) = (\varepsilon_1, \cdots, \varepsilon_k)$ as follows. Since every branching involves deleting a row, Δ_x has at most $m - k$ rows. It follows that $k \leqq m$. Since τ is regular, and each ν_i occurs above a critical vertex, the row used for branching in ν_i has at least three nonzero elements. Then each nonzero component of $\mathscr{E}(x)$ corresponds to deleting at least three columns. It follows that the number l of nonzero components of $\mathscr{E}(x)$ does not exceed $m/3$. The number of vectors $\mathscr{E}(x)$ of length k with l nonzero components does not exceed $\binom{k}{l}$; hence the number of vectors $\mathscr{E}(x)$ of length k does not exceed $\sum_{l \leqq m/3} \binom{k}{l}$. Lemma 3 implies that the number of leaves of τ descending from x does not exceed λ^{m-k}. Hence

$$L(\tau) \leqq \sum_{\substack{k,l \\ 0 \leqq l \leqq k \leqq m \\ l \leqq m/3}} \binom{k}{l} \lambda^{m-k} \leqq (m+1)\left(\frac{m}{3}+1\right) \max_{\substack{0 \leqq l \leqq k \leqq m \\ l \leqq m/3}} F(k, l)$$

where $F(k, l) = \binom{k}{l}\lambda^{m-k}$. To estimate $F(k, l)$, consider two cases.

Case 1. $k \leqq 2m/3$. Then

$$F(k, l) \leqq 2^k \cdot \lambda^{m-k} = \lambda^m \cdot (2\lambda^{-1})^k \leqq \lambda^m (2\lambda^{-1})^{2m/3}$$

$$= ((4\lambda)^{1/3})^m < (6^{1/3})^m < \left(\frac{3}{4^{1/3}}\right)^m.$$

Case 2. $k > 2m/3$. Then

$$F(k, l) \leqq F(k, m/3),$$

and $F(k, m/3)$ is an increasing function of k:

$$\frac{F(k, m/3)}{F(k-1, m/3)} = \frac{\binom{k}{m/3}\lambda^{m-k}}{\binom{k-1}{m/3}\lambda^{m-k+1}} = \frac{k}{(k-m/3)\lambda} \geqq \frac{m}{(m-m/3)\lambda} = \frac{3}{2\lambda} > 1.$$

Hence, by Stirling's approximation for factorials,

$$F(k, l) \leqq F(m, m/3) = \binom{m}{m/3} = O\left(\left(\frac{3}{4^{1/3}}\right)^m\right).$$

Theorem 4 is proved. □

3. The most probable behavior of the set covering algorithms. Introduce Ω_{mn}, the set of all families $\Delta = (D_i)_{i \in I}$ with $I = \{1, \cdots, m\}$, $D_i \subset \{1, \cdots, n\}$. Given $\Delta \in \Omega_{mn}$, associate with it the $m \times n$ Boolean matrix $\omega = (\omega_{ij})$, where $\omega_{ij} = 1$ if and only if $j \in D_i$. Assume that the family Δ, i.e., the matrix ω, is taken in Ω_{mn} at random. In other words, we introduce the uniform distribution $P(\cdot)$,

$$P(\omega) = |\Omega_{mn}|^{-1} = 2^{-mn} \quad \forall \omega \in \Omega_{mn},$$

and all the functions of ω become random variables. (In particular, the entries ω_{ij} are independent and $P(\omega_{ij} = 1) = P(\omega_{ij} = 0) = \frac{1}{2}$, $\forall i, j$.)

Note. Remember that n has previously stood for $|\cup_i D_i|$, and now $|\cup_i D_i| < n$ with positive probability. Still, we decided not to introduce a new symbol because, under the condition of Theorem 5 below, $P(|\cup_i D_i| = n) \to 1$ as $m, n \to \infty$.

Our goal is to study the random number of recursive calls of the set covering procedure.

Notice first that in the preceding section we obtained estimates for the worst-case size of Δ-trees (more appropriately, ω-trees) under a single restriction on their structure (except regular ω-trees), namely that branching is not used if not necessary. Thus the choice of reductions was not defined completely.

In order to consider the size $|\tau|$ of an ω-tree τ as a random variable, we shall study here completely defined (deterministic) algorithms. For these algorithms, the choice of reduction at each vertex of the ω-tree is uniquely determined by the submatrix of ω assigned to this vertex. (More generally, the choice may be determined by the submatrices assigned to vertices which form the path leading to the vertex from the root.)

Consider large m and n. The theorem below clearly demonstrates that, for each of these algorithms, the most probable computation time is a subexponential function of $\max(m, n)$. Henceforth it is considerably smaller than that in the worst case (see Theorems 2, 3).

THEOREM 5. *Let $m, n \to \infty$ in such a way that*

$$\lim_{m,n\to\infty} \frac{\ln n}{\ln m} = \gamma \in (0, +\infty).$$

Then, for each deterministic algorithm,

$$\lim_{m,n\to\infty} P(m^{c_1 \ln m} \leqq |\tau| \leqq m^{c_2 \ln m}) = 1,$$

where

$$c_1 = \begin{cases} 0.36 \cdot \gamma^2 & \text{if } \gamma \leqq 2, \\ 1.44 \cdot (\gamma - 1) & \text{if } \gamma > 2, \end{cases}$$

$$c_2 = 0.37 \cdot (1 + \sqrt{1+4\gamma})^2.$$

Remark. Notice that the constants c_1, c_2 are the same for all the algorithms in question. Particular properties of an algorithm can be used to sharpen the estimates. For example, if it is agreed to always choose for branching the first available row, then c_1 in Theorem 5 can be replaced by 1.44γ.

This theorem and some experience in dealing with subexponentially growing random variables [11] lead us to:

CONJECTURE. *For a given deterministic algorithm,*

$$\frac{\ln |\tau|}{\ln^2 m} \to c \quad \textit{in probability},$$

where c is a constant dependent upon γ and the algorithm.

Also, it is worth mentioning that a *lower* probabilistic estimate similar to one in Theorem 5 was conjectured in [2] with regard to an algorithm estimating the stability number of a graph.

Proof of Theorem 5. *Part* 1. *Lower estimate.* Introduce the maximal *binary* subtree $\tilde{\tau}$ of the ω-tree τ which grows from the same root as τ. It consists of only the root if and only if the first step of the algorithm is not branching. For all its vertices except endpoints. the algorithm calls for branching. Clearly, $|\tau| \geqq |\tilde{\tau}|$.

Fix $\alpha, \beta \in (0, 1)$ and introduce $S_{\alpha\beta} = S_{\alpha\beta}(\omega)$ the number of the endpoints of $\tilde{\tau}$ which are reached from the root by making no more than $[m^\beta]$ moves in such a way that the number of moves to the right does not exceed $a = [(1-\alpha)\log_2 n]$. (Each of these endpoints is associated with a submatrix of ω obtained by elimination of no more than $[m^\beta]$ rows, and no more than a of these eliminations are accompanied by elimination of the correspondent columns.)

Prove that

$$\lim_{m,n\to\infty} P(\omega : S_{\alpha\beta}(\omega) \neq 0) = 0,$$

provided $\beta < \alpha \cdot \gamma$.

To this end, let us estimate $E(S_{\alpha\beta})$. Notice first that $\tilde{\tau}$ is a binary subtree of the complete binary tree with $2^m - 1$ vertices. Let x be a vertex of the latter and $l = l(x)$ be the length of the path leading to it from the root. As in the proof of Theorem 4,

introduce the Boolean vector $\mathscr{E}(x) = (\varepsilon_1, \cdots, \varepsilon_l)$, so that $\varepsilon_i = 1$ if and only if ith edge of the path is rightward directed. Denote $|\mathscr{E}(x)| = \varepsilon_1 + \cdots + \varepsilon_{l(x)}$. Then

$$E(S_{\alpha\beta}) = \sum_{\{x | l(x) \leqq [m^\beta], |\mathscr{E}(x)| \leqq a\}} P(x),$$

where $P(x)$ is the probability that the vertex x is an endpoint of $\tilde{\tau}$. Now,

$$P(x) = \sum_{\bar{i}} P(x, \bar{i}),$$

where summation is taken over all *ordered* tuples $\bar{i} = (i_1, \cdots, i_l)$, $1 \leqq i_1, \cdots, i_l \leqq m$, $l = l(x)$, and $P(x, \bar{i})$ is the probability that x is an endpoint of $\tilde{\tau}$ and that the branchings leading to x use consecutively rows with numbers $i_1, i_2, \cdots, i_l$. Denote

$$I^{(1)}(x, \bar{i}) = \{i : i = i_k, \varepsilon_k = 1 \text{ for some } k = 1, \cdots, l\},$$

$$J(x, \bar{i}, \omega) = \bigcup_{\substack{i \neq i_k \\ 1 \leqq k \leqq l}} D_i \Big\backslash \bigcup_{i \in I^{(1)}(x, \bar{i})} D_i.$$

Call a submatrix of ω *degenerate* if it has either a necessary row or a majorizing row (column). Introduce

$$P_1(x, \bar{i}) = P(\omega : J(x, \bar{i}, \omega) = \varnothing),$$

$$P_2(x, \bar{i}) = P(\omega : J(x, \bar{i}, \omega) \neq \varnothing,$$

but $(\omega_{\mu\nu})$, $(\mu \neq i_1, \cdots, i_l, \nu \in J(x, \bar{i}, \omega))$, is degenerate).

By the definition of $P(x, i)$, we obtain

$$P(x, \bar{i}) \leqq P_1(x, \bar{i}) + P_2(x, \bar{i}). \tag{3.1}$$

Further, as $|I^{(1)}(x, \bar{i})| = |\mathscr{E}(x)| \leqq a = [(1-\alpha)\log_2 n]$, $l(x) \leqq [m^\beta]$,

$$\begin{aligned} P_1(x, \bar{i}) &= \left[P\left(\omega : \max_{i \in I^{(1)}(x, \bar{i})} \omega_{i1} = 1 \text{ or } \max_{i \neq i_1, \cdots, i_l} \omega_{i1} = 0\right)\right]^n \\ &= [1 - 2^{-|\mathscr{E}(x)|} + 2^{-(m - l(x) + |\mathscr{E}(x)|)}]^n \leqq c[1 - 2^{-|\mathscr{E}(x)|}]^n \\ &\leqq c \exp(-n2^{-a}) \leqq c \exp(-cn^\alpha). \end{aligned} \tag{3.2}$$

(Here and below, we use a letter c to denote various positive constants whose actual values do not matter.)

To estimate $P_2(x, \bar{i})$, write

$$P_2(x, \bar{i}) \leqq \sum_{k=1}^{n} \binom{n}{k} (2^{-|\mathscr{E}(x)|})^k \cdot (1 - 2^{-|\mathscr{E}(x)|})^{n-k} \cdot \pi(m - |l(x)|, k), \tag{3.3}$$

where $\pi(s, t)$ stands for the probability that an $s \times t$ submatrix of ω is degenerate. (Really, $\binom{n}{k}(2^{-|\mathscr{E}(x)|})^k (1 - 2^{-|\mathscr{E}(x)|})^{n-k}$ is the probability that elimination of rows with numbers $i_1, \cdots, i_l$, coupled with elimination of columns covered by rows with numbers from the subset $I^{(1)}(x, \bar{i})$, leads to an $(m - |l(x)|) \times k$ submatrix of ω.) For an ordered pair of rows (columns), the first majorizes the second with probability $(\frac{3}{4})^t((\frac{3}{4})^s)$, and a fixed column has a single nonzero element at a fixed position with probability $(\frac{1}{2})^s$. Hence,

$$\begin{aligned} \pi(s, t) &\leqq s(s-1) \cdot (\tfrac{3}{4})^t + t(t-1) \cdot (\tfrac{3}{4})^s + st(\tfrac{1}{2})^s \\ &\leqq c(m^2 + n^2)(\rho^t + \rho^s), \qquad \rho = \tfrac{3}{4}. \end{aligned} \tag{3.4}$$

By (3.3), (3.4),

$$
\begin{aligned}
P_2(x, \bar{i}) &\leqq c(m^2+n^2) \sum_{k=0}^{n} \binom{n}{k} (2^{-|\mathscr{E}(x)|})^k \cdot (1-2^{-|\mathscr{E}(x)|})^{n-k} \cdot (\rho^{m-|l(x)|}+\rho^k) \\
&= c(m^2+n^2)[\rho^{m-|l(x)|}+(1+(\rho-1)2^{-|\mathscr{E}(x)|})^n] \\
&\leqq c(m^2+n^2)[\rho^{m-[m^\beta]}+\exp(n(\rho-1)2^{-a})] \\
&\leqq c[\rho^{m/2}+\exp(-cn^\alpha)].
\end{aligned} \tag{3.5}
$$

Combining (3.1), (3.2), (3.5), we get

$$P(x, \bar{i}) \leqq c[\rho^{m/2}+\exp(-cn^\alpha)].$$

Consequently

$$
\begin{aligned}
P(x) = \sum_{\bar{i}} P(x, \bar{i}) &\leqq c(m)_l[\rho^{m/2}+\exp(-cn^\alpha)] \\
&\leqq cm^{m^\beta}[\rho^{m/2}+\exp(-cn^\alpha)] = g(m, n).
\end{aligned}
$$

Hence

$$
\begin{aligned}
E(S_{\alpha\beta}) &= \sum_{\{x|l(x)\leqq[m^\beta],|\mathscr{E}(x)|\leqq a\}} P(x) \\
&\leqq g(m, n) \cdot \sum_{\{x|l(x)\leqq[m^\beta],|\mathscr{E}(x)|\leqq a\}} 1 \\
&= g(m, n) \cdot \sum_{j=0}^{[m^\beta]} \sum_{k=0}^{\min(a,j)} \binom{j}{k} = g(m, n) \cdot \sum_{k=0}^{a} \sum_{j=k}^{[m^\beta]} \binom{j}{k} \\
&= g(m, n) \cdot \sum_{k=0}^{a} \binom{[m^\beta]+1}{k+1} \leqq g(m, n)(a+1)\binom{[m^\beta]+1}{a+1} \\
&\leqq g(m, n) \cdot ([m^\beta]+1)^{a+1} = cm^{m^\beta}[\rho^{m/2}+\exp(-cn^\alpha)]([m^\beta]+1)^{a+1},
\end{aligned}
$$

or

$$
\begin{aligned}
E(S_{\alpha\beta}) \leqq c\Big[&\exp\left(\frac{m}{2}\ln\rho + m^\beta \cdot \ln m + c \ln n \cdot \ln m\right) \\
&+ \exp(-cn^\alpha + m^\beta \ln m + c \ln n \ln m)\Big].
\end{aligned}
$$

Recalling that $\rho < 1$, $\beta < \min(1, \alpha\gamma)$, $(\gamma = \lim \ln n/\ln m)$, we conclude that

$$\lim_{m,n\to\infty} E(S_{\alpha\beta}) = 0,$$

and therefore

$$P(\Omega_{mn}(\alpha, \beta)) \to 1, \qquad \Omega_{mn}(\alpha, \beta) = \{\omega \in \Omega_{mn} : S_{\alpha\beta}(\omega) = 0\}.$$

But, on $\Omega_{mn}(\alpha, \beta)$, the binary subtree $\tilde{\tau}$ contains all the vertices of the complete binary tree which are connected with the root by paths of the length $[m^\beta]$ having $a = [(1-\alpha)\log_2 n]$ moves to the right. Thus, for these ω's,

$$
\begin{aligned}
|\tau(\omega)| \geqq |\tilde{\tau}(\omega)| &\geqq \binom{[m^\beta]}{a} = \exp(a \cdot \ln[m^\beta](1+o(1))) \\
&= \exp((1-\alpha)\beta \cdot \ln m \cdot \log_2 n(1+o(1))) \\
&= m^{c(\alpha,\beta,\gamma)\ln m(1+o(1))},
\end{aligned}
$$

where

$$c(\alpha, \beta, \gamma) = (1-\alpha)\beta\gamma/\ln 2.$$

Elementary arguments show that

$$\sup\{c(\alpha, \beta, \gamma): \alpha, \beta \in (0, 1), \beta < \alpha\gamma\} = \begin{cases} \gamma^2/4 \ln 2 & \text{if } \gamma \leqq 2, \\ (\gamma-1)/\ln 2 & \text{if } \gamma > 2. \end{cases}$$

It leads directly to the lower estimate in the theorem, as $(4 \ln 2)^{-1} = 0.3606\cdots > 0.36$, $(\ln 2)^{-1} > 1.44$.

Part 2. Upper estimate. Let x be a leaf of the ω-tree τ. Of the branchings on the way from the root of τ to x, consider only the ones in which rows are eliminated together with the columns covered by them. (These branchings correspond to rightward moves of the path.) Introduce $R(x)$, the set of these rows according to the order of elimination. It is important that

$$R(x') \neq R(x'') \quad \text{if } x' \neq x''. \tag{3.6}$$

Observe also that if $R(x) = (i_1, \cdots, i_k)$ then

$$|\{j: \omega_{ij} = 0, i = i_1, \cdots, i_{s-1}, \text{ and } \omega_{i_s j} = 1\}| \geqq 2, \qquad s = 1, \cdots, k, \tag{3.7}$$

i.e., the row i_s covers at least two columns which are not covered by the *preceding* rows $i_1, \cdots, i_{s-1}$ $(s = 1, \cdots, k)$. (Otherwise, for some s, the row i_s is either a necessary row of the submatrix assigned to the correspondent vertex of τ, or majorized by another row of this submatrix: in any case, branching is not warranted.)

Let $\mathscr{L}(\omega)$ be the set of all the leaves x of $\tau(\omega)$, $L(\omega) = |\mathscr{L}(\omega)|$. Fix $\alpha > 0$ and introduce $L_\alpha(\omega)$, the number of leaves x for which

$$|R(x)| \geqq a = [(1+\alpha)\log_2 n].$$

We have (see (3.6), (3.7)):

$$\begin{aligned} E(L_\alpha) &= E\Big(\sum_{x \in \mathscr{L}(\omega)} 1_{\{\omega: |R(x)| \geqq a\}}\Big) \\ &= E\Big(\sum_{R: |R| \geqq a} 1_{\{\omega: R = R(x) \text{ for some } x \in \mathscr{L}(\omega)\}}\Big) \\ &\leqq \sum_{R: |R| \geqq a} P(n, R). \end{aligned} \tag{3.8}$$

Here summation is taken over $R = (i_1, \cdots, i_k)$, $k \geqq a$, ordered subsets of $I = (1, \cdots, m)$, and

$$P(n, R) = P\{\omega: \text{the condition (3.7) holds true}\}.$$

Clearly, $P(n, R) = P(n, (1, \cdots, |R|)) = f(n, |R|)$ and (3.8) becomes

$$E(L_\alpha) \leqq \sum_{r=a}^{m} (m)_r f(n, r) \leqq \sum_{r=a}^{m} m^r f(n, r). \tag{3.9}$$

To proceed further, we need

LEMMA 4. *Uniformly over* $r \geqq a = [(1+\alpha)\log_2 n]$,

$$f(n, r) \leqq \exp[-\alpha^2 \log_2 n \cdot \ln n(1+o(1))](c \cdot n^{-2\alpha})^{r-a}. \tag{3.10}$$

Proof. Define $f(n, 0) = 1$ $\forall n$. Then, from the definition of $f(\cdot, \cdot)$, it follows easily that

$$f(n, s) = \sum_{j=2}^{n} 2^{-n} \cdot \binom{n}{j} \cdot f(n-j, s-1), \qquad n \geqq 2, \quad s \geqq 1. \tag{3.11}$$

Introduce $F(x, s)$, the *exponential* generating function of $f(\cdot, s)$:

$$F(x, s) = \sum_{n \geq 0} \frac{f(n, s)x^n}{n!}.$$

In particular, $F(x, 0) = \exp(x)$. In view of (3.11), for $s \geq 1$, we obtain

$$\begin{aligned} F(x, s) = \sum_{n \geq 0} \frac{f(n, s)x^n}{n!} &= \sum_{n \geq 2} \frac{f(n, s)x^n}{n!} \\ &= \sum_{n \geq 2} \frac{x^n}{n!} \cdot \left(\sum_{j=2}^{n} 2^{-n} \binom{n}{j} f(n-j, s-1) \right) \\ &= \sum_{j \geq 2} \frac{(x/2)^j}{j!} \cdot \left(\sum_{n \geq j} (x/2)^{n-j} \cdot f(n-j, s-1) \Big/ (n-j)! \right) \\ &= [\exp(x/2) - 1 - (x/2)] \cdot F(x/2, s-1). \end{aligned}$$

Hence ($F(x, 0) = \exp(x)$),

$$F(x, s) = \exp(x/2^s) \cdot \prod_{j=1}^{s} [\exp(x/2^j) - 1 - (x/2^j)].$$

The last relation implies (see the definition of $F(x, s)$) that

$$f(n, s) \leqq n! x^{-n} \cdot \exp(x/2^s) \cdot \prod_{j=1}^{s} [\exp(x/2^j) - 1 - (x/2^j)], \tag{3.12}$$

where $x > 0$, and is otherwise arbitrary. Choose $x = n$ and consider $s \geq a = [(1+\alpha) \log_2 n]$. Notice first that, by Stirling's formula,

$$n! n^{-n} \cdot \exp(n/2^s) = \exp(-n + O(\ln n) + O(n^{-\alpha})) \leqq \exp(-n + c \ln n). \tag{3.13}$$

Further,

$$\begin{aligned} \sum_{j=1}^{s} \ln [\exp(n/2^j) - 1 - (n/2^j)] &= \sum_{1 \leqq j \leqq [\log_2 n]} + \sum_{[\log_2 n] < j \leqq a} + \sum_{a < j \leqq s} \\ &= \sum_1 + \sum_2 + \sum_3. \end{aligned} \tag{3.14}$$

Here

$$\sum_1 \leqq \sum_{1 \leqq j \leqq [\log_2 n]} \ln [\exp(n/2^j)] \leqq n \sum_{j \geq 1} 2^{-j} = n. \tag{3.15}$$

Also, ($\exp(x) - 1 - x \leqq cx^2$, for $x \leqq 1$),

$$\begin{aligned} \sum_2 &\leqq \sum_{[\log_2 n] < j \leqq a} \ln [c(n/2^j)^2] = \sum_{[\log_2 n] < j \leqq [(1+\alpha) \log_2 n]} 2(\ln n - j \ln 2) + O(\ln n) \\ &\leqq -\alpha^2 \log_2 n \cdot \ln n + c \ln n, \end{aligned} \tag{3.16}$$

and

$$\sum_3 \leqq \sum_{a < j \leqq s} \ln [c(n/2^j)^2] \leqq (s-a) \ln [c(n/2^a)^2] \leqq (s-a) \ln (cn^{-2\alpha}). \tag{3.17}$$

Putting together (3.12)–(3.17) completes the proof of Lemma 4. □

By (3.9) and this lemma,

(3.18)
$$E(L_\alpha) \leqq \exp\left[-(\alpha^2 \log_2 n \cdot \ln n - (1+\alpha)\log_2 n \cdot \ln m) + o(\ln^2 n)\right] \sum_{r=a}^{m} (cmn^{-2\alpha})^{r-a}.$$

Now,

$$\lim_{m,n\to\infty} (\alpha^2 \log_2 n \cdot \ln n - (1+\alpha)\log_2 n \cdot \ln m)/(\log_2 n \cdot \ln m) = \alpha^2\gamma - (1+\alpha).$$

Choose $\alpha > \alpha(\gamma) = (1+\sqrt{1+4\gamma})/2\gamma$, which is the positive root of an equation

$$z^2 \cdot \gamma - z - 1 = 0.$$

For this choice of α,

$$\alpha^2 \cdot \gamma - (1+\alpha) > 0,$$

and

(3.19)
$$mn^{-2\alpha} = \exp(-2\alpha \ln n + \ln m) = \exp\left[-\ln m(2\alpha\gamma - 1) + o(\ln m)\right],$$

where

(3.20)
$$2\alpha\gamma - 1 > \sqrt{1+4\gamma} > 0.$$

Invoking (3.18)–(3.20), we obtain: if $\alpha > (1+\sqrt{1+4\gamma})/2\gamma$, then

$$E(L_\alpha) \leqq \exp(-c(\ln m)^2)(1 - cmn^{-2\alpha})^{-1} \to 0, \qquad m, n \to \infty.$$

Hence,

$$P(\omega: |R(x)| < a \text{ for all leaves } x \text{ of } \tau(\omega)) = P(\omega: L_\alpha(\omega) = 0) \to 1,$$

as $m, n \to \infty$. This implies directly that

$$P\left(\omega: L(\omega) < \sum_{r<a} (m)_r\right) \to 1, \qquad m, n \to \infty.$$

But

$$\sum_{r<a} (m)_r < am^a = m^{c_2(\alpha,\gamma)\ln m(1+o(1))},$$

$c_2(\alpha, \gamma) = (1+\alpha)\gamma/\ln 2$, so

$$P(\omega: L(\omega) < m^{c_2(\alpha,\gamma)\ln m(1+o(1))}) \to 1.$$

As $|\tau(\omega)| \leqq (m+n+1)L(\omega)$ and

$$\inf_{\alpha>\alpha(\gamma)} c_2(\alpha, \gamma) = c_2(\alpha(\gamma), \gamma) = (1+\sqrt{1+4\gamma})^2/4 \cdot \ln 2 < 0.37(1+\sqrt{1+4\gamma})^2,$$

we arrive finally at the upper estimate in the theorem. □

Acknowledgment. The authors thank the editor and referees for many valuable remarks and suggestions which helped us to improve the presentation of the paper.

REFERENCES

[1] D. AVIS, *A note on some computationally difficult set covering problems*, Math. Programming, 18 (1980), pp. 138–145.

[2] V. CHVÁTAL, *Determining the stability number of a graph*, this Journal, 6 (1977), pp. 643–662.

[3] ———, *Hard knapsack problems*, J. Oper. Res., 28 (1980), pp. 1402–1411.
[4] J. EDMONDS, *Paths, trees, and flowers*, Canad. J. Math., 17 (1965), pp. 449–467.
[5] S. EVEN, *Algorithmic Combinatorics*, Macmillan, New York, 1973.
[6] R. FULKERSON, G. NEMHAUSER AND L. TROTTER, *Two computationally difficult set covering problems that arise in computing the* 1*-width of incidence matrices of Steiner triple systems*, Math. Programming Stud., 2 (1974), pp. 72–81.
[7] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability, A Guide to the Theory of* NP*-Completeness*, Freeman, San Francisco, 1979.
[8] 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.
[9] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.
[10] V. LIFSCHITZ, *The efficiency of an algorithm of an integer programming: a probabilistic analysis*, Proc. Amer. Math. Soc., 79 (1980), pp. 72–76.
[11] V. LIFSCHITZ AND B. PITTEL, *The number of increasing subsequences of the random permutation*, J. Combin. Theory Ser. A, 3 (1981), pp. 1–20.
[12] V. LIFSCHITZ AND L. PESOTCHINSKY, *The analysis of a partition algorithm*, J. Assoc. Comp. Mach., to appear.
[13] C. MCDIARMID, *Determining the chromatic number of a graph*, this Journal, 8 (1979), pp. 1–14.
[14] J. W. MOON AND L. MOSER, *On cliques in graphs*, Israel J. Math., 3 (1965), pp. 23–28.
[15] R. Z. NORMAN AND M. O. RABIN, *An algorithm for a minimum cover of a graph*, Proc. Amer. Math. Soc., 10 (1959), pp. 315–319.
[16] J. M. PLOTKIN AND J. W. ROSENTHAL, *On the expected number of branches in analytic tableaux analysis in propositional calculus*, Notices Amer. Math. Soc., 25 (1978), pp. A-437.
[17] R. E. TARJAN AND A. E. TROJANOVSKI, *Finding a maximum independent set*, this Journal, 6 (1977), pp. 537–546.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0010 $01.25/0

TOWARDS A GENUINELY POLYNOMIAL ALGORITHM FOR LINEAR PROGRAMMING*

NIMROD MEGIDDO†

Abstract. A linear programming algorithm is called genuinely polynomial if it requires no more than $p(m, n)$ arithmetic operations to solve problems of order $m \times n$, where p is a polynomial. It is not known whether such an algorithm exists. We present a genuinely polynomial algorithm for the simpler problem of solving linear inequalities with at most two variables per inequality. The number of operations required is $O(mn^3 \log m)$. The technique used was developed in a previous paper where a novel binary search idea was introduced.

Key words. linear programming, genuinely polynomial-time, convex minimization

1. Introduction. A major result in computational complexity theory was reported by Khachiyan [6] in 1979, namely, that the feasibility of linear inequalities can be decided in polynomial time. However, many researchers interested in linear programming have not been completely satisfied with Khachiyan's result for the following reasons. First, the fact that Khachiyan's algorithm is polynomial depends on the numbers being given in binary encoding. It is not hard (see [9]) to establish encoding schemes with respect to which Khachiyan's algorithm requires an *exponential* number of operations, although the operations themselves require polynomial time. The number of operations tends to infinity with the magnitude of the coefficients and thus for any given class of problems with fixed numbers of variables and inequalities, the number of arithmetic operations required by Khachiyan's algorithm is unbounded. Secondly, Khachiyan's algorithm has not yet been proven practical, while the simplex algorithm is usually efficient [4].

By solving a set of linear inequalities we mean producing a feasible solution or else recognizing that the set is infeasible. An interesting open question is the following; Do there exist an algorithm and a polynomial $p(m, n)$ such that every set of m linear inequalities with n variables is solved by the algorithm in less than $p(m, n)$ arithmetic operations? We shall call such an algorithm *genuinely polynomial.* It is not even known whether the transportation problem has a genuinely polynomial algorithm. The scaling method of Edmonds and Karp [5] has a polynomial time-bound but, as in Khachiyan's algorithm, the number of arithmetic operations depends on the magnitude of the coefficients.

In this paper we shall be discussing a special type of system of linear inequalities, namely, sets of m inequalities with n variables but no more than two variables per inequality. Previous results were obtained by Chan [3] and Pratt [11]. They solved the special case of inequalities of the form $x - y \leqq c$ (i.e., the dual of a shortest-path problem) in $O(n^3)$ operations. Shostak [12] developed a nice theory, on which we base our results in this paper, but his algorithm is exponential in the worst-case. Nelson [10] gave an $O(mn^{\lceil \log_2 n \rceil + 4} \log n)$ algorithm. Polynomial-time algorithms for this problem were given by Aspvall and Shiloach [2] and by Aspvall [1]. The former requires $O(mn^3 I)$ arithmetic operations, where I is the size of the binary encoding of the input, while the latter requires $O(mn^2 I)$ operations.

We shall present an algorithm which requires $O(mn^3 \log m)$ operations, i.e., a genuinely polynomial algorithm for solving systems of linear inequalities of order

* Received by the editors August 17, 1981, and in revised form July 15, 1982. This research was partially supported by the National Science Foundation under grants ECS7909724 and ECS8121741. The work was done while the author was visiting at Northwestern University.

† Statistics Department, Tel Aviv University, Tel Aviv, Israel.

$m \times n$ with at most two variables per inequality. Our algorithm is based on that of Aspvall and Shiloach [2] and on Shostak's [12] result. A similar construction can be based on Aspvall's [1] algorithm but no better complexity is obtained. Thus, although this paper is intended to be self-contained, the reader may find it helpful to refer to [2] and [12] for further clarifications.

2. Preliminaries. Given is a set S of m linear inequalities involving n variables but no more than two variables per inequality. Suppose $S = S_1 \cup S_2$, where S_i is the set of inequalities involving exactly i distinct variables ($i = 1, 2$). Without loss of generality, assume that S_1 is given in the form $\text{lo}(y) \leqq y \leqq \text{up}(y)$, where $\text{lo}(y)$ and $\text{up}(y)$ are the lower and upper bounds, respectively, on the variable y; these bounds may be infinite. It will be convenient to maintain for every variable y a list of all the inequalities in which y participates.

Throughout the computation there will be derived more and more restrictive lower and upper bounds, $\underline{y}$ and $\bar{y}$ respectively, for each variable y. The basic step of updating such bounds makes use of a single inequality from S_2. Given the current bounds $\underline{y}$, $\bar{y}$ on y and any inequality $ay + bz \leqq c$ in which y participates ($a, b \neq 0$), the bounds on z may be updated in an obvious way. We define the routine FORWARD $(y, ay + bz \leqq c)$ to be the updating procedure which operates according to the following case classification:

$$\begin{aligned}
&\text{case (i): } a, b > 0, && \bar{z} \leftarrow \min[\bar{z}, (c - a\underline{y})/b],\\
&\text{case (ii): } a > 0, b < 0, && \underline{z} \leftarrow \max[\underline{z}, (c - a\underline{y})/b],\\
&\text{case (iii): } a < 0, b > 0, && \bar{z} \leftarrow \min[\bar{z}, (c - a\bar{y})/b],\\
&\text{case (iv): } a, b < 0, && \underline{z} \leftarrow \max[\underline{z}, (c - a\bar{y})/b].
\end{aligned}$$

The routine FORWARD detects infeasibility when $\bar{z} < \underline{z}$.

The routine FORWARD may repeatedly be applied along "chains" of inequalities. Specifically, a sequence of inequalities $a_i y_i + b_i y_{i+1} \leqq c_i$, $i = 1, \cdots, k$, may be used for updating the bounds on y_{k+1} by starting from the bounds on y_1 and updating $\underline{y_{i+1}}$, $\overline{y_{i+1}}$ according to the updated $\underline{y_i}$, $\overline{y_i}$ $(i = 1, \cdots, k)$. Consider the case where the initial bounds are $\underline{y} = \text{lo}(y)$, $\bar{y} = \text{up}(y)$ for all $y \neq y_1$ and $\underline{y_1} = \overline{y_1} = g$, where g is any real number. Obviously, the bounds that will be derived with respect to $y_2, \cdots, y_{k+1}$ will be linear functions of g (not excluding the possibility of infinite bounds).

A special case of chains is that of a "loop", i.e., when y_{k+1} and y_1 are the same variable, which we now denote by x. Consider, for example, a case where applying the routine FORWARD around a loop starting and ending at x yields $\underline{x} = \alpha g + \beta$. A necessary condition for feasibility is that $x \geqq \alpha x + \beta$. This is an inequality which is "hidden" in our loop and obviously has the following consequences:

(i) If $\alpha = 1$ and $\beta > 0$ then S is infeasible; in this case we say that the loop is infeasible.

(ii) If $\alpha < 1$ then $x \geqq \beta/(1-\alpha)$ is a necessary condition for feasibility.

(iii) If $\alpha > 1$ then $x \leqq \beta/(1-\alpha)$ is necessary.

Obviously, the number $h = \beta/(1-\alpha)$ (in case $\alpha \neq 1$) is the solution of the equation $g = \alpha g + \beta$. Suppose we apply the routine FORWARD around each simple loop and along every simple chain. If either an infeasible loop is discovered or an infeasibility is detected by FORWARD (in the form $\underline{z} > \bar{z}$) then the problem is infeasible; otherwise, we may adjoin all the necessary conditions so obtained to our set of inequalities and that of course will not restrict the set of solutions. By doing this we obtain what Shostak [12] calls a *closure* S' of our set of inequalities. Shostak's main

theorem states that S is feasible if and only if S' does not have any infeasible simple loop nor a simple chain along which FORWARD detects infeasibility. This is the essence of Shostak's algorithm. That algorithm is exponential since it needs to consider all simple loops.

Aspvall and Shiloach obtained a polynomial-time algorithm by considering another extension S^* of S. Specifically, $S^* = S_1^* \cup S_2$ where S_1^* is the set of the most restrictive inequalities in S' with respect to a single variable and S_2 is the original set of inequalities involving exactly two variables. Following Aspvall and Shiloach we denote those most restrictive bounds for a variable x by $x$low and $x$high, i.e., S_1^* consists of the inequalities $x\text{low} \leqq x \leqq x\text{high}$. Once $x$low and $x$high have been found, Aspvall and Shiloach can find a solution, or else recognize infeasibility, in $O(mn^2)$ operations. We shall develop an $O(mn^2 \log m)$ algorithm for finding $x$low and $x$high for a single variable x.

3. The functions $r(g)$ and $r'(g)$. It has already been noted that the bounds obtained at the end of a fixed chain are themselves linear functions of the value g which is assigned to the variable at the start of the chain. Let x be an arbitrary variable. We define $r(g)$ to be the largest lower-bound on x which may be obtained in one of the following ways: (i) Apply FORWARD along any chain of length not greater than n, with the initial bounds $y = \text{lo}(y)$, $\bar{y} = \text{up}(y)$ for all y, (ii) Apply FORWARD around any loop of length not greater than n, starting and ending at x (where x is the selected variable) with the same initial bounds except for $\underline{x} = \bar{x} = g$. Analogously, $r'(g)$ is defined to be the least upper bound on x that may be obtained in such a way. It follows that $r(g)$ is convex piecewise-linear function of g while $r'(g)$ is concave and piecewise linear.

By definition, if g is a feasible value of x (i.e., there is a solution of S in which $x = g$) then, necessarily, $r(g) \leqq g \leqq r'(g)$. The properties of the functions r, r' imply that the set of the values of g such that $r(g) \leqq g \leqq r'(g)$ is convex, i.e., there exist (possibly infinite) numbers a, b such that $r(g) \leqq g \leqq r'(g)$ if and only if $a \leqq g \leqq b$. If this set is empty we take $a = \infty$, $b = -\infty$. On the other hand, if h is either a lower or an upper bound which is hidden in a loop then there exist $\alpha \neq 1$ and β such that $h = \alpha h + \beta$ and either $\alpha g + \beta \leqq r(g)$ for all $g \in [a, b]$ or $\alpha g + \beta \geqq r'(g)$ for all $g \in [a, b]$. Moreover, if h is a bound obtained from a chain then either $h \leqq r(g)$ or $h \geqq r'(g)$ for every g. It thus follows (see Fig. 1) that the endpoints a, b are precisely the most

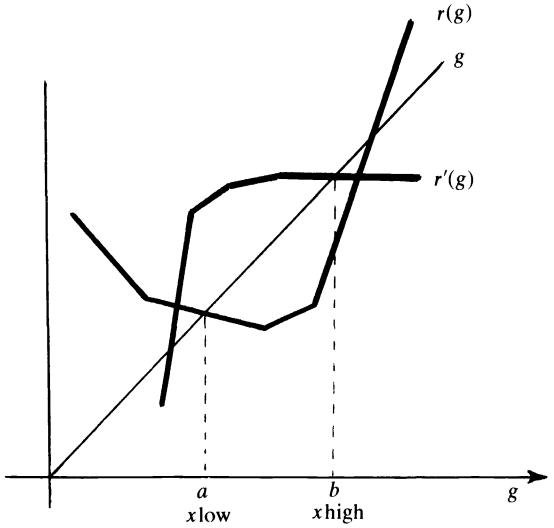


FIG. 1

restrictive bounds that may be obtained either along chains or around loops (all of length no greater than n), i.e., $a = x\text{low}$ and $b = x\text{high}$. In other words, $x\text{low} = \min\{g : r(g) \leqq g \leqq r'(g)\}$ and $x\text{high} = \max\{g : r(g) \leqq g \leqq r'(g)\}$. We shall develop a search algorithm for $x\text{low}$ and $x\text{high}$.

4. A useful generalization. As a matter of fact, we can handle a more general situation which is more convenient to describe. Consider the function $R(g) = \min[r'(g) - g, g - r(g)]$. Note that this function is defined with respect to a variable x. Obviously, $r(g) \leqq g \leqq r'(g)$ if and only if $R(g) \geqq 0$, while R is concave and piecewise linear. We are interested in finding $a = \min\{g : R(g) \geqq 0\}$ and $b = \max\{g : R(g) \geqq 0\}$. Let $R_+(g)$ and $R_-(g)$ denote the slopes of R at g on the right-hand side and on the left-hand side, respectively. Thus, $R_-(g) \geqq R_+(g)$ and this inequality is strict if and only if g is a breakpoint of R. If $R(g)$, $R_+(g)$ and $R_-(g)$ are known at a certain g, then the location of g relative to a and b can be decided according to the following table:

$R(g) \geqq 0$	$a \leqq g \leqq b$
$R(g) > 0, \quad R_-(g) \geqq 0$	$g < a$
$R(g) < 0, \quad R_+(g) \leqq 0$	$g > b$

Note that this table exhausts all possible cases since $R_-(g) \geqq R_+(g)$. Furthermore, if $R_-(g) \geqq 0 \geqq R_+(g)$ and $R(g) < 0$, then R takes on only negative values ($a = \infty, b = -\infty$).

An algorithm for evaluating $r(g)$ and $r'(g)$ (with respect to a variable x) was given by Aspvall and Shiloach [2]. To conform with the notation used in the present paper, we state the following algorithm which is essentially the same as Algorithm 1 in [2].

```
procedure EVAL(g);
  begin
    for each variable y   [ȳ ← up(y); y̲ ← lo(y)];
    x̄ ← min(x̄, g); x̲ ← max(x̲, g);
    for i ← 1 until n do
      begin
        for each y and each ay + bz ≦ c   FORWARD (y, ay + bz ≦ c);
      end
    r ← x̲; r' ← x̄;
  end
```

Clearly, EVAL(g) requires $O(mn)$ arithmetic operations. For our purposes we need to know not only $r(g)$ and $r'(g)$ but also the one-sided slopes of r and r' at g. Thus, we have to modify EVAL a little. Imagine all the quantities $\underline{x}, \bar{y}$ (including $\underline{x}$ and $\bar{x}$) to be themselves functions of g in some neighborhood of a given value. There exists a neighborhood over which all these functions consist of at most two linear pieces with the given g being the unique breakpoint. It is fairly simple to keep track of the slopes of these linear pieces. At the start, every y has both $\bar{y}$ and $\underline{y}$ with slope zero on both sides. The next step is $\bar{x} \leftarrow \min(\bar{x}, g)$. Here we have one of the following cases: (i) If $g < \text{up}(x)$ then $\bar{x}$ has slope unity on both sides. (ii) If $g = \text{up}(x)$ then $\bar{x}$ has slope zero on the right-hand side and slope unity on the left-hand side. (iii) If $g > \text{up}(x)$ then $\bar{x}$ has both slopes equal to zero. Later, when functions

are multiplied by constants (see the routine FORWARD), the slopes are multiplied by the same constants. Adding a constant does not affect the slope. The effect of the "min" operation is also straightforward. Without loss of generality assume we perform $f_3 \leftarrow \min(f_1, f_2)$, where $f_1 \leqq f_2$. The solution is as follows. If $f_1(g) < f_2(g)$ then f_3 inherits its slopes from f_1; otherwise, if $f_1(g) = f_2(g)$ then f_3 inherits the minimum slope on either side of g. Thus, in general, as long as in the evaluation of $R(g)$ the variable g is involved only in comparisons, additions and multiplications by constants, we can evaluate the slopes $R_+(g)$ and $R_-(g)$ with the same computational complexity as that of $R(g)$. In our particular case this is $O(mn)$.

5. Solving $R(g) \geqq 0$. We shall now develop an algorithm for finding a and b. Assume that we have an algorithm for evaluating $R(g)$ such that g itself is involved only in comparisons, additions and multiplications by constants (and R is a concave function of g). In view of the discussion in the preceding section, we assume without loss of generality that this algorithm computes not only $R(g)$ but also the slopes $R_+(g)$ and $R_-(g)$.

We maintain bounds $\underline{a}$, $\bar{a}$, $\underline{b}$, $\bar{b}$ which are repeatedly updated and always satisfy $\underline{a} \leqq a \leqq \bar{a}$ and $\underline{b} \leqq b \leqq \bar{b}$. The initial values are $\underline{a} = \underline{b} = -\infty$ and $\bar{a} = \bar{b} = \infty$. The basic idea is to follow the known algorithm for evaluating R with g being indeterminate; however, g will always be confined to $D = [\underline{a}, \bar{a}] \cup [\underline{b}, \bar{b}]$. Whenever the result of the succeeding step depends on the value of g within D, a test which amounts to one R-evaluation (i.e., with a specific argument g) is performed, in order to update D appropriately. The fundamental principle used here was first introduced in [7] and later applied in [8].

The details are as follows. At the start, the available quantities are the indeterminate g together with several constants, while $D = [-\infty, \infty]$. We distinguish two phases in the computation: Phase 1 lasts as long as $\underline{a} = \underline{b}$ and $\bar{a} = \bar{b}$; when this does not hold any more then we are in Phase 2. Consider a typical point at Phase 1. Assume, by induction on the number of steps since the start, that all the "program variables" are linear functions of g over D, possibly constants. If the next operation is an addition or a multiplication by a constant, then it can be carried out with the indeterminate g over the entire D. Suppose the next operation is a comparison, $f_3 \leftarrow \min(f_1, f_2)$, say. If the linear functions f_1 and f_2 do not intersect over D, or if they coincide over D, then the assignment can be carried out symbolically and f_3 is a linear function of g over D; otherwise, denote the intersection point by g' and assume, without loss of generality, that $f_1(g) < f_2(g)$ for $g < g'$ while $f_2(g) < f_1(g)$ for $g > g'$ $(g \in D)$. At this point we test the value g', i.e., we evaluate $R(g')$, $R_+(g')$ and $R_-(g')$ and update D as follows:

$R(g') \geqq 0$	(enter[1] Phase 2) $\bar{a} \leftarrow g'$; $\underline{b} \leftarrow g'$; $f_3 \leftarrow \min(f_1, f_2)$
$R(g') < 0$ and $R_-(g') \geqq 0$	$\underline{a} \leftarrow g'$; $\underline{b} \leftarrow g'$; $f_3 \leftarrow f_2$
$R(g') < 0$ and $R_+(g') \leqq 0$	$\bar{a} \leftarrow g'$; $\bar{b} \leftarrow g'$; $f_3 \leftarrow f_1$

If Phase 1 continues then all the available quantities remain linear functions of g over the updated D.

[1] Phase 2 will work on the two intervals separately; the assignment will be different but constant over each interval.

When Phase 2 starts we have $\bar{a} = \underline{b}$ and all the quantities consist of at most two linear pieces with the breakpoint occurring at $\bar{a} = \underline{b}$). During Phase 2 we split the computation of a from that of b. Consider, for example, the computation of a. We continue with the evaluation of R, where g is indeterminate but confined to $[\underline{a}, \bar{a}]$. The situation is very similar to that of Phase 1. If g' and f_1, f_2 and f_3 are as before, then the assignments are according to the following table:

$R(g') \geqq 0$	$\bar{a} \leftarrow g'; f_3 \leftarrow f_1$
$R(g') < 0$ and $R_-(g) \geqq 0$	$\underline{a} \leftarrow g'; f_3 \leftarrow f_2$
$R(g') < 0$ and $R_+(g) \leqq 0$	$\bar{a} \leftarrow g'; f_3 \leftarrow f_1$

As a result we have $R(g)$ as a linear function over $[\underline{a}, \bar{a}]$. It is then straightforward to decide which of the following is the case: (i) There is a unique solution to $R(g) = 0$ over $[\underline{a}, \bar{a}]$; this solution is then assigned to a (i.e., $\underline{a} = \bar{a}$). (ii) $R(g) \geqq 0$ for all $g \in [\underline{a}, \bar{a}]$; this is possible only if $\underline{a} = -\infty$, in which case $a \leftarrow -\infty$. (iii) $R(g) < 0$ for all $g \in [\underline{a}, \bar{a}]$; this is possible only if $\bar{a} = \infty$, in which case $a \leftarrow \infty$ and $R(g) < 0$ for every real g (i.e., infeasible system). The computation of b is analogous.

If the evaluation of R at a single g requires T operations, including C comparisons, then the computation of a and b takes $O(\mathrm{CT})$ operations, since it amounts to $O(\mathrm{C})$ evaluations of R (see [7] for a more detailed discussion of this point).

6. Finding $x$low and $x$high. When we solve $r(g) \leqq g \leqq r'(g)$ (equivalently, $R(g) \geqq 0$) according to the scheme presented in the preceding section, we run the routine EVAL with g being indeterminate. However, here we do not have to test every critical value g' right away. Specifically, consider for example the value of $\underline{z}$ which is obtained at the end of the second loop of a single iteration of EVAL (i.e., while i is fixed). As a function of g over D , this is the maximum envelope of the linear functions corresponding to the different inequalities in which z participates together with the previous function corresponding to $\underline{z}$. If there are m_z such inequalities, then we can can find all the breakpoints of the maximum function in $O(m_z \log m_z)$time (see the Appendix of [7]). Thus, the set of all breakpoints produced during one interation can be found and sorted in $O(m \log m)$ time. Assuming that these breakpoints are $g_1 \leqq \cdots \leqq g_q$ $(q = 0(m))$, we may perform a binary search over these q values which amounts to testing only $O(\log\ \mathrm{q})$ of them. If this occurs during Phase 1, then by testing the number $g_{[q/2]}$ we either enter Phase 2 or discard approximately a half of the set of critical values. During Phase 2 each test cuts the set of critical values (lying in $[\underline{a}, \bar{a}]$, say) in half. Thus, the computation of $x$low and $x$high takes n stages during each of which we have to evaluate $r(g)$ and $r'(g)$ at $O(\log m)$ values of g. This amounts to $O(mn^2 \log m)$ arithmetic operations. This procedure needs to be repeated for every other variable so that the bounds $x$low and $x$high are found for all variables x in $O(mn^3 \log m)$ time.

7. Solving S. Let $y$low and $y$high denote the bounds obtained in the previous section. The following routine (which was essentially given by Aspvall and Shiloach [2]) either discovers that S is infeasible or else produces a feasible solution $(x_j = x_j^*, j = 1, \cdots, n)$:

procedure FINAL:
  begin
    for each variable x $[\bar{x} \leftarrow x\text{high}; \underline{x} \leftarrow x\text{low}]$;
    for $j \leftarrow 1$ **until** n **do**
      begin
        for $i \leftarrow 1$ **until** n **do**
        begin
          for each y and $(ay + bz \leqq c)$ FORWARD $(y, ay + bz \leqq c)$;
        end
        if there is a finite ξ such that $x_j \leqq \xi \leqq \bar{x}_j$ **then** $[\underline{x}_j \leftarrow \xi$;
        $x_j^* \leftarrow \xi$; $\bar{x}_j \leftarrow \xi]$ **else return** (INFEASIBLE);
      end
    return$(x_j = x_j^*, j = 1, \cdots, n)$;
  end

The validity of the routine FINAL follows from Shostak's theorem. Since we are now working with the set of inequalities extended so as to include the necessary conditions $x\text{low} \leqq x \leqq x\text{high}$, if no infeasible loops or chains of length n are discovered, then the problem is feasible.

The routine FINAL takes only $O(mn^2)$ operations, i.e., the whole process is dominated by the computation of the bounds $x\text{low}$ and $x\text{high}$ for all the variables. The genuinely polynomial algorithm hence runs in $O(mn^3 \log m)$ operations.

Acknowledgment. The author is grateful to the referees for their helpful comments.

REFERENCES

[1] B. ASPVALL, *Efficient algorithms for certain satisfiability and linear programming problems*, Ph.D. dissertation, Dept. Computer Science, Stanford University, Stanford, CA, August 1980.
[2] B. ASPVALL AND Y. SHILOACH, *A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality*, this Journal, 9 (1980), pp. 827–845.
[3] T.-H. CHAN, *An algorithm for checking PL/CV arithmetic inferences*, Report TR-77-326, Dept. Computer Science, Cornell University, Ithaca, NY, 1977.
[4] G. B. DANTZIG, *Linear Programming and Extensions*, Princeton Univ. Press, Princeton, NJ, 1963.
[5] J. EDMONDS AND R. M. KARP, *Theoretical improvements in algorithmic efficiency for network flow problems*, J. Assoc. Comput. Mach., 19 (1972), pp. 248–264.
[6] L. G. KHACHIYAN, *A polynomial algorithm in linear programming*, Soviet Math. Dokl., 20 (1979), pp. 191–194.
[7] N. MEGIDDO, *Combinatorial optimization with rational objective functions*, Math. OR, 4 (1979), pp. 414–424.
[8] ———, *Applying parallel computations algorithms in the design of serial algorithms*, Proc. IEEE 22nd Symposium on Foundations of Computer Science, 1981, pp. 399-408.
[9] ———, *Is binary encoding appropriate for the problem-language relationship*?, Theoret. Comp. Sci., to appear.
[10] C. G. NELSON, *A $n^{\log n}$ algorithm for the two-variable-per-constraint linear programming satisfiability problem*, Tech. Rep. AIM-319, Dept. Computer Science, Stanford University, Stanford, CA, 1978.
[11] V. R. PRATT, *Two easy theories whose combination is hard*, unpublished manuscript, 1977.
[12] R. SHOSTAK, *Deciding linear inequalities by computing loop residues*, J. Assoc. Comput. Mach., 28 (1981), pp. 769–779.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0011 $01.25/0

VLSI ALGORITHMS FOR THE CONNECTED COMPONENT PROBLEM*

SUSANNE E. HAMBRUSCH†

Abstract. This paper presents algorithms for the connected component problem that are suitable for VLSI implementation and use $o(n^2)$ area. Two VLSI models, which differ in the number of input/output ports allowed to be placed on the boundary of the chip, are considered. It is shown how to achieve a continuous area-time tradeoff of $AT^2 = O(n^4)$ in the range $\Omega(n) = A = o(n^2)$. This tradeoff is optimal for one model, and for the other model continuous tradeoffs of the form $AT = O(n^{5/2})$ in the range $\Omega(n) = A = O(n^{5/4})$ and $AT^3 = O(n^5)$ in the range $\Omega(n^{5/4}) = A = o(n^2)$ are exhibited.

Key words. VLSI algorithms, connected components, area-time tradeoffs, d-dimensional meshes

1. Introduction. Advances in the fabrication technology of VLSI electronics have motivated the study of parallel algorithms on networks with simple and input-independent interconnections [2], [7], [13]. Within the VLSI model, a number of problems have been shown to exhibit tradeoffs between the time to solve the problem and the area in which the network is laid out [1], [4], [15]. In this paper we examine VLSI algorithms for the connected component problem: Given an undirected graph $G = (V, E)$, where $|V| = n$, $|E| = e$, determine the connected components of G (i.e. two vertices $v_1, v_2 \in V$ are in the same connected component if and only if there is a path between v_1 and v_2 in G). We present several algorithms with area-time performance superior to previous solutions, and which raise several intriguing questions about the relation between time and area in graph problems.

Hirschberg et al. [5], Savage and Ja'Ja' [6] and Wyllie [16] have solved this problem in time $O((\log n)^2)$, using $O(n^2/\log n)$, $O(e+n \log n)$ and $O(n+e)$ processing elements, respectively. However, their model of computation, in which processing elements share a common main memory, is not suitable for VLSI because of the number of interconnections. Kung et al. [4] have given a systolic algorithm for the directed transitive closure problem, which can be used to solve our problem. The algorithm runs in $O(n^3/k^2)$ time and uses $O(k^2)$ processing elements, $1 \leqq k \leqq n$, but would require $O(n^2)$ locations to store the adjacency matrix. Nassimi and Sahni [12] implement Hirschberg's algorithm on a d-dimensional mesh-connected network of n processing elements in time $O((d+g)n^{1/d} \log n^{1/d})$, where g is the maximum degree allowed for any vertex in G. They do not consider the area of their network, but each of the n processing elements must have g registers, each containing $\log n$ bits, indicating at least n^2 area to handle general graphs.

The algorithms cited above assume the adjacency matrix or adjacency lists are stored in the network, which requires $O(n^2)$ and $O(e)$ storage locations, respectively. We present algorithms for solving the connected component problem on networks of $o(n^2)$ area, which rules out storing the edges of the graph in the network explicitly. The algorithms run on d-dimensional mesh-connected networks and allow the graph to be input in a very general form. Lipton and Valdes [10] more recently have presented an algorithm that finds the connected components on binary tree network for a more restrictive form of graph representation.

In § 2 we define the two models of computation, which differ only in the number of input/output ports allowed on the chip. In § 3 we develop the basic algorithm for the connected component problem, which runs in time $O(n^{3/2})$ on a 2-dimensional

* Received by the editors April 23, 1981, and in revised form June 28, 1982.
† Computer Science Department, Pennsylvania State University, University Park, Pennsylvania 16802.

mesh-connected network of n processing elements. In § 4 we show how to solve the connected component problem on a d-dimensional mesh-connected network, in time $O(n^{1+1/d})$ using area $O(n^{2-2/d})$, where d can be any real number $\geqq 2$. This gives a continuous tradeoff of $AT^2 = O(n^4)$ in the range $\Omega(n) = A = o(n^2)$, which is optimal for the first model. In § 5 we develop algorithms for the second model, which exhibit two forms of continuous area-time tradeoff: the first one, of $AT = O(n^{5/2})$ in the range $\Omega(n) = A = O(n^{5/4})$, is obtained by modifying the network and the algorithm given in § 3; the second tradeoff, of $AT^3 = O(n^5)$ in the range $\Omega(n^{5/4}) = A = o(n^2)$, is achieved by performing the same modifications on the d-dimensional mesh-connected network.

2. Model of computation. We consider models similar to those developed in the literature [4], [7], [13] and which conform to the basic VLSI restrictions in Mead and Conway [11]. The main assumptions are given below.

1. The *processing elements* (PE's) of the chip contain a constant number of registers and are able to execute a simple set of instructions. Each register can hold a number of size at most n, where n is the number of vertices in the graph. Each PE is connected to a constant number of other PE's. The PE's operate synchronously.

2. The chip communicates with the outside world through the input/output (I/O) ports, which lie on the boundary of the chip. In one model, the *limited boundary model*, the number of I/O ports can be at most the square root of the total area of the chip. With this restriction the chip can be laid out so that the bounding rectangle has a constant aspect ratio (= width/length). In the other model, *the boundary model*, I/O ports can be placed everywhere on the boundary of the chip.

3. Each input is read once and each output is generated once. The locations at which the inputs arrive and the outputs are generated are independent of the input data. The time at which the inputs arrive can depend on the input data; i.e. the chip is allowed to determine when to read the next input sequence.

4. A constant number of parallel layers can be used, and the layout on each layer is supposed to be planar.

Our cost measures are *time* and *area*. Two definitions of time have been used in papers on VLSI. One defines time as the number of *steps* required to generate all the outputs, where one step is either an operation on two bits of a PE, or the transmission of a bit from one PE to an adjacent PE. This definition is generally used in papers on lower bounds in VLSI [2], [14], [15]. Many algorithms for VLSI networks perform operations on entire registers rather than on bits of the registers, and this motivated the second definition. Here time is the total number of *cycles* needed to generate all the outputs, where one cycle is either an operation on two registers of a PE, or the transmission of the content of a register of a PE to an adjacent PE. See [4], [7], [10] and [13].

The area is the space necessary to lay out the PE's with their interconnections on a small, constant number of layers. When time is measured in terms of steps a bit is considered to have unit size, and thus each PE occupies $\log n$ area. When time is measured in terms of cycles we consider a register to have unit size, and thus each PE occupies a constant number of units of area. Throughout this paper we will measure time in terms of cycles. Thus, in order to compare our results with results obtained in the other model, the area needs to be multiplied by a factor of $\log n$. The time needs to be multiplied by a factor of $(\log n)^{1/2}$ or $\log n$, depending on whether or not in the model PE's of $\log n$ area are interconnected by wires of $(\log n)^{1/2}$ or constant bandwidth.

In our algorithms we charge unit time for the transmission of the content of a register to an adjacent PE independent of the length of the interconnecting wire. Recent papers [1], [3] studied VLSI models where time is proportional to the length of the interconnecting wire. Results in [1] suggest that under the current technology unit time is a reasonable assumption.

The graph G will be represented in the *form of edges*: each one of the k input ports of the chip will read e/k edges (i, j), $(i, j) \in E$, of the graph. The output is a vector C of size n, where the *component number* C_i of vertex i is the smallest vertex reachable from vertex i, $1 \leqq i \leqq n$.

3. A 2-dimensional mesh. We show how to solve the connected component problem in time $O(n^{3/2})$ using a 2-dimensional mesh-connected network of n PE's, that can be laid out in $O(n)$ area. It can be shown that in either model the lower bound on the number of PE's needed is n. For simplicity we assume n to be a perfect square; the modifications to be done when n is arbitrary are straightforward (use a network of size $\lceil n^{1/2} \rceil \times \lceil n^{1/2} \rceil$ that contains dummy PE's). See Fig. 1.

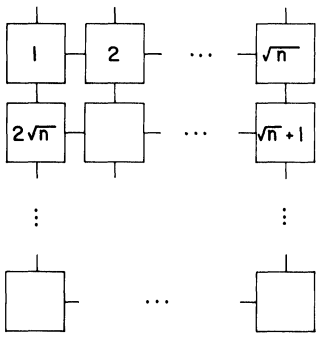


FIG. 1

In our algorithms the PE's are indexed in snake-like row-major order (i.e. row-major order where the even rows are reversed). This allows PE_i to transmit to PE_{i-1} and PE_{i+1} in constant time, provided they exist. Each PE_i has 5 registers:

C_i contains the current component number of vertex i;

X_i, Y_i hold initially an input edge (x_i, y_i), later the component number of x_i and y_i, respectively;

$A1_i$, $A2_i$, $A3_i$ are three auxiliary registers.

The network is initialized with every vertex being a component by itself, $C_i = i$, $1 \leqq i \leqq n$.

In the *input phase* of algorithm CONNECT each input port reads the next $n^{1/2}$ edges and propagates them vertically down to the PE's in the same column. After the input phase each PE holds an edge (x_i, y_i). The following *lookup phase* determines for each edge (x_i, y_i) whether or not it causes two components to get merged. First the current component numbers of x_i and y_i are found. An obvious, but inefficient strategy is to let all PE_i, $1 \leqq i \leqq n$, look up the component number of x_i by sending x_i to PE_{x_i}, updating x_i to C_{x_i}, and sending it back to PE_i. The path from PE_i to PE_{x_i} can be found, for example, by first moving horizontally until the column containing PE_{x_i} is hit, then going vertically to PE_{x_i} (e.g., if PE_i is in row p and column r, PE_{x_i} is in row s and column q, p and s are odd, and $p > s$, $r < q$, then move $q - r$ steps horizontally and $p - s$ steps vertically). Since it is possible that different PE's want to look up the same component numbers or look up component numbers on the same communication

path, the strategy above can lead to problems of congestion. We overcome this problem by using sorting algorithms to control the flow of information. Recall that there are efficient sorting algorithms for 2-dimensional mesh-connected networks, e.g., [8] presents an algorithm that uses $O(n^{1/2})$ parallel steps. Our strategy for the lookup phase is to first sort the PE's of the network according to the x_i's. If two or more PE's want to look up the same component number, only one PE does the actual lookup and it propagates the result of the lookup to the other PE's. The *merging phase* considers all the edges that merge components and merges up to $n^{1/2}$ components simultaneously.

Let C_i, C_j be the current component numbers of the vertices i and j before the kth input sequence is read, $C_i \neq C_j$. Assume the $(k+1)$st input sequence contains the edge (i, j). If $C_i < C_j$, we want to change, in the merging phase of the $(k+1)$st input sequence, every occurrence of C_j to C_i, and we denote this by $C_j \to C_i$ and call it a *change* (similarly for $C_i > C_j$).

ALGORITHM CONNECT (outline)
1. *Initialization*
//each vertex is a component by itself//
$C_i = i$, $1 \leqq i \leqq n$;
repeat until all edges have been read
2. *Input phase*
each one of the $n^{1/2}$ input ports reads $n^{1/2}$ edges and propagates them vertically down to the PE's in the same column;
3. *Lookup phase*
//PE_i holds edge (x_i, y_i), $1 \leqq i \leqq n$,—determine the//
//current component number of x_i and y_i, and the//
//components to be merged//
(i) call DET-COMPNR to determine the component numbers;
(ii) determine which of the PE's contain changes;
(iii) if some changes occur more than once in the network, remove all but one occurrence of them;
4. *Merging phase*
repeat until all changes have been processed
(i) find the next set of (at most) $n^{1/2}$ changes and call UPDATE
(ii) send the updated changes to all the PE's and update their component number and change if necessary;
endrepeat;
endrepeat;
end CONNECT;

The following algorithm determines the current component numbers of the vertices of the input edges.

ALGORITHM DET-COMPNR;
//After the input phase each PE_i contains an edge//
//(x_i, y_i), $1 \leqq i \leqq n$, and the current component numbers C_{x_i}//
//and C_{y_i} are determined; x_i is stored in register X_i, y_i is stored in register Y_i//
(i) sort the registers X_i of the PE's of the network in increasing order such that $x_{i-1} \leqq x_i$, $1 \leqq i \leqq n$, after the sort;
(ii) //if edges have the same first entry x_i, determine//
//the PE's with smallest index containing such a x_i and call them the leaders//

for each PE_i, $1 \leqq i \leqq n$ **pardo**
if $x_i > x_{i-1}$ **then**
store $(i, x_i, 0)$ in the auxiliary registers
//PE_i contains a leader//
fi
odpar
make PE_1 be a leader by storing $(1, x_1, 0)$ in the auxiliary registers of PE_1;
(iii) //the leaders $(i, x_i, 0)$ look up the component//
//numbers of x_i and record it in the third entry//
while not all $(i, x_i, 0)$ have arrived at PE_{x_i} **do**
for each PE_j containing a leader $(i, x_i, 0)$ **pardo**
(let PE_j be in row p, column r and PE_{x_i} in row s, column q)
if $s = p$ **then** send $(i, x_i, 0)$ along row s to the PE closer to column q
else send $(i, x_i, 0)$ along column r to the PE closer to row s;
fi
odpar
endwhile;
(iv) //when all leaders $(i, x_i, 0)$ have arrived at//
//PE_{x_i} record the component number of x_i//
for each leader **pardo** $(i, x_i, 0) = (i, x_i, C_{x_i})$ **odpar**;
(v) send the leaders (i, x_i, C_{x_i}) back to PE_i;
(vi) propagate C_{x_i} to all the other PE_r with $x_r = x_i$ and change x_r to C_{x_i};
(vii) look up the component numbers of the y_i's by performing steps (i) to (vi) on the y_i's;
end DET-COMPNR;

In step 3(ii) each PE examines the current component numbers obtained in step 3(i) and determines the changes as shown in the following program:

for each PE_i, $1 \leqq i \leqq n$, **pardo**
//PE_i reads the edge (x_i, y_i); the component number of//
//x_i is stored in register X_i, the component number of y_i is stored in register y_i
if $C_{x_i} = C_{y_i}$ **then** $x_i = y_i = 0$ **fi**
//vertices x_i and y_i are already in the same component//
if $C_{x_i} > C_{y_i}$ **then** $C_{x_i} \to C_{y_i}$ is a change
else interchange registers X_i and Y_i, $C_{y_i} \to C_{x_i}$ is a change
fi
odpar;

In order to remove in step 3(iii) the multiple changes contained in the network we sort the PE's of the network according to the changes, and then perform:

for each PE_i, $2 \leqq i \leqq n$ **pardo**
//if PE_i contains the same change as PE_{i-1} remove the change in PE_i//
if $(x_{i-1} = x_i) \& (y_{i-1} = y_i)$ **then** $x_i = y_i = \phi$ **fi**
//set the registers containing a multiple change to 0//

odpar;

Let $x_1 \to y_1, \cdots, x_s \to y_s$ be the s changes found in step 4(i) of the merging phase, $s \leqq n^{1/2}$. Let them be stored in $PE_1, \cdots, PE_s$ of the network. The changes represent a not-necessarily connected graph that can contain transitive and multiple edges. In

order to send the changes through the network and merge components simultaneously, the changes have to be updated first: When merging the components the ith change $x_i \to y_i$ will arrive at all PE's after the changes $1, \cdots, i-1$. Thus if the changes $1, \cdots, i-1$ change the component number of vertex x_i to x_j, $x_j < x_i$, (or y_i to y_j, $y_j < y_i$), we need to update in change i x_i to x_j, (or y_i to y_j), before sending out change i. The updating is done by letting the changes "ripple" to the right in the first row of the network and is described in algorithm UPDATE. Thus algorithm UPDATE detects the transitive and multiple edges. Figure 2 gives an example of how the updating is performed. (The changes $4 \to 3$, $4 \to 2$, $2 \to 1$, $3 \to 2$ get updated to $4 \to 3$, $3 \to 2$, $2 \to 1$.)

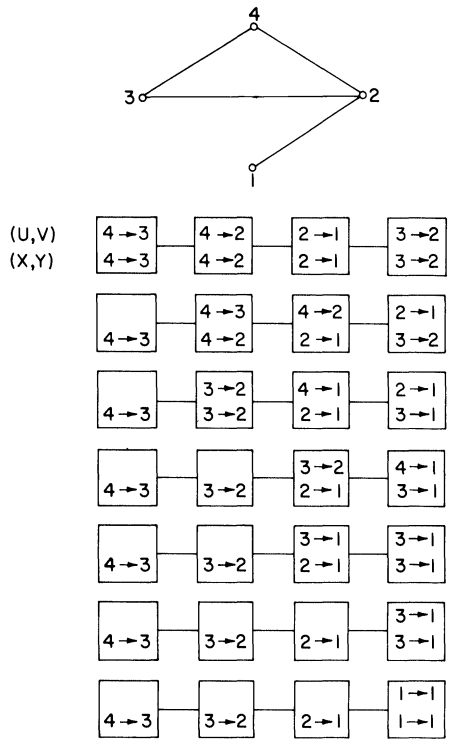


FIG. 2

ALGORITHM UPDATE;
//Update the component numbers x_i and y_i of the ith//
//change to the index of the component the vertices//
//x_i and y_i are in after the $i-1$ preceding changes have been processed//
//let (u_i, v_i) be two auxiliary registers in PE_i//
for each PE_i, $1 \leqq i \leqq s$, **pardo** $(u_i, v_i) = (x_i, y_i)$ **odpar**;
for $i = 1$ **to** s **do**
    for each PE_j, $i \leqq j \leqq s-1$, **pardo**
        send (u_j, v_j) of PE_j to PE_{j+1} and store in (u_{j+1}, v_{j+1})
    odpar;

    //the ith change is updated, continue updating change $i+1, \cdots, s$//
    for each PE_j, $i+1 \leqq j \leqq s$, **pardo**
        case of:
            //the jth change merges two components already//
            //merged by the preceding changes//

```
                (x_i = y_i): do nothing
                ((u_j = x_j) & (v_j = y_j)): x_j = y_j   //change x_j → y_j into y_j → y_j//
                //in the changes j+1, ···, s it is already recorded//
                //that x_j and y_j get merged; every occurrence of//
                //x_j is changed into y_j//
                (v_j = x_j): v_j = y_j          //change u_j → v_j into u_j → y_y//
                (v_j = y_j): do nothing
                (u_j = y_j): y_j = v_j          //change x_j → y_j into x_j → v_j//
                (u_j = x_j): if   v_j > y_j
                             then//change u_j → v_j and x_j → y_j into v_j → y_j//
                                  u_j = v_j, v_j = y_j, x_j = u_j
                             else //change u_j → v_j and x_j → y_j into y_j → v_j//
                                  u_j = y_j, x_j = u_j, y_j = v_j;
        end case;
        odpar;
    endfor;
    end UPDATE;
```

After the updating the s changes are sent out to all the PE's in the network such that change i arrives at each PE after change $i-1$, $2 \leqq i \leqq s$. Assume change $x_i \to y_i$ arrives at PE_j. Then update the change $x_j \to y_j$ in PE_j if $x_j = x_i$ or $y_j = x_i$, and if $C_j = x_i$ change C_j to y_i.

LEMMA. *Algorithm* UPDATE *updates the s changes,* $s \leqq n^{1/2}$, *in* $O(n^{1/2})$ *time, so that* x_i *and* y_i *of the ith change,* $1 \leqq i \leqq s$, *contain the index of the component they are in after the* $i-1$ *preceding changes are performed.*

Proof. The s changes stored in the first row of the network "ripple" simultaneously to the right using the horizontal interconnections. The changes $i-1, \cdots, 2, 1$ pass through PE_i containing change $x_i \to y_i$ in this order. After the kth change, $1 \leqq k \leqq i-1$, has passed through PE_i, x_i and y_i contain the index of the component they are in after the changes $k, k+1, \cdots, i-1$ are performed. Thus after all $i-1$ changes have passed through PE_i, x_i and y_i contain the desired value. Change i arrives at all PE_j, $j > i$, before change k, $1 \leqq k \leqq i-1$, and it might be necessary to update change k when it passes through PE_i (since in PE_j every occurrence of x_i gets changed to y_i before change k arrives). When change 1 passes through PE_s, which requires $O(s) = O(n^{1/2})$ time, the updating is completed. □

THEOREM 1. *Algorithm* CONNECT *finds the connected components of an undirected graph G in time* $O(n^{3/2})$ *and uses* $O(n)$ *area.*

Proof. When the ith input sequence is read, $i \geqq 1$, the network has computed the connected components of the previously read $n(i-1)$ edges. We collapse two components x_i' and y_i' if and only if we have read an input edge (x_i, y_i) such that before the merge is performed we had $C_{x_i} = x_i'$ and $C_{y_i} = y_i'$. The component numbers and changes are determined in the lookup phase. If more than one change is found, algorithm UPDATE of the merging phase updates the changes such that they can be performed in parallel (see lemma). This shows correctness.

The time bound is obtained as follows. The initial values of the PE's can be generated in $O(n^{1/2})$ time. Reading an input sequence of n edges requires $O(n^{1/2})$ time. Each one of the steps 3(i), (ii) and (iii) of the lookup phase can be done in $O(n^{1/2})$ time. Since we can read at most n input sequences, the overall time spent in the input and lookup phase is $O(n^{3/2})$. In the worst case only two components are combined in the merging phase and no parallelism is used in step 4(ii). Since we

cannot merge more than $n-1$ components and the merging of two components requires $O(n^{1/2})$ time, the overall time spent in the merging phase is $O(n^{3/2})$.

Since each PE is considered to occupy a constant number of units of area, the 2-dimensional mesh-connected network can be laid out in $O(n)$ area. □

THEOREM 2. *A lower bound on the time required to solve the connected component problem on a limited boundary network of area A is* $\Omega(n^2/A^{1/2})$.

Proof. The limited boundary network has $O(A^{1/2})$ input ports and thus at most $O(A^{1/2})$ input edges can be read at each clock pulse. Hence the time needed to read n^2 input edges is $\Omega(n^2/A^{1/2})$. □

COROLLARY 1. *The time bound of* $O(n^{3/2})$ *achieved by Algorithm* CONNECT *is optimal for a limited boundary network of* $O(n)$ *area.*

4. Higher dimensional meshes. The algorithm for the connected component problem described in the previous section has a running time of $O(n^{3/2})$. In this time bound the factor $n^{1/2}$ reflects the time necessary to sort and route information in the 2-dimensional mesh-connected network, and the algorithm can be improved by using a faster sorting and routing algorithm. In a d-dimensional mesh-connected network we can sort and route in time $O(n^{1/d})$, $d \geqq 2$ [8]. Using the same techniques as in algorithm CONNECT we can obtain an algorithm, CONNECTD, which solves the connected component problem on a d-dimensional mesh-connected network in time $O(n^{1+1/d})$.

In a d-dimensional network, each of the n PE's is connected to its $2d$ nearest neighbors in each of the d dimensions. The network has $n^{1-1/d}$ input ports and in Theorem 3 we show how to lay it out in $O(n^{2-2/d})$ area.

For nonintegral values of d, $d = d' + a$, $0 \leqq a < 1$, the d-dimensional mesh-connected network is defined to have $d'+1$ dimensions and size $n^{1/d} \times \cdots \times n^{1/d} \times n^{a/d}$. Since $1/d > a/d$ we can sort this network in $O(n^{1/d})$ time. Figure 3 shows a 2.5-dimensional mesh for $n = 32$. It has size $n^{2/5} \times n^{2/5} \times n^{1/5}$, i.e. it consists of $n^{2/5}$ $(=4$ in the example) 2-dimensional meshes, each of size $n^{2/5} \times n^{1/5} = (4 \times 2)$, ($n^{2/5}$ rows, $n^{1/5}$ columns). Each PE in a row of the 2-dimensional mesh is connected to the corresponding PE in the two adjacent 2-dimensional meshes. The network can be laid out in $n^{2/5} \cdot n^{2/5} \cdot n^{1/5} \cdot n^{1/5} = n^{6/5}$ area.

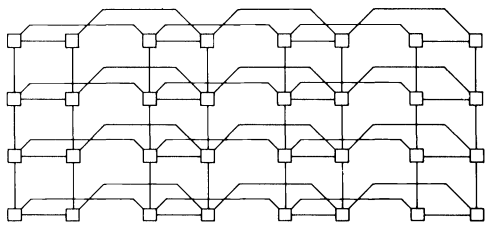


FIG. 3

The initialization of algorithm CONNECTD is done as in the 2-dimensional case. In the *input phase* each one of the $n^{1-1/d}$ input ports reads $n^{1/d}$ edges and propagates so that each PE holds an edge after the input phase. The *lookup* and *merging phases* consist of the same steps as in CONNECT. Each step can be done in $O(n^{1/d})$ time.

THEOREM 3. *The connected component problem can be solved on a d-dimensional mesh-connected network in time* $O(n^{1+1/d})$ *using area* $O(n^{2-2/d})$, *where d can be any real number* $\geqq 2$. *This gives a continuous tradeoff of* $AT^2 = O(n^4)$ *in the range* $\Omega(n) = A = o(n^2)$.

Proof. The proof of the $O(n^{1+1/d})$ time bound is analogous to the proof of the time bound in Theorem 1; (replace $n^{1/2}$ by $n^{1/d}$). The bound on the area is proven by induction on d'. Let $d = d' + a$ be the dimension, $d' \geqq 2$, $0 \leqq a < 1$. For $d' = 2$ the network consists of $n^{1/d}$ blocks, where each block is a 2-dimensional mesh-connected network of size $n^{1/d} \times n^{1-2/d}$. Each block has $n^{1/d}$ rows and $n^{1-2/d}$ columns.

Row i, $i \leqq i \leqq n^{1/d}$, of the jth block, $1 \leqq j \leqq n^{1/d}$, has connections to row i of the blocks $j+1$ and $j-1$, provided they exist (see Fig. 4). To lay out these connections we need $n^{1-2/d}$ additional space between two rows of each block. The total space needed for the network is $n^{1/d} \cdot n^{1-2/d} \cdot n^{1-2/d} \cdot n^{1/d}$ which is $O(n^{2-2/d})$.

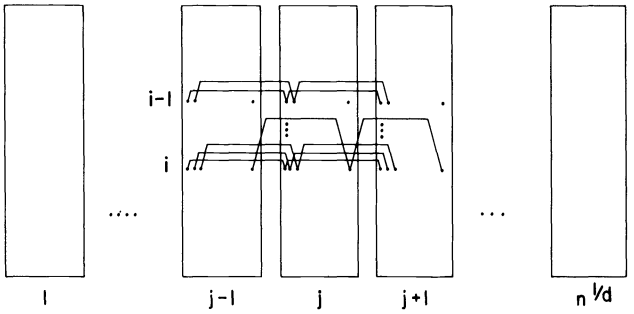


FIG. 4

For $d' > 2$ the network consists of $n^{1/d}$ blocks, where each block is a d'-dimensional mesh-connected network of size $n^{1/d} \times \cdots \times n^{1/d} \times n^{a/d}$. It has $n^{1/d}$ PE's in a column and $n^{1-2/d}$ PE's in a row. Each block can be laid out in $O((n^{(d-1)/d})^{2-2/(d-1)}) = O(n^{2-4/d})$ area, where the space between two rows of PE's is $O(n^{1-3/d})$. To lay out the connections for the $(d'+1)$st dimension we need additional $n^{1-2/d}$ space between two rows of PE's in each block, and the total space necessary between two rows is $O(n^{1-3/d} + n^{1-2/d}) = O(n^{1-2/d})$. Thus the area for the d-dimensional mesh-connected network is $O(n^{1/d} \cdot n^{1-2/d} \cdot n^{1-2/d} \cdot n^{1/d})$, which is $O(n^{2-2/d})$. □

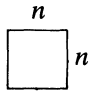
We have just learned that Leiserson [9] obtained a similar result for the layout of the d-dimensional mesh-connected network when d is an integer by using a divide-and-conquer approach.

Using Theorems 2 and 3 we get:

COROLLARY 2. *For the limited boundary network, an optimal tradeoff for the connected component problem of* $AT^2 = O(n^4)$ *in the range* $\Omega(n) = A = o(n^2)$ *can be achieved.*

5. Better tradeoffs in the less restricted model. When the number of I/O ports is not limited to $O(A^{1/2})$, we can solve the connected component problem in time $O(n^{3/2}/k)$ using $O(nk)$ area, $1 \leqq k \leqq n^{1/4}$, and achieve a tradeoff of $AT = O(n^{5/2})$ in the range $\Omega(n) = A = O(n^{5/4})$. We sketch the network that achieves these bounds.

Denote by



the 2-dimensional mesh-connected network described in § 3, and call it a block. Then the new network is as shown in Fig. 5.

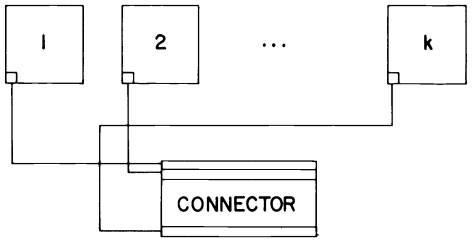


FIG. 5

The network consists of k blocks and a connector, a 2-dimensional mesh-connected network of size $n^{1/2} \times k$. The lowest and leftmost PE of block i is connected to the leftmost PE in row i of the connector.

Each block is initialized like the mesh in § 3, $C_i^j = i$, $1 \leqq i \leqq n$, $1 \leqq j \leqq k$. The network has $kn^{1/2}$ input ports and each one will read $e/kn^{1/2}$ edges. In the *input phase* each block reads n edges as described in algorithm CONNECT. In the *lookup phase* the blocks again work independently and determine the changes by using the algorithm given in CONNECT. The *merging phase* differs from the one in CONNECT: Block i, $1 \leqq i \leqq k$, enters up to $n^{1/2}$ changes into the ith row of the connector. If none of the k blocks has entered $n^{1/2}$ changes, we sort the connector to find a number of distinct changes as large as possible, but not exceeding $n^{1/2}$. After updating the changes in the connector (by using the algorithm UPDATE of § 3), we send the changes from the connector back to the blocks, where we perform up to $n^{1/2}$ merges simultaneously.

ALGORITHM CONNECTK (outline);
1. *Initialization*
//each vertex is a component by itself//
$C_i^j = i$, $1 \leqq i \leqq n$, $1 \leqq j \leqq k$;
repeat until all edges have been read
2. *Input phase*
for each block i, $1 \leqq i \leqq k$ **pardo**
read the next n input edges (as in CONNECT)
odpar;
3. *Lookup phase*
for each block i, $1 \leqq i \leqq k$ **pardo**
determine the changes in block i (as in the lookup phase of CONNECT)
odpar;
4. *Merging phase*
repeat until all changes are processed
(i) **for** each block i, $1 \leqq i \leqq k$ **pardo**
enter up to $n^{1/2}$ distinct changes into the ith row of the connector
odpar;
(ii) **if** every block entered fewer than $n^{1/2}$ changes
then sort the connector according to the changes in it and remove multiple changes
fi;

(iii) let $x_1 \to y_1, \cdots, x_s \to y_s$ be the distinct changes in the connector; (either entered by one block, $(s = n^{1/2})$, or found in step (ii)); call UPDATE;

(iv) duplicate the updated changes stored in one row of the connector to the other $k-1$ rows;
send the changes in the ith row of the connector to block i, where components are merged as described in CONNECT;

endrepeat;

endrepeat;

end CONNECTK;

THEOREM 4. *Algorithm* CONNECTK *finds the connected components of an undirected graph G in time $O(n^{3/2}/k)$ and uses $O(nk)$ area, $1 \leqq k \leqq n^{1/4}$. This gives a continuous tradeoff of $AT = O(n^{5/2})$ in the range $\Omega(n) = A = O(n^{5/4})$.*

Proof. Besides having more parallelism for the same operations, algorithm CONNECTK differs from CONNECT in the merging phase, where the blocks combine the changes found independently and try to determine up to $n^{1/2}$ distinct changes.

Initialization, input and lookup phase can be done in $O(n^{1/2})$ time. Since we can read at most n^2/nk input sequences, the overall time spent in input and lookup phase is $O(n^{3/2}/k)$. Steps 4(i)–(iv) of the merging phase can be done in $O(n^{1/2})$ time. If we obtained $n^{1/2}$ changes in step 4(ii) we will merge at least $n^{1/4}$ components: assume we are dealing with the worst case and the $n^{1/2}$ changes describe a complete graph on $n^{1/4}$ vertices. Then after the updating only $n^{1/4}$ distinct changes will be left, and thus the $n^{1/2}$ initial changes merge only $n^{1/4}$ components. If we obtained fewer than $n^{1/2}$ changes in step 4(ii), we will read nk input edges after the current changes are processed. We either merge at least $n^{1/4}$ components or read nk edges after step 4(iv) is completed. Since we can merge at most $n-1$ components and read at most n/k input sequences, the overall time of CONNECTK is $(n/n^{1/4})n^{1/2} + (n/k)n^{1/2}$ which is $O(n^{3/2}/k)$ for $1 \leqq k \leqq n^{1/4}$.

The area necessary to lay out the k blocks is $O(nk)$ and the connector requires $O(kn^{1/2})$ area, $1 \leqq k \leqq n^{1/4}$. Therefore the blocks, the connector can be laid out in $O(nk)$ area. (Since the connector and the blocks perform the same set of operations, it is possible to declare one block to be the connector.) □

Using k d-dimensional blocks and a d-dimensional connector we can achieve time $O(n^{1+1/d}/k)$.

THEOREM 5. *The connected component problem can be solved in time $O(n^{1+1/d}/k)$ using area $O(kn^{2-2/d})$, where $1 \leqq k \leqq n^{1/2d}$, d a real number $\geqq 2$. For $k = n^{1/2d}$ this gives a continuous tradeoff of $AT^3 = O(n^5)$ in the range $\Omega(n^{5/4}) = A = o(n^2)$.*

Proof. Similar to the proof of Theorem 4. □

Acknowledgments. I would like to thank Joseph Ja'Ja' for introducing me to this area and Greg Frederickson for many helpful and useful discussions.

REFERENCES

[1] G. BILARDI, M. PRACCHI AND F. P. PREPARATA, *A critique and an appraisal of VLSI models of computation*, in Proc. CMU Conference on VLSI Systems and Computations, 1981, pp. 81–88.

[2] R. P. BRENT AND H. T. KUNG, *The area-time complexity of binary multiplication*, J. Assoc. Comput. Mach., 28 (1981), pp. 521–534.

[3] B. CHAZELLE AND L. MONIER, *A model of computation for VLSI with related complexity results*, in Proc. 13th annual ACM Symposium on Theory of Computing, 1981, pp. 318–325.

[4] L. J. GUIBAS, H. T. KUNG AND C. D. THOMPSON, *Direct VLSI implementations for combinatorial algorithms*, in Proc. of Conference on VLSI Technical Design and Fabrication, California Institute of Technology, Pasadena, CA, 1979, pp. 509–525.

[5] D. S. HIRSCHBERG, A. K. CHANDRA AND D. V. SARWATE, *Computing connected components on parallel computers*, Comm. ACM, 22 (1979), pp. 461–464.

[6] C. SAVAGE AND J. JA'JA', *Fast efficient parallel algorithms for some graph problems*, this Journal, 4 (1981), pp. 682–691.

[7] H. T. KUNG AND CH. E. LEISERSON, *Systolic arrays for VLSI*, in Introduction to VLSI systems, C. Mead and L. Conway, eds., Addison-Wesley, Reading, MA, 1980, pp. 260–292.

[8] H. T. KUNG AND C. D. THOMPSON, *Sorting on a mesh-connected parallel computer*, Comm. ACM, 20 (1977), pp. 263–271.

[9] C. E. LEISERSON, *Area efficient graph layouts*, Proc. of 21st IEEE Conference on Foundations of Computer Science, 1980, pp. 270–281.

[10] R. J. LIPTON AND J. VALDES, *Census function: An approach to VLSI upper bounds*, Proc. 22nd IEEE Conference on Foundations of Computer Science, 1981, pp. 13–22.

[11] C. MEAD AND L. CONWAY, eds., *Introduction to VLSI systems*, Addison-Wesley, Reading, MA, 1980.

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

[13] F. P. PREPARATA AND J. E. VUILLEMIN, *The cube connected cycles: A versatile network for parallel computation*, Proc. 20th IEEE Conference on Foundations of Computer Science, 1979, pp. 140–147.

[14] C. D. THOMPSON, *Area-time complexity for VLSI*, Proc. 11th ACM Symposium on Theory of Computing, 1979, pp. 81–88.

[15] J. E. VUILLEMIN, *A combinatorial limit to the complexity of VLSI circuits*, in Proc. 21st IEEE Conference on Foundations of Computer Science, 1980, pp. 294–300.

[16] J. C. WYLLIE, *The complexity of parallel computation*, Ph.D. thesis, Cornell University, Ithaca, NY, 1979.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0012 $01.25/0

INITIAL AND FINAL ALGEBRA SEMANTICS FOR DATA TYPE SPECIFICATIONS: TWO CHARACTERIZATION THEOREMS*

J. A. BERGSTRA† AND J. V. TUCKER‡

Abstract. We prove that those data types which may be defined by conditional equation specifications and final algebra semantics are exactly the cosemicomputable data types—those data types which are effectively computable, but whose inequality relations are recursively enumerable. And we characterize the computable data types as those data types which may be specified by conditional equation specifications using both initial algebra semantics and final algebra semantics. Numerical bounds for the number of auxiliary functions and conditional equations required are included in both theorems.

Key words. data types, algebraic specifications, conditional equations, initial algebra semantics, final algebra semantics, computable, semicomputable and cosemicomputable algebras

Introduction. Suppose you have it in mind to define a data type by means of a set of operators Σ whose behavior is to be governed by a set of axioms E. Then *initial* and *final algebra semantics* represent two distinct, though natural, ways of settling upon a unique meaning for the specification (Σ, E) when the axioms E are written in certain algebraic normal forms. As its semantics, they each assign to (Σ, E) a many-sorted algebra, unique up to isomorphism, from the class $\mathrm{ALG}(\Sigma, E)$ of all algebraic systems of signature Σ which satisfy the properties prescribed by E. Viewed from the proof theory of the axioms E, initial algebra semantics formalize the decision that two formal syntactic expressions, or terms, t, t' over Σ should be semantically equivalent if, and only if, $t = t'$ can be *proved* from the axioms E. While final algebra semantics allows t, t' to be semantically identified as long as $t = t'$ does not *contradict* the requirements of E, or—as one says in the terminology of model theory—$t = t'$ is *consistent* with E.

Both techniques have been widely discussed in the literature devoted to the design of programming languages with varying degrees of exactness and approval; and it seems fair to say that most theoretical and practical work on algebraic data types can be placed in one or other of these opposing initial and final camps, usually the former. For example, looking at the origins of the algebraic specification methods, one sees that the ADJ group [9], [10] and Zilles and Liskov [16], [25], [26] used initial algebra semantics for their specifications, but that J. V. Guttag [11] probably thought in terms of final algebra semantics. (At least, Guttag and Horning [12] deny they are taking initial models for their specifications and come close to an informal description of the final model strategy. Moreover, in the first paper to explicitly formulate final algebra semantics [23], Wand argues that it is indeed the denotational semantics of Guttag's theory of specifications.) Mathematically exact declarations in favor of the far less well-understood final algebra semantics can be seen in Hornung and Raulefs [13], Kamin [14], Kapur and Srivas [15], the Munich Group [8], [24] and Wand [23].

The issues involved in the initial and final alternatives are many and complex; they seem to turn on independent theoretical and practical options for specific problems

* Received by the editors November 3, 1980, and in revised form April 26, 1982.

† Department of Computer Science, University of Leiden, Waasenaarseweg 80, 2300 RA Leiden, The Netherlands.

‡ Department of Computer Studies, University of Leeds, Leeds LS2 9JT, Britain. The results in this paper were obtained while this author was at the Mathematical Centre, Amsterdam, and an earlier edition of this paper is registered there as MC Report IW 142/80.

to do with data types. Unfortunately, no thoroughly researched comparative study of the questions involved is yet available. However, it is an objective of this paper to provide some theoretical perspective for such a discussion by reporting the technical facts of life about a rather basic problem, *the adequacy problem for specification methods*, which lead to the conclusion that the theory of algebraic data types needs both the initial and the final techniques. We will prove two theorems, the First and Second Characterization Theorems below, which characterize two kinds of effectively computable data types in terms of the initial and final algebra semantics for algebraic specifications *allowing finite sets of conditional equations only*. Before giving their statements we shall explain some background issues that have to do with data types which the theorems are meant to resolve; after this introduction we shall adopt an exclusively technical outlook.

Roughly speaking, a specification method M is characterized partly by syntactical properties of the specifications it uses and partly by the semantical conditions it imposes on their meanings. For example, a method M may allow specifications with equations only, or with conditional equations; it may require those sets of axioms to be finite or it may let them be recursively enumerable. Each of these four decisions yields two distinct methods depending on which of the initial and final algebra semantics is chosen. And the two ways of introducing hidden or auxiliary operators to assist in specifying data types doubles the number of methods based upon these familiar options. The adequacy problem for a particular specification method M is the informal question *Does the method M define all the data types one wants*? Our theorems will frame exact answers to two of three precise formulations of this question when M is assumed to use finite sets of conditional equations only and an elementary mechanism for involving hidden operators. The three versions of the adequacy question are determined by three natural and distinct kinds of effectively computable data type semantics:

Let us say that an algebra A is *effectively presented* whenever we possess an effective enumeration of its elements and we can effectively calculate its operations. Then A is said to be a *semicomputable algebra*, or a *cosemicomputable algebra*, if in addition the equality relation of A is r.e., or co-r.e., respectively. A is a *computable algebra* when equality is decidable.

Now it is obvious that an r.e. algebraic specification (Σ, E) defines a semicomputable algebra under its initial semantics: remembering the proof theoretical basis of the technique, with an r.e. set of axioms E one can simply enumerate all proofs and list the identifications $E \vdash t = t'$. It is less well known, though almost equally obvious, that the final algebra semantics of an r.e. algebraic specification defines a cosemicomputable algebra. Therefore, *if a data type can be specified both initially and finally by two r.e. sets of axioms then it must be computable.* Clearly, then, equational term rewriting systems, formal grammars, and so forth, with r.e. but not recursive word problems qualify as data types without any effectively definable final algebra specification. On the other hand in [6], we showed that the set of functions computed by LOOP-programs on the natural numbers—the primitive recursive functions—composed a data type with a finite equational specification (allowing hidden functions) under final algebra semantics, but that it does not possess an effective algebraic specification of any kind using initial algebra semantics. We will give some new examples to divide the methods in § 2.

Concentrating on the two methods based upon finite sets of conditional equations (and allowing hidden operators), the three formal adequacy problems per method

boil down to the question *Can the following known implications be reversed*?

FINITE CONDITIONAL SPECIFICATIONS + INITIAL SEMANTICS
→ SEMICOMPUTABLE DATA TYPES,

FINITE CONDITIONAL SPECIFICATIONS + FINAL SEMANTICS
→ COSEMICOMPUTABLE DATA TYPES,

BOTH SPECIFICATION METHODS
→ COMPUTABLE DATA TYPES.

In § 3, we prove that the second implication can be reversed. This argument will go quite some way toward reversing the first implication, at least far enough to prove that the third implication is an equivalence; we deal with these points in § 4. On top of the characterizations, we are able to put numerical upper bounds for the number of auxiliary operators and the number of equations necessary to specify the cosemicomputable and computable data types:

FIRST CHARACTERIZATION THEOREM. *Let A be an algebra finitely generated by elements named in its signature Σ. Then the following are equivalent*:

1. *A is cosemicomputable.*
2. *A possesses a conditional equation specification, involving at most 5 hidden functions and $15+|\Sigma|$ axioms, which defines A under its final algebra semantics.*

SECOND CHARACTERIZATION THEOREM. *Let A be an algebra finitely generated by elements named in its signature Σ. Then the following are equivalent*:

1. *A is computable.*
2. *A possesses two conditional equation specifications, each involving at most 5 hidden functions and $15+|\Sigma|$ axioms, such that one specification defines A under its initial algebra semantics while the other defines A under its final algebra semantics.*

This paper is the sixth in our series of mathematical studies of the power of definition and adequacy of algebraic specification methods for data type definition [2]–[6] (see also [7]). Obviously, the reader is assumed to be familiar with the informal issues and basic algebraic machinery of algebraic specifications and their semantics. For this material ADJ [10] is essential, but the reader ought also to be experienced in following algebraic arguments as he or she will then find this paper virtually self-contained: our previous work is involved explicitly in an appeal to [5] which dispenses with finite data types, and implicitly in that *we talk about single-sorted algebras only*. Our previous articles established a standard procedure for turning single-sorted adequacy theorems into their many-sorted generalizations, and that procedure readily applies here.

1. Specifications and their semantics. The purpose of this first section is to describe, in a summary form, two denotational semantics for algebraic data type specifications: *initial algebra semantics* and *final algebra semantics*. Our working definitions of these two mechanisms for assigning a meaning to a specification are given as Definitions 1.5 and 1.6 below: *they, and they alone, represent what we will have in mind for initial and final algebra semantics in the technical work which follows.* By way of exposition of these two different semantics we describe them from the standpoints of category theory, logic and lastly algebra. Let us repeat that we take it for granted that the reader is well versed in the mathematical theory of data types created by the ADJ group [10], [21], [22]:

Semantically, a data type is modelled by an algebra A finitely generated by elements named in its signature Σ, a so-called (finitely generated) *minimal algebra.* A specification (Σ, E) for a data type distinguishes the category ALG* (Σ, E) of all minimal algebras of signature Σ satisfying the axioms E and all morphisms between them. Thus, the semantics of a specification (Σ, E) is designed so as to pick out some algebra from ALG*(Σ, E) as the *unique meaning* $\mathcal{M}(\Sigma, E)$ where the uniqueness of $\mathcal{M}(\Sigma, E)$ is measured up to algebraic isomorphism. Given a data type semantics (modelled by an algebra) A, a specification (Σ, E) can be said to *correctly define* the data type when $\mathcal{M}(\Sigma, E) \cong A$.

Seen from the point of view of the category ALG* (Σ, E), *initial algebra semantics* for algebraic specifications assigns as the meaning of (Σ, E) the initial algebra $I(\Sigma, E)$ in ALG* (Σ, E); this $I(\Sigma, E)$ always exists and is unique up to isomorphism. On the other hand, *final algebra semantics* would like to pick out the final object from ALG* (Σ, E) as the meaning of (Σ, E), but clearly this final algebra is in all cases the *trivial one-point,* or *unit,* Σ*-algebra* $1 \in$ ALG* (Σ, E). (Notice 1 may not play an initial role in ALG* (Σ, E) because of the minimality assumption.) Instead, final algebra semantics turns to the category $\mathrm{ALG}_0^*(\Sigma, E)$ which is simply ALG* (Σ, E) with the unit algebra removed. Unfortunately, $\mathrm{ALG}_0^*(\Sigma, E)$ need not always possess a final object $F(\Sigma, E)$, but when it does this object is unique.

Because of this asymmetry, defining and using the final algebra semantics of algebraic specifications is a rather delicate matter when compared with the initial technique. Nevertheless, the technical motives behind final algebra semantics are natural enough and complement those behind initial algebra semantics. To explain these we adopt a logical point of view toward algebraic specifications from which the *raison d'être* of the semantics becomes evident.

Given any data type semantics A, a minimal algebra of finite signature Σ, consider the algebra $T(\Sigma)$ of all syntactic terms over Σ. There is an obvious semantic mapping $v_A: T(\Sigma) \to A$ which evaluates the formal expressions over Σ as data belonging to A; v_A is an epimorphism of Σ-algebras and is uniquely determined as a function by A. The congruence $\equiv_A$ induced on $T(\Sigma)$ by v_A, defined by

$$t \equiv_A t \text{ if and only if } v_A(t) = v_A(t') \text{ in } A, \tag{1}$$

for $t, t' \in T(\Sigma)$, is uniquely determined as a set by A and clearly

$$A \cong T(\Sigma)/\equiv_A. \tag{2}$$

Combinatorially, devising a specification (Σ, E) for A amounts to devising axioms E which determine this congruence $\equiv_A$ in some precise way.

The first, and most obvious, method is to choose axioms E such that $t, t' \in T(\Sigma)$ have the same meaning in A if, and only if, one can prove that $t = t'$ from the axioms E. In the standard notation of logic, the *desired relationship* between A and E is

$$A \models t = t' \text{ if and only if } E \vdash t = t', \tag{3}$$

or, equivalently,

$$t \equiv_A t' \text{ if and only if } E \vdash t = t'. \tag{4}$$

This is exactly the decision made when one seeks an algebraic specification (Σ, E) and uses initial algebra semantics to define A: the equivalence

$$I(\Sigma, E) \models t = t' \text{ if and only if } E \vdash t = t'$$

is always true and entails equivalence (4) when $I(\Sigma, E) \cong A$.

Final algebra semantics corresponds to a different use of the axioms in a specification (Σ, E). There one decides to assume $t, t' \in T(\Sigma)$ to have the same meaning in A if, and only if, one can assert $t = t'$ without contradicting the axioms of E:

$$t \equiv_A t' \text{ if and only if } t = t' \text{ is consistent with } E.$$

This notion of consistency simply means that there is some *nonunit* model $B \in \mathrm{ALG}^*(\Sigma, E)$ where $B \models t = t'$. Equivalently, the relationship desired between A and E can be expressed as follows: the congruence $\equiv_A$ has the property that *for every congruence* $\equiv$ *on* $T(\Sigma)$ *which defines a nonunit algebra* $T(\Sigma)/\equiv$ *in* $\mathrm{ALG}^*(\Sigma, E)$ *we have that*

$$t \equiv t' \text{ implies } t \equiv_A t'.$$

As will be seen, when this relationship between $\equiv_A$ and E can be arranged we have A as the final object of $\mathrm{ALG}_0^*(\Sigma, E)$. And it is precisely these technical observations to do with consistency which lie behind the notion of *semantic observability* much used in writings on final algebra semantics.

While thinking about these semantics for (Σ, E) in logical terms let us consider their effectivity. For initial algebra semantics,

$$t \equiv_A t' \text{ if and only if } E \vdash t = t'$$

and $\equiv_A$ is clearly recursively enumerable. For final algebra semantics however, $\equiv_A$ is co-r.e. This is proved as follows.

First, notice that

$$t \not\equiv_A t' \text{ if and only if } t = t' \text{ is inconsistent with } E.$$

Now, $t = t'$ is inconsistent with E if, and only if, the following condition holds: for any $t_1, t_2 \in T(\Sigma)$, $E \cup \{t = t'\} \vdash t_1 = t_2$. To list the elements of $\not\equiv_A$ two cases must be considered:

Case (i). There exist distinct $t_1, t_2 \in T(\Sigma)$ such that $t_1 \not\equiv_A t_2$. Choose such a pair of terms. The algorithm for $\not\equiv_A$ takes each $t, t' \in T(\Sigma)$ and lists all term identities provable from $E \cup \{t = t'\}$ seeking $t_1 = t_2$. If $t_1 = t_2$ appears in the list then $t \not\equiv_A t'$.

Case (ii). There does not exist $t_1, t_2 \in T(\Sigma)$ such that $t_1 \not\equiv_A t_2$. Here $\not\equiv_A$ is empty and therefore r.e.

Now we come to our purely algebraic definitions of these semantics framed in terms of congruences on $T(\Sigma)$.

Let A be an algebra of signature Σ.

A congruence $\equiv$ on A is said to be the *unit congruence* if for any $a, b \in A$ we have $a \equiv b$; or, equivalently, if $A/\equiv$ is the unit algebra of signature Σ.

A congruence $\equiv_2$ on A is said to *extend* another congruence $\equiv_1$ on A if for any $a, b \in A$ we have $a \equiv_1 b$ implies $a \equiv_2 b$.

Let E be a set of conditional equations over Σ.

If A satisfies the axioms of E we say that A is an *E-algebra.*

A congruence $\equiv$ on algebra A is said to be an *E-congruence* if for each conditional equation in variables $X = (X_1, \cdots, X_n)$

$$t_1(X) = t_1'(X) \wedge \cdots \wedge t_k(X) = t_k'(X) \rightarrow t(X) = t'(X)$$

and for any $a \in A^n$

$$\text{if } t_1(a) \equiv t_1'(a), \cdots, t_k(a) \equiv t_k'(a) \text{ in } A \text{ then } t(a) \equiv t'(a) \text{ in } A.$$

LEMMA 1.1. *Let* $\equiv$ *be a congruence relation on* $A \in \mathrm{ALG}(\Sigma, E)$. *Then* $\equiv$ *is an E-congruence if, and only if,* $A/\equiv$ *is an E-algebra.*

We will now define certain least and largest E-congruences on $T(\Sigma)$ whose corresponding factor algebras will be the initial and final objects of $\mathrm{ALG}_0^*(\Sigma, E)$ respectively. Let us consider the initial case first.

Define $\equiv_{\min(E)}$ to be the intersection of all E-congruences on $T(\Sigma)$ and set $T_I(\Sigma, E) = T(\Sigma)/\equiv_{\min(E)}$. It is easy to see that $\equiv_{\min(E)}$ is an E-congruence and to verify that

LEMMA 1.2. *$T_I(\Sigma, E)$ is isomorphic to any initial object $I(\Sigma, E)$ of* $\mathrm{ALG}^*(\Sigma, E)$.

Define $\equiv_{\max(E)}$ to be the smallest E-congruence extending all the nonunit E-congruences on $T(\Sigma)$. Equivalently, let $\equiv_{\max(E)}$ be the smallest E-congruence containing the union of all nonunit E-congruences on $T(\Sigma)$. And set $T_F(\Sigma, E) = T(\Sigma)/\equiv_{\max(E)}$.

Of course we have no guarantee that $\equiv_{\max(E)}$ is not the unit congruence and that $T_F(\Sigma, E)$ is not the unit algebra, but it is easy to prove that

LEMMA 1.3. *Whenever $\equiv_{\max(E)}$ is not the unit congruence, $T_F(\Sigma, E)$ is isomorphic to any final object $F(\Sigma, E)$ of* $\mathrm{ALG}_0^*(\Sigma, E)$.

OBSERVATION 1.4. *For $t, t' \in T(\Sigma)$, $t \not\equiv_{\max(E)} t'$ if, and only if, the least E-congruence extending $\equiv_{\min(E)} \cup \{t = t'\}$ is the unit congruence.*

We can now define precisely what we mean by initial and final algebra specifications for data types.

SEMANTICS OF ALGEBRAIC SPECIFICATIONS 1.5. Let E be a set of conditional equations over the signature Σ and let A be an algebra of signature Σ.

The pair (Σ, E) is said to be a *conditional equation specification* of the algebra A with respect to (1) *initial algebra semantics* or (2) *final algebra semantics* if (1) $T_I(\Sigma, E) \cong A$ or (2) $T_F(\Sigma, E) \cong A$.

When the set of axioms E is finite we speak of *finite conditional equation specifications* with respect to these semantics.

To conclude this preparatory section, we shall explain our favored method of involving hidden or auxiliary functions into algebraic specifications for data types.

Let A be an algebra of signature Σ_A and let Σ be a signature $\Sigma \subset \Sigma_A$. Then we mean by

$A|_\Sigma$ the Σ-algebra whose domain is that of A and whose constants and operators are those of A named in Σ: the Σ-reduct of A; and by

$\langle A \rangle_\Sigma$ the Σ-subalgebra of A generated by the constants and operators of A named in Σ: the smallest Σ-subalgebra of $A|_\Sigma$.

The following represents the two basic working definitions of specification theory in this paper.

ALGEBRAIC SPECIFICATIONS WITH HIDDEN OPERATORS 1.6. The specification (Σ, E) is said to be a *finite conditional equation hidden enrichment specification* of the algebra A with respect to (1) *initial algebra semantics* or (2) *final algebra semantics* if $\Sigma_A \subset \Sigma$, and E is a finite set of conditional equations over the (finite) signature Σ such that

$$T_I(\Sigma, E)|_{\Sigma_A} = \langle T_I(\Sigma, E) \rangle_{\Sigma_A} \cong A \tag{1}$$

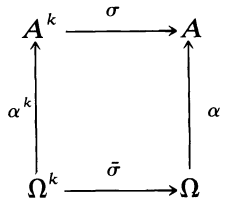
or

$$T_F(\Sigma, E)|_{\Sigma_A} = \langle T_F(\Sigma, E) \rangle_{\Sigma_A} \cong A. \tag{2}$$

In this paper, all specifications involving hidden operators are made to define data types as described in Definition 1.6.

A complex example of a specification of a data type using final algebra semantics is included in [6].

2. Effectively computable algebras. A countable algebraic system A is said to be *effectively presented* when it is given an effective coordinatization consisting of a recursive set $\Omega \subset \omega$ and a surjection $\alpha: \Omega \to A$, and, for each k-ary operation σ of A, a recursive *tracking function* $\bar{\sigma}$ which commutes the following diagram



wherein $\alpha^k(x_1, \cdots, x_k) = (\alpha x_1, \cdots, \alpha x_k)$.

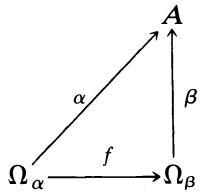
The algebra A is said to be *computable*, *semicomputable*, or *cosemicomputable*, if there exists an effective presentation $\alpha: \Omega \to A$ for which the relation $\equiv_\alpha$ on Ω defined by

$$n \equiv_\alpha m \text{ if and only if } \alpha n = \alpha m \text{ in } A$$

is recursive, r.e., or co-r.e., respectively.

These three notions are the standard formal definitions of constructive algebraic structures currently in use in mathematical logic and they derive from the work of M. O. Rabin [20] and, in particular, A. I. Mal'cev [18] devoted to founding a theory of computable algebraic systems. Their special feature is that they make computability into a *finiteness condition* of algebra: an isomorphism invariant possessed of all finite structures. In the case of finitely generated algebras, the concepts enjoy a much stronger recursion-theoretic invariance property which we shall now explain.

Let α and β be effective presentations of some algebra A. Then α *recursively reduces* to β (in symbols: $\alpha \leqslant \beta$) if there exists a recursive function f to commute the following diagram,



And α is *recursively equivalent* to β if both $\alpha \leqslant \beta$ and $\beta \leqslant \alpha$.

Recursive equivalence is the fundamental identity relation between numberings of algebras and is meant to measure the uniqueness of the recursion-theoretical concepts under their translation to algebraic systems.

Let R be a k-ary relation on A and let A be effectively presented by α. Then R is said to be α-computable if its preimage

$$\alpha^{-1}R = \{(x_1, \cdots, x_k) \in \Omega_\alpha^k : (\alpha x_1, \cdots, \alpha x_k) \in R\}$$

is recursive. The definitions of *α-semicomputable* and *α-cosemicomputable relations* follow mutato nomine. The following fact is easy to check.

LEMMA 2.1. *Let R be an α-computable (α-semicomputable or α-cosemicomputable) relation on A. If β is another effective presentation for A and β recursively reduces to α then R is β-computable (β-semicomputable or β-cosemicomputable). In particular,*

the effectivity of a relation on an algebra is unique up to the recursive equivalence of codifications.

The invariance property for finitely generated algebras which interests us is the existence of certain canonical effective presentations which solve the irritating problem of how to speak of a relation as being computable (say) without also having to name a coordinatization.

Henceforth, assume A is algebra finitely generated by elements named in its signature Σ.

Clearly, the term algebra $T(\Sigma)$ is computable under any natural Gödel numbering of terms. It is easy to make a general definition of such a Gödel numbering and to go on to prove that Gödel numberings compose an equivalence class under recursive equivalence; so the choice of $\gamma_*: \omega \to T(\Sigma)$ is unimportant. Let $v: T(\Sigma) \to A$ be the unique term evaluation homomorphism. We define the standard effective presentation of A derived from γ_* to be the composition

$$\gamma_A = v\gamma_* : \omega \to T(\Sigma) \to A.$$

To see that γ_A is indeed an effective coordinatization of A one need only observe that an effective presentation for A is nothing more than an epimorphism between A and a recursive algebra of natural numbers.

REDUCTION LEMMA 2.2. *The standard effective presentation γ_A of A recursively reduces to every effective presentation α of A.*

A proof of this can be found in Mal'cev [18]; coupled with Lemma 2.1, it has several important consequences.

INVARIANCE THEOREM 2.3. *The algebra A is computable, semicomputable or cosemicomputable if and only if it is so under the standard effective presentation γ_A.*

COROLLARY 2.4. *Any two semicomputable coordinatizations of A are recursively equivalent.*

Let R be a recursive number algebra and $\alpha: R \to A$ a homomorphism. Let us say that α is a *decidable*, *r.e.* or *co-r.e. homomorphism* accordingly as the congruence it induces on R

$$n \equiv_\alpha m \text{ if and only if } \alpha n = \alpha m \text{ in } A$$

is recursive, r.e. or co-r.e. respectively.

REPRESENTATION LEMMA 2.5. *If A is semicomputable, or cosemicomputable, then it can be represented as the image of a recursive number algebra R with domain ω under an r.e., or co-r.e., homomorphism α respectively. In particular, A is isomorphic to the factor algebra $R/\equiv_\alpha$ of R under the r.e., or co-r.e., congruence induced by α. If A is computable then it is isomorphic to a recursive number algebra R with domain ω provided A is infinite.*

What material we need from the theory of the recursive functions is elementary and is well covered by Machtey and Young [17] with one exception: Matijacevič's Diophantine Theorem.

Let $\mathbb{Z}[X_1, \cdots, X_n]$ denote the ring of polynomials in indeterminates $X_1, \cdots, X_n$ and with integer coefficients. A set $\Omega \subset \omega^n$ is said to be *diophantine* if there exists a polynomial $p \in \mathbb{Z}[X_1, \cdots, X_n, Y_1, \cdots, Y_m]$ such that

$$(x_1, \cdots, x_n) \in \Omega \Leftrightarrow \exists y_1, \cdots, y_m \in \omega . [p(x_1, \cdots, x_n, y_1, \cdots, y_m) = 0].$$

Clearly, each diophantine set is recursively enumerable; the converse is a hard theorem of Y. Matijacevič:

DIOPHANTINE THEOREM 2.6. *All recursively enumerable sets are diophantine.*

The number of search variables $y_1, \cdots, y_n$ can always be chosen to be 13 or less, incidentally. A good exposition of the theorem appears in Manin [19].

We will always use the Diophantine Theorem to obtain polynomials over the natural numbers ω rather than over $\mathbb{Z}$. We will now write down an equivalent characterization of a diophantine set of natural numbers, one more suited to our special tasks.

Let $\omega[X_1, \cdots, X_n]$ denote the set of all polynomials having natural number coefficients and involving only addition and multiplication.

A set $\Omega \subset \omega^n$ is *ω-diophantine* if there exist p and $q \in \omega[X_1, \cdots, X_n, Y_1, \cdots, Y_m]$ such that

$$(x_1, \cdots, x_n) \in \Omega \Leftrightarrow \exists y_1, \cdots, y_m \in \omega . [p(x_1, \cdots, x_n, y_1, \cdots, y_m) = q(x_1, \cdots, x_n, y_1, \cdots, y_m)].$$

It is easy to check that the ω-diophantine sets are precisely the diophantine sets.

These technical preliminaries concluded, we can now turn our attention to data types and their specifications.

BASIC LEMMA 2.7. *Let (Σ, E) be a specification with E a recursively enumerable set of conditional equations. Then $T_I(\Sigma, E)$ is semicomputable and $T_F(\Sigma, E)$ is cosemicomputable. In particular, if algebra A possesses an r.e. conditional equation hidden enrichment specification with respect to* (1) *initial algebra semantics or* (2) *final algebra semantics then* (1) *A is semicomputable or* (2) *A is cosemicomputable. If A possesses such specifications with respect to both initial and final algebra semantics then A is computable.*

The proof of Basic Lemma 2.7 is routine and is left as an exercise for the reader. (Hint: Use Observation 1.4 and recall the discussion of effectivity in § 1.) Here are examples of semicomputable and cosemicomputable algebras which are not computable.

COMBINATORY LOGIC 2.8. Consider the signature Σ consisting of constants K, S, I and a single binary operation $\cdot$. Combinatory logic can be axiomatized by three equations over this Σ.

$$(K \cdot X) \cdot Y = X,$$
$$((S \cdot X) \cdot Y) \cdot Z = (X \cdot Z) \cdot (Y \cdot Z),$$
$$I \cdot X = X,$$

where X, Y, Z are variables. The initial algebra of the resulting variety is known as the *term model for combinatory logic* and we denote it TMCL. Clearly, it is an algebra having a finite equational specification and it is semicomputable. It is not a computable algebra, however, because combinatory logic is a formalism strong enough to define all recursive functions; see Barendregt [1] for details.

POLYNOMIAL FUNCTIONS 2.9. The typical cosemicomputable algebra is a set of computable functions structured by some effective operators. For example, let A be a computable algebra of signature Σ and let $T_\Sigma[X_1, \cdots, X_n]$ be the algebra of formal polynomials in n indeterminates over Σ. Each $t \in T_\Sigma[X_1, \cdots, X_n]$ defines an n-argument *polynomial function* $A^n \to A$ which is computable. It is easy to derive an effective presentation of the Σ-algebra $PF(A^n, A)$ of all n-ary polynomial functions over A from a computable coordinatization of $T_\Sigma[X_1, \cdots, X_n]$; and to prove that $PF(A^n, A)$ is a cosemicomputable algebra. We give an A for which $PF(A^n, A)$ is not computable when $n \geq 13$.

Let A have domain ω, constants $0, 1 \in \omega$, and the operations of addition $x+y$, minus $x \dot{-} y$, multiplication $x \cdot y$, and

$$\min(x) = \begin{cases} 0 & \text{if } x = 0, \\ 1 & \text{if } x \geqq 1. \end{cases}$$

Let Σ be the signature of A.

Now let Ω be any r.e. subset of ω and assume it defined by the Diophantine Theorem as

$$n \in \Omega \Leftrightarrow \exists y \in \omega^n [p(n, y) = q(n, y)]$$

for $p, q \in \omega[X, Y_1, \cdots, Y_m]$. Define a family of polynomial maps $\omega^m \to \omega$ over A by

$$H_n(y) = \min(p_n(y) \dot{-} q_n(y) + q_n(y) \dot{-} p_n(y)).$$

Clearly, the family $\{H_n : n \in \omega\}$ is a computable subset of $PF(A^m, A)$ and if $PF(A^m, A)$ were a computable algebra then we could decide whether or not

$$H_n = 1$$

where $1(y) = 1$ for all $y \in \omega^m$. However, it is easy to see that

$$H_n = 1 \text{ if and only if } n \in \Omega.$$

Thus, choosing Ω to be r.e. and not recursive shows that $PF(A^m, A)$ cannot be computable. The reader might care to find a finite algebraic specification for $PF(A^m, A)$ as an exercise.

3. Cosemicomputable data types. This section is entirely given over to proving the First Characterization Theorem stated in the Introduction. In view of Basic Lemma 2.7, we have only to prove that (1) implies (2).

Let A be a cosemicomputable algebra of signature Σ.

First of all we dispense with the relatively easy case when A is finite. In [5], we proved that any finite algebra possesses a finite conditional equation specification under initial algebra semantics which involves at most 1 auxiliary operator, 1 simple equation and 2 conditional equations. As it happens, precisely the same syntactic specification designed there for a finite algebra A also defines A under its final algebra semantics. Thus, we leave the reader to check this claim (or to devise his or her own proof of the theorem in this special case) and we move on to the considerably more difficult case when A is cosemicomputable and infinite.

We divide the proof of this case into two parts. First, we will frame an auxiliary hypothesis H and prove the First Characterization Theorem for any infinite cosemicomputable algebra satisfying the extra condition H. This done, we will then prove that, indeed, every infinite cosemicomputable algebra satisfies our hypothesis H.

PARTITION HYPOTHESIS 3.1. Let A be effectively presented by $\alpha : \Omega_\alpha \to A$. By an *$\alpha$-computable partition* we mean a family $V = \{V_i : i \in \omega\}$ of nonempty subsets of A such that

(i) $V_i \cap V_j = \varnothing$ if $i \neq j$.

(ii) $\bigcup_{i \in \omega} V_i = A$.

(iii) The V_i are α-computable subsets of A uniformly in i; that is, the function $M_V : \omega \times \Omega_\alpha \to \{0, 1\}$ defined by

$$M_V(i, n) = \begin{cases} 0 & \text{if } \alpha n \in V_i, \\ 1 & \text{if } \alpha n \notin V_i \end{cases}$$

is recursive.

Thanks to Lemma 2.1 and the Reduction Lemma 2.2 we need not be careful about the coding α to which an α-computable partition is tied. And as our hypothesis H we may take the statement that *A possesses a computable partition.*

THE PROOF FOR A INFINITE, COSEMICOMPUTABLE, AND POSSESSING A COMPUTABLE PARTITION 3.2. Let A be the image of recursive number algebra R, with domain ω, under co-r.e. homomorphism $\alpha: R \to A$ (Representation Lemma 2.5) and assume A has a computable partition with respect to this α. In outline, our plan is to add to R a constant and some 5 functions to make a new recursive number algebra R_0 such that

(a) $R_0|_\Sigma = \langle R_0 \rangle_\Sigma = R$,

(b) $\equiv_\alpha$ is a congruence on R_0.

In consequence,

$$R_0/\equiv_\alpha|_\Sigma = \langle R_0/\equiv_\alpha \rangle_\Sigma = R/\equiv_\alpha \cong A.$$

For $R_0/\equiv_\alpha$ we will make a conditional equation specification (Σ_0, E_α) which defines it under final algebra semantics and which satisfies the required boundedness conditions. The first four new functions are designed to simulate arithmetic on R and, in particular, to respect the congruence $\equiv_\alpha$ on R. This latter condition will mean that the new functions will induce an arithmetic on $R_0/\equiv_\alpha$. With arithmetic internalized in this way, the fifth function will internalize an entirely new coding of R whose domain is the arithmetic on R. And, because the fifth function respects the congruence $\equiv_\alpha$, this coding may be internalized to $R_0/\equiv_\alpha$. This done we are able to systematically specify the coding, the recursive functions of R and the congruence $\equiv_\alpha$ itself, all by means of the Diophantine Theorem 2.6. So much for the informal description: first we build the arithmetic in R from the hypothesis of A having an α-computable partition.

Let $V = \{V_i : i \in \omega\}$ be an α-computable partition of A. Now V determines an α-computable equivalence relation $\equiv_V$ on A whose equivalence classes are the V_i. In particular, the factor set $A/\equiv_V$ is a computable set under coordinatization $\alpha(V): R \to A/\equiv_V$ defined by $\alpha(V)(n) = [\alpha n]$. This set supports a natural arithmetic based upon using V_0 as zero, and taking

$$\begin{aligned} &V_i \to V_{i+1} && \text{as successor,} \\ &(V_i, V_j) \to V_{i+j} && \text{as addition,} \\ &(V_i, V_j) \to V_{i \cdot j} && \text{as multiplication} \end{aligned}$$

and this *partition arithmetic* on $A/\equiv_V$ is what we propose to model in R by special tracking functions for $\equiv_{\alpha(V)}$.

The first four functions are determined by choosing a recursive transversal for $\equiv_{\alpha(V)}$ on R.

Let $t: \omega \to R$ be a recursive function enumerating the following "least code" transversal $T(V, R)$ for $\equiv_{\alpha(V)}$,

$$t(i) = (\mu z \in R)[\alpha z \in V_i].$$

Thus $T(V, R) = im(t)$ and, obviously, it is a recursive set such that $T(V, A) = \{\alpha n : n \in T(V, R)\}$ is an α-computable transversal for $\equiv_V$ on A.

We also define the recursive function $d: R \to \omega$ by

$$d(n) = (\mu z \in \omega)[\alpha n \in V_z]$$

which gives the location of n, and so of αn, in the equivalence classes of $\equiv_{\alpha(V)}$, and $\equiv_V$.

The new operators required on R to make R_0 are

Projection	$\text{proj}_R(x) = t(d(x))$,
Zero	$0_R = t(0)$,
Successor	$\text{succ}_R(x) = t(d(x)+1)$,
Addition	$\text{add}_R(x, y) = t(d(x)+d(y))$,
Multiplication	$\text{mult}_R(x, y) = t(d(x) \,.\, d(y))$.

The reader should pause to become familiar with the effects of these functions. Notice, for example, that by the guiding principles of their design, these operators make an algebra

$$(T(V, R); 0_R, \text{succ}_R, \text{add}_R, \text{mult}_R)$$

which is recursive and is isomorphic to the arithmetic we described on $A/\equiv_V$ under the mapping $\alpha(V): T(V, R) \to A/\equiv_V$. The role of proj_R is solely to *internalize* this *transversal arithmetic* within R. Notice, too, that what the partition property provides is this: because $\equiv_V$ is a coarser equivalence relation than equality in A, the relation $\equiv_{\alpha(V)}$ is coarser than $\equiv_\alpha$ on R with the result that each of the four new maps respect $\equiv_\alpha$ in a particularly strong way:

$$x \equiv_\alpha x' \text{ implies } \text{proj}_R(x) = \text{proj}_R(x'),$$

$$x \equiv_\alpha x' \text{ implies } \text{succ}_R(x) = \text{succ}_R(x')$$

and similarly for add_R and mult_R. Thus, $\equiv_\alpha$ remains a congruence on the algebra R when these functions are added.

Let $\Sigma_{\text{arith}} = \{0, \text{SUCC}, \text{ADD}, \text{MULT}\}$ denote the signature of the transversal arithmetic.

The fifth function required is there to code R by our transversal arithmetic. Choose any recursive bijection $\text{enum}_{V,R}: T(V, R) \to R$. This bijective renumbering of R we refer to as the *transversal coding*, but it should be thought of strictly in terms of the arithmetical structure of $T(V, R)$ and divorced from its original connections with α-codes. This can be made visible in our notations. Observe that the arithmetical structure of $T(V, R)$ entails we may write the set as a list without repetitions

$$T(V, R) = \{\text{succ}_R^n(0_R): n \in \omega\}$$

and, moreover, implicit in our view of the transversal coding is this composition

$$\omega \xrightarrow{\lambda n.\text{succ}_R^n(0_R)} T(V, R) \xrightarrow{\text{enum}_{V,R}} R$$

Still, the transversal coding must be internalized and this means it must be defined outside $T(V, R)$. Thus, we take as our last function in the construction of R_0 from R:

$$\text{enum}_R(n) = \text{enum}_{V,R}(\text{proj}_R(n)).$$

Again, we see that the partition yields

$$x \equiv_\alpha x' \text{ implies } \text{enum}_R(x) = \text{enum}_R(x')$$

and so we know that $\equiv_\alpha$ is a congruence on R_0. Thus, given (a) and (b) we concentrate on the problem of specifying $R_0/\equiv_\alpha$ by conditional equations (without hidden functions and using $15 + |\Sigma|$ formulae). This task we divide into the problems of specifying R_0

and then pressing on to specify $R_0/\equiv_\alpha$. As it turns out, the first job will be to give a specification (Σ_0, E_0), involving no hidden functions and 14 axioms, which defines R_0 by means of *initial algebra semantics*. Whence one more axiom e_α added to E_0 will make a specification (Σ_0, E_α) which completes the proof of the theorem in Case 3.2 (the reader curious about this arrangement is invited to read the proof of Lemma 3.7 first). Here is the specification (Σ_0, E_0) for R_0.

The first 10 equations specify the transversal arithmetic.

Projection

(1) $\quad \text{PROJ}(0) = 0,$

(2) $\quad \text{PROJ}(\text{SUCC}(X)) = \text{SUCC}(\text{PROJ}(X)),$

(3) $\quad \text{PROJ}(X) = \text{PROJ}(\text{PROJ}(X)),$

Successor

(4) $\quad \text{SUCC}(X) = \text{SUCC}(\text{PROJ}(X)),$

Addition

(5) $\quad \text{ADD}(X, 0) = \text{PROJ}(X),$

(6) $\quad \text{ADD}(X, \text{SUCC}(Y)) = \text{SUCC}(\text{ADD}(X, Y)),$

(7) $\quad \text{ADD}(X, Y) = \text{ADD}(\text{PROJ}(X), \text{PROJ}(Y))$

Multiplication

(8) $\quad \text{MULT}(X, 0) = 0,$

(9) $\quad \text{MULT}(X, \text{SUCC}(Y)) = \text{ADD}(\text{MULT}(X, Y), X),$

(10) $\quad \text{MULT}(X, Y) = \text{MULT}(\text{PROJ}(X), \text{PROJ}(Y)).$

Next we construct three formulae to specify the transversal coding of R. Consider these two sets designed to recover $T(V, R)$ from that coding. (We drop the subscript R from the operations of R_0.)

$$J_1 = \{(n, m) \in \omega \times \omega : \text{enum}(\text{succ}^n(0_R)) \in T(V, R) \,\&\, \text{enum}(\text{succ}^n(0_R)) = \text{succ}^m(0_R)\},$$

$$J_2 = \{(n, m) \in \omega \times \omega : \text{enum}(\text{succ}^n(0_R)) \notin T(V, R) \,\&\, \text{proj}(\text{enum}(\text{succ}^n(0_R))) = \text{enum}(\text{succ}^m(0_R))\}.$$

Our hypotheses imply that both sets are r.e. subsets of $\omega \times \omega$ and hence, by the Diophantine Theorem, there are polynomials p_1, q_1 and p_2, q_2, in $2 + k(1)$ and $2 + k(2)$ variables respectively, such that

$$(n, m) \in J_1 \Leftrightarrow \exists z \in \omega^{k(1)} . [p_1(n, m, z) = q_1(n, m, z)],$$

$$(n, m) \in J_2 \Leftrightarrow \exists z \in \omega^{k(2)} . [p_2(n, m, z) = q_2(n, m, z)].$$

Let P_1, Q_1 and P_2, Q_2 be formal polynomials over Σ_{arith} corresponding to p_1, q_1 and p_2, q_2 respectively. Then our enumeration axioms are

(11) $$P_1(X, Y, Z_1, \cdots, Z_{k(1)}) = Q_1(X, Y, Z_1, \cdots, Z_{k(1)}) \rightarrow \text{ENUM}(\text{PROJ}(X)) = \text{PROJ}(Y)$$

(12) $$P_2(X, Y, Z_1, \cdots, Z_{k(2)}) = Q_2(X, Y, Z_1, \cdots, Z_{k(2)}) \rightarrow \text{PROJ}(\text{ENUM}(\text{PROJ}(X))) = \text{ENUM}(\text{PROJ}(Y))$$

(13) $$\text{ENUM}(X) = \text{ENUM}(\text{PROJ}(X))$$

It now remains to add axioms to specify the new constant 0 and the original constants and operations of R. We need one formula in each case and this will make the total $|E_0| = 14 + |\Sigma|$.

Let $c \in R$ be a constant named by $\underline{c} \in \{0\} \cup \Sigma$. To this c there corresponds a unique $n \in \omega$ such that $c = \text{enum}(\text{succ}^n(0_R))$: assign the identification

$$\underline{c} = \text{ENUM}(\text{SUCC}^n(0)).$$

Let $f: R^k \to R$ be an operation named by $\underline{f} \in \Sigma$. Consider the graph of f translated to the transversal coding

$$G(f) = \{(n(1), \ldots, n(k), m): f(\text{enum}(\text{succ}^{n(1)}(0_R), \cdots, \text{enum}(\text{succ}^{n(k)}(0_R))) = \text{enum}(\text{succ}^m(0_R))\}.$$

Our hypotheses imply $G(f)$ is an r.e. set and again we define it by means of the Diophantine Theorem. Let p_f, q_f be polynomials in $k+1+k(f)$ variables such that

$$(n(1), \cdots, n(k), m) \in G(f) \Leftrightarrow \exists z \in \omega^{k(f)} . [p_f(n(1), \cdots, n(k), m, z) = q_f(n(1), \cdots, n(k), m, z)].$$

Choosing formal polynomials P_f, Q_f over Σ_{arith} corresponding to p_f, q_f we assign the axiom

$$P_f(X_1, \cdots, X_k, Y, Z_1, \cdots, Z_{k(f)}) = Q_f(X_1, \cdots, X_k, Y, Z_1, \cdots, Z_{k(f)}) \to f(\text{ENUM}(\text{PROJ}(X_1)), \cdots, \text{ENUM}(\text{PROJ}(X_k))) = \text{ENUM}(\text{PROJ}(Y))$$

LEMMA 3.3. *The specification* (Σ_0, E_0) *defines* R_0 *with respect to initial algebra semantics*:

$$T_I(\Sigma_0, E_0) \cong R_0.$$

Proof. First we picture R_0 through the transversal coding

$$R_0 = \{\text{enum}(\text{succ}^n(0)): n \in \omega\}.$$

Remembering that

$$\omega \xrightarrow{\lambda n.\text{succ}^n(0_R)} T(V, R) \xrightarrow{\text{enum}} R$$

is bijective, we define $\phi: R_0 \to T(\Sigma_0, E_0)$ by

$$\phi(\text{enum}(\text{succ}^n(0_R))) = [\text{ENUM}(\text{SUCC}^n(0))].$$

We write $\equiv_{E_0}$ as $\equiv$ and denote the equivalence class of $t \in T(\Sigma_0)$ under $\equiv$ by $[t]$. To show that ϕ is bijective is to prove that

$$T = \{\text{ENUM}(\text{SUCC}^n(0)): n \in \omega\}$$

is a transversal for $\equiv$. To show ϕ is a homomorphism will be an easy exercise afterwards.

Consider T as a transversal. It is easy to check that no distinct elements of T are equivalent under $\equiv$ because they denote different elements of R_0 and R_0 is an E_0-algebra. Thus, we have to prove that each $t \in T(\Sigma_0)$ is E_0-equivalent to some member of T and this is done by induction on term complexity.

The basis is obvious thanks to the identifications assigned to the constants of Σ_0.

Assume, as induction hypothesis, that all subterms of $t \in T(\Sigma_0)$ are E_0-equivalent to some element of T. We have to consider each situation corresponding to the leading

function symbol of t:

$$\text{PROJ, SUCC, ADD, MULT, ENUM, } f \in \Sigma$$

Case 1. $t = \text{PROJ}(s)$.

By the induction hypothesis $s = \text{ENUM}(\text{SUCC}^n(0))$.

Subcase 1.1. If $\text{enum}(\text{succ}^n(0_R)) \in T(V, R)$ then $\text{PROJ}(s) \equiv \text{ENUM}(\text{SUCC}^n(0))$.

Subcase 1.2. If $\text{enum}(\text{succ}^n(0_R)) \notin T(V, R)$ then $\text{PROJ}(s) \equiv \text{ENUM}(\text{SUCC}^m(0))$ for $(n, m) \in J_2$.

Proof of Subcase 1.1. This first subcase is quite involved as it introduces techniques and lemmata of use throughout the proof of Lemma 3.2; we shall write out its argument in detail. The bulk of the work lies in showing this important fact:

LEMMA 3.4. *Let* $\text{enum}(\text{succ}^n(0_R)) = \text{succ}^m(0_R)$. *Then* $\text{ENUM}(\text{SUCC}^n)(0)) \equiv \text{SUCC}^m(0)$.

Assume we have done this. Thus, immediately we know that for $(n, m) \in J_1$

$$\text{PROJ}(\text{ENUM}(\text{SUCC}^n(0))) \equiv \text{PROJ}(\text{SUCC}^m(0)).$$

A little lemma already required in the proof of Lemma 3.4 is this:

LEMMA 3.5. *For any* $k \in \omega$, $\text{PROJ}(\text{SUCC}^k(0)) \equiv \text{SUCC}^k(0)$.

Proof. This is an easy induction on k whose basis is covered by equation (1) and whose induction step is covered by equation (2).

Applying Lemma 3.5 we can deduce that

$$\text{PROJ}(\text{ENUM}(\text{SUCC}^n(0))) \equiv \text{SUCC}^m(0) \equiv \text{ENUM}(\text{SUCC}^n(0)),$$

the latter step using Lemma 3.4 again. This is what is required for Subcase 1.1.

Consider the proof of Lemma 3.4. We must use equation (11) which means we must verify the premiss that there exist $t_1, \cdots, t_{k(1)} \in T(\Sigma_0)$ for which

$$P_1(\text{SUCC}^n(0), \text{SUCC}^m(0), t_1, \cdots, t_{k(1)}) \equiv Q_2(\text{SUCC}^n(0), \text{SUCC}^m(0), t_1, \cdots, t_{k(1)}).$$

From this premiss we can conclude, directly, that

$$\text{ENUM}(\text{PROJ}(\text{SUCC}^n(0))) \equiv \text{PROJ}(\text{SUCC}^m(0)).$$

By Lemma 3.5, the Lemma 3.4 follows.

So consider the premiss. Since $(n, m) \in J_1$ we know there exists $z = (z(1), \cdots, z(k(1))) \in \omega^{k(1)}$ such that $p_1(n, m, z) = q_1(n, m, z)$. We claim the premiss is true on choosing $t_i = \text{SUCC}^{z(i)}(0)$, $1 \leqq i \leqq k(1)$. This follows from another invaluable general lemma:

LEMMA 3.6. *Let* $p(x_1, \cdots, x_k)$ *be any polynomial over* ω *and let* $P(X_1, \cdots, X_k)$ *be its formal translation to a polynomial over* Σ_{arith}. *Then for any* $n(1), \cdots, n(k) \in \omega$

$$P(\text{SUCC}^{n(1)}(0), \cdots, \text{SUCC}^{n(k)}(0)) \equiv \text{SUCC}^{p(n(1), \cdots, n(k))}(0).$$

Proof. This is done by a straightforward induction on the complexity of the polynomial $P(X_1, \cdots, X_k)$. The basis case, where $P(X_1, \cdots, X_k) = 0$ or $P(X_1, \cdots, X_k) = X_i$ for $1 \leqq i \leqq k$, is immediate. The induction step divides into three cases determined by the leading operator symbol of $P(X_1, \cdots, X_k)$. When this is SUCC the induction step is immediate. When it is ADD one requires an easy induction on m to prove that

$$\text{ADD}(\text{SUCC}^n(0), \text{SUCC}^m(0)) \equiv \text{SUCC}^{n+m}(0).$$

The basis of this induction will use (5) and Lemma 3.5; the induction step will use (6). When the leading operator symbol is MULT one has to prove

$$\text{MULT}(\text{SUCC}^n(0), \text{SUCC}^m(0)) \equiv \text{SUCC}^{n \cdot m}(0)$$

by induction on m. Here the basis is covered by equation (8); and the induction step is covered by equation (9) together with the previously completed case of addition.

Proof of Subcase 1.2. Given the pattern of reasoning in Subcase 1.1, this subcase can be completed quite concisely. Let $\text{proj}(\text{enum}(\text{succ}^n(0_R))) = \text{enum}(\text{succ}^m(0_R))$ so that $(n, m) \in J_2$. We shall prove that

$$\text{PROJ}(\text{ENUM}(\text{SUCC}^n(0))) \equiv \text{ENUM}(\text{SUCC}^m(0))$$

by using (12). Thanks to Lemma 3.5, it is enough to verify the premiss of (12) that there exist $t_1, \cdots, t_{k(2)} \in T(\Sigma_0)$ such that

$$P_2(\text{SUCC}^n(0), \text{SUCC}^m(0), t_1, \cdots, t_{k(2)}) \equiv Q_2(\text{SUCC}^n(0), \text{SUCC}^m(0), t_1, \cdots, t_{k(2)}).$$

Since $(n, m) \in J_2$, there exists $z = (z(1), \cdots, z(k(2))) \in \omega^{k(2)}$ such that $p_2(n, m, z) = q_2(n, m, z)$. Taking $t_i = \text{SUCC}^{z(i)}(0)$ and applying Lemma 3.6 the premiss is true.

This first case provides two evidently important identities: Lemma 3.4 and the statement of Subcase 1.2:

$(n, m) \in J_1$ if and only if $\text{ENUM}(\text{SUCC}^n(0)) \equiv \text{SUCC}^m(0)$,

$(n, m) \in J_2$ if and only if $\text{PROJ}(\text{ENUM}(\text{SUCC}^n(0)) \equiv \text{ENUM}(\text{SUCC}^m(0))$.

From these we can deduce for $\text{enum}(\text{succ}^n(0_R)) \notin T(V, R)$

$\text{PROJ}(\text{ENUM}(\text{SUCC}^n(0))) \equiv \text{SUCC}^m(0)$ if and only if $\exists z \,.\, [(n, z) \in J_2 \;\&\; (z, m) \in J_1]$

and taken together we have the means to access the algebraic specification's model of the transversal arithmetic. The next three cases $t = \text{SUCC}(s)$, $t = \text{ADD}(s_1, s_2)$ and $t = \text{MULT}(s_1, s_2)$ are routine to check.

Case 2. $t = \text{SUCC}(s)$.

By the induction hypothesis $s \equiv \text{ENUM}(\text{SUCC}^n(0))$.

Subcase 2.1. If $\text{enum}(\text{succ}^n(0_R)) \in T(V, R)$ then $\text{SUCC}(s) \equiv \text{ENUM}(\text{SUCC}^{n_i}(0))$ for $(n, z) \in J_1$ and $(z+1, m) \in J_1$.

Subcase 2.2. If $\text{enum}(\text{succ}^n(0_R)) \notin T(V, R)$ then $\text{SUCC}(s) \equiv \text{ENUM}(\text{SUCC}^m(0))$ for $(n, z) \in J_2$ and (z, w), $(w+1, m) \in J_1$.

Consider $\text{enum}(\text{succ}^n(0_R)) \in T(V, R)$. Then Lemma 3.4 says that

$$\text{SUCC}(\text{ENUM}(\text{SUCC}^n(0))) \equiv \text{SUCC}(\text{SUCC}^z(0)) \equiv \text{SUCC}^{z+1}(0) \quad \text{for } (n, z) \in J_1.$$

To make a reduction to an element of T, we have only to prefix an ENUM to the right-hand side by applying Lemma 3.4 again: $\text{SUCC}^{z+1}(0) \equiv \text{ENUM}(\text{SUCC}^z(0))$ for $(z+1, m) \in J_1$.

Consider $\text{enum}(\text{succ}^n(0_R)) \notin T(V, R)$. Then equation (4) and Subcase 1.2 allows us to write

$$\text{SUCC}(\text{ENUM}(\text{SUCC}^n(0))) \equiv \text{SUCC}(\text{PROJ}(\text{ENUM}(\text{SUCC}^n(0)))) \equiv \text{SUCC}(\text{ENUM}(\text{SUCC}^z(0))) \quad \text{for } (n, z) \in J_2.$$

But $\text{enum}(\text{succ}^z(0_R)) \in T(V, R)$ so the right-hand side is covered by Subcase 2.1. Thus

$\text{SUCC}(\text{ENUM}(\text{SUCC}^z(0))) \equiv \text{ENUM}(\text{SUCC}^m(0))$ for $(z, w) \in J_1$ and $(w+1, m) \in J_1$.

The two other arithmetical cases follow the same pattern: equations (7) and (10) guarantee that the identities of Lemma 3.4 and Subcase 1.2 can reduce the subterms to numerals. Lemma 3.4 gives the numeral which is E_0-equivalent to t. And the prefixing of an ENUM, to complete the reduction of t to an element of T, is again done by Lemma 3.4. We omit the details, leaving them as a straightforward, if tedious, exercise for the reader.

Case 5. $t = \text{ENUM}(s)$.

By the induction step $s \equiv \text{ENUM}(\text{SUCC}^n(0))$.

Subcase 5.1. If $\text{enum}(\text{succ}(0_R)) \in T(V, R)$ then $\text{ENUM}(s) \equiv \text{ENUM}(\text{SUCC}^m(0))$ for $(n, m) \in J_1$.

Subcase 5.2. If $\text{enum}(\text{succ}(0_R)) \notin T(V, R)$ then $\text{ENUM}(s) \equiv \text{ENUM}(\text{SUCC}^m(0))$ for $(n, z) \in J_2$ and $(z, m) \in J_1$.

Subcase 5.1 is immediate from Lemma 3.4 which says that $\text{ENUM}(\text{SUCC}^n(0)) \equiv \text{SUCC}^m(0)$ for $(n, m) \in J_1$.

In Subcase 5.2, we may use equation (12) and Subcase 1.2 to write

$$\text{ENUM}(\text{ENUM}(\text{SUCC}^n(0))) \equiv \text{ENUM}(\text{PROJ}(\text{ENUM}(\text{SUCC}^n(0))))$$
$$\equiv \text{ENUM}(\text{ENUM}(\text{SUCC}^z(0))) \quad \text{for } (n, z) \in J_2.$$

But $\text{enum}(\text{succ}^z(0_R)) \in T(V, R)$ so we are in the situation of Subcase 5.1 again.

Case 6. $t = f(s_1, \cdots, s_k)$.

By the induction hypothesis $s_i \equiv \text{ENUM}(\text{SUCC}^{n(i)}(0))$, $1 \leqq i \leqq k$. We claim that

$$f(s_1, \cdots, s_k) \equiv \text{ENUM}(\text{SUCC}^m(0)) \quad \text{for } (n(1), \cdots, n(k), m) \in G(f).$$

Now, given $n = (n(1), \cdots, n(k)) \in \omega^k$ and m with $(n, m) \in G(f)$, we can choose $z = (z(1), \cdots, z(k(f)))$ such that $p_f(n, m, z) = q_f(n, m, z)$. Substituting $\text{SUCC}^{n(i)}(0)$, $\text{SUCC}^m(0)$ and $\text{SUCC}^{z(i)}(0)$ into the premiss of equation (13) we can (via Lemma 3.6) detach the identity

$$f(\text{ENUM}(\text{PROJ}(\text{SUCC}^{n(1)}(0))), \cdots, \text{ENUM}(\text{PROJ}(\text{SUCC}^{n(k)}(0)))))$$
$$\equiv \text{ENUM}(\text{PROJ}(\text{SUCC}^m(0))).$$

By Lemma 3.5 this reduces to our claim.

To complete the proof of Lemma 3.3 we have now to verify that $\phi : R_0 \to T(\Sigma_0, E_0)$ is a homomorphism. Each constant and each operation of R_0 must be considered, but we will write out only one case which is entirely typical. We will now show that for any $x \in R_0$,

$$\phi(\text{enum}(x)) = \text{ENUM}(\phi(x)).$$

Write $x = \text{enum}(\text{succ}^n(0_R))$.

If $\text{enum}(\text{succ}^n(0_R)) \in T(V, R)$ then

$$\phi(\text{enum}(\text{enum}(\text{succ}^n(0_R)))) = \phi(\text{enum}(\text{succ}^m(0_R))) \text{ for } (n, m) \in J_1$$
$$= [\text{ENUM}(\text{SUCC}^m(0))] \text{ by definition of } \phi$$
$$= [\text{ENUM}(\text{ENUM}(\text{SUCC}^n(0)))] \text{ by Subcase 5.1}$$
$$= \text{ENUM}[\text{ENUM}(\text{SUCC}^n(0))] \text{ by definition of ENUM}$$
$$= \text{ENUM}[\phi(\text{enum}(\text{succ}^n(0_R)))].$$

If enum $(\text{succ}^n\ (0_R)) \notin T(V, R)$ then

$$\begin{aligned}\phi(\text{enum }(\text{enum }(\text{succ}^n\ (0_R)))) &= \phi(\text{enum }(\text{proj }(\text{enum }(\text{succ}^n\ (0_R))))) \\ &= \phi\ (\text{enum }(\text{enum }(\text{succ}^z\ (0_R)))) \text{ for } (n, z) \in J_2 \\ &= \phi\ (\text{enum }(\text{succ}^m\ (0_R))) \text{ for } (z, m) \in J_1 \\ &= [\text{ENUM }(\text{SUCC}^m\ (0))] \text{ by definition of } \phi \\ &= [\text{ENUM }(\text{ENUM }(\text{SUCC}^n\ (0)))] \text{ by Subcase 5.2} \\ &= \text{ENUM }[\text{ENUM }(\text{SUCC}^n\ (0))] \text{ by definition of ENUM} \\ &= \text{ENUM }(\phi\ (\text{enum }(\text{succ}^n\ (0_R)))).\end{aligned}$$

This concludes the proof of Lemma 3.3.

Finally, we shall make one new axiom e_α which when added to E_0 forms $E_\alpha = E_0 \cup \{e_\alpha\}$ and completes a final algebra specification for $R_0/\equiv_\alpha$. Translating $\equiv_\alpha$ into the transversal coding we get

$$J_\alpha = \{(n, m) \in \omega \times \omega : \text{enum }(\text{succ}^n\ (0_R)) \not\equiv_\alpha \text{enum }(\text{succ}^m\ (0_R))\}.$$

By our hypothesis, this is an r.e. set and so we can define it, via the Diophantine Theorem, as

$$\{(n, m) \in \omega \times \omega : \exists z \in \omega^{k(\alpha)} . [p_\alpha(n, m, z) = q_\alpha(n, m, z)]\}.$$

Taking P_α, Q_α as formal translations of p_α, q_α we set e_α to be the formula

$$\begin{aligned}P_\alpha(X, Y, Z_1, \cdots, Z_{k(\alpha)}) &= Q_\alpha(X, Y, Z_1, \cdots, Z_{k(\alpha)}) \\ &\wedge \text{ENUM }(\text{PROJ }(X)) = \text{ENUM }(\text{PROJ }(Y)) \to U = V.\end{aligned}$$

LEMMA 3.7. *The specification* (Σ_0, E_α) *defines* $R_0/\equiv_\alpha$ *with respect to final algebra semantics*:

$$T_F(\Sigma, E_\alpha) \cong R_0/\equiv_\alpha.$$

Proof. We prove the representation as follows. Let ψ be the unique semantic evaluation epimorphism $T(\Sigma_0) \to R_0/\equiv_\alpha$ so that $T(\Sigma_0)/\equiv_\psi$ is isomorphic to $R_0/\equiv_\alpha$. We will show that $\equiv_\psi$ is a maximal E_α-congruence on $T(\Sigma_0)$ whence it will follow that $\equiv_\psi$ is $\equiv_{\max(E_\alpha)}$ and

$$T(\Sigma_0)/\equiv_{\max(E_\alpha)} = T_F(\Sigma_0, E_\alpha) \cong R_0/\equiv_\alpha.$$

It is a routine matter to check that $\equiv_\psi$ is nonunit and is, itself, an E_α-congruence. Consider its maximality. We have to show that if $\equiv$ is any nonunit E_α-congruence then $\equiv$ is a subcongruence of $\equiv_\psi$. Contrapositively, we shall argue that if $\equiv$ is an E_α-congruence which is not a subcongruence of $\equiv_\psi$ then $\equiv$ is the unit congruence.

This is done by finding terms $t, t' \in T(\Sigma_0)$ such that

(i) there exists $s = (s_1, \cdots, s_{k(\alpha)}) \in T(\Sigma_0)^{k(\alpha)}$ for which $P_\alpha(t, t', s) \equiv Q_\alpha(t, t', s)$;

(ii) $\text{ENUM}(t) \equiv \text{ENUM}(t')$

because then we may apply conditional equation e_α to deduce $\equiv$ is unit. We have to get these terms from R_0, of course.

Using the assumption that $\equiv$ is an E_α-congruence and the initiality of R_0 for E_0-algebras (Lemma 3.3), we can define an epimorphism $\phi: R_0 \to T(\Sigma_0)/\equiv$ which translates $\equiv$ into the numerical congruence $\equiv_\phi$ since $R_0/\equiv_\phi$ is isomorphic with $T(\Sigma_0)/\equiv$. It is easy to see that our hypothesis $\equiv \not\subset \equiv_\psi$ means that $\equiv_\phi \not\subset \equiv_\alpha$ in R_0.

Thus, we choose enum $(\text{succ}^n(0_R))$, enum $(\text{succ}^m(0_R)) \in R_0$ such that

$$\text{enum}(\text{succ}^n(0_R)) \equiv_\phi \text{enum}(\text{succ}^m(0_R))$$

$$\text{but enum}(\text{succ}^n(0_R)) \not\equiv_\alpha \text{enum}(\text{succ}^m(0_R)).$$

By the diophantine definition of $\not\equiv_\alpha$ there exist $z = (z(1), \cdots, z(k(\alpha))) \in \omega^{k(\alpha)}$ for which $p_\alpha(n, m, z) = q_\alpha(n, m, z)$. We set $t = \text{SUCC}^n(0)$, $t' = \text{SUCC}^m(0)$ and $s_i = \text{SUCC}^{z(i)}(0)$ for $1 \leqq i \leqq k(\alpha)$. Now condition (i) follows from Lemma 3.6, and condition (ii) from our choice of n, m and, in both cases, the initiality of R_0.

This concludes the proof of Case 3.2.

PROPOSITION 3.8. *Every infinite cosemicomputable algebra possesses a computable partition.*

Proof. Thanks to the Representation Lemma 2.5, this proposition follows from this next statement whose proof is an exercise in recursive function theory.

LEMMA 3.9. *Let $\equiv$ be a co-r.e. equivalence relation on ω having infinitely many equivalence classes. Then there is a family $V = \{V_i : i \in \omega\}$ of nonempty disjoint subsets of ω such that*

(1) $U_{i \in \omega} V_i = \omega$,

(2) *$n \in V_i$ is recursive uniformly in i,*

(3) *if $n \equiv m$ and $m \in V_i$ then $n \in V_i$.*

Proof. We will describe an effective procedure which constructs the family V in stages. These stages we index by natural numbers. At each even stage $s = 2n$ we will have started the building of $V_0, \cdots, V_{n-1}$, but no other members of V. Our task at this stage will be to give V_n its first element. At each odd stage $s = 2n+1$ we will ensure that n, itself, belongs to one of $V_0, \cdots, V_{n-1}$. Thus at the beginning of each stage s we will have made only finite parts of $V_0, \cdots, V_{n-1}$ and nothing else. Let V_i^s denote the status of V_i at the beginning of stage s.

Even from this outline it is clear that conditions (1) and (2) will hold for V. By construction,

$$n \in V_i \Leftrightarrow i \leqq 2n \;\&\; n \in V_i^{2n}$$

and we will know that every n is assigned sooner or later at an odd stage. Condition (3) will be routine to check after we have described the procedure. We formalize an enumeration of $\not\equiv$ by

$$n \not\equiv m \text{ if and only if } \exists k \,.\, R(k, n, m)$$

for some recursive predicate R.

Stage $s = 2n$. Now $V_0^{s-1}, \cdots, V_{n-1}^{s-1}$ are nonempty, but $V_n^{s-1} = \varnothing$. We want to name the first element of V_n. We enumerate the finite set $V^{s-1} = V_0^{s-1} \cup \cdots \cup V_{n-1}^{s-1}$ searching for some $z \in \omega$ such that for all $m \in V^{s-1}$, $z \not\equiv m$. Such an element z will exist because $\omega/\equiv$ is infinite. This z is put into V_n with the result that at the conclusion of this stage

$$V_i^s = V_i^{s-1} \text{ for } 0 \leqq i \leqq n-1 \text{ and } V_n^s = \{z\}.$$

Stage $s = 2n+1$. Again $V_0^{s-1}, \cdots, V_{n-1}^{s-1}$ are nonempty but we are concerned only with the number n. First, we recursively decide whether $n \in V^{s-1} = V_0^{s-1} \cup \cdots \cup V_{n-1}^{s-1}$. If this is so we are done and at the conclusion of this test $V_i^s = V_i^{s-1}$ for $0 \leqq i \leqq n-1$.

Assume $n \in V^{s-1}$. Now we will put this n in some V_i, $1 \leqq i \leqq n-1$. By searching sufficiently far out in the enumeration of $\not\equiv$ it is possible to find some k_0 and an

$1 \leqq i \leqq n-1$ such that for every $j \neq i$, and $0 \leqq j \leqq n-1$, and for every $m \in V_j^{s-1}$ there is a $k < k_0$ for which $R(k, m, n)$ is true. That is we will come across a V_i^{s-1} for which we can verify that $n \neq m$ for $m \in V^s - V_i^{s-1}$. We put $n \in V_i^{s-1}$. Thus, at the end of this case of stage $s = 2n+1$,

$$V_j^s = V_j^{s-1} \text{ for } j \neq i \text{ and } 1 \leqq j \leqq n-1, \text{ and } V_i^s = V_i^{s-1} \cup \{n\}.$$

This construction proves Lemma 3.9 and so concludes the proofs of Proposition 3.8, and of our main theorem.

4. Semicomputable data types. Our characterization theorem for cosemicomputable data types focuses attention on a question we noticed and left open in the first paper of our series [2] (see also [7]). We shall reformulate it now as an opinion:

CONJECTURE 4.1. *Let A be an algebra finitely generated by elements named in its signature Σ. Then there exist $N \in \omega$ and $M = M(|\Sigma|) \in \omega$, numbers independent of A, such that the following are equivalent:*

1. *A is semicomputable.*

2. *A possesses a conditional equation specification, involving at most N hidden functions and M conditional equations, which defines A as a hidden enrichment under its initial algebra semantics.*

Moreover, we expect that $N \leqq 6$ and $M \leqq 20 + |\Sigma|$.

Since (2) implies (1) by Basic Lemma 2.7, the conjecture is the statement that (1) implies (2). Actually, we did not ask for bounds in [2], but we do so here although the unbounded adequacy problem remains open. Until the conjecture is settled, the precise numerical values of the bounds are of secondary importance, of course.

The theoretical importance of a confirmation of the conjecture is evident. First, semicomputable data types abound and one simply wants an adequacy theorem for them (one sharper than the result we proved in [2], certainly). And, secondly, if Conjecture 4.1 could be turned into a theorem then it would completely resolve the debate between the advocates of initial and final algebra semantics for specifications, at least for *theoria* if not for *praxis*. It seems hard to imagine a more elegant state of affairs than that depicted in the Venn diagram of Figure 4.1.

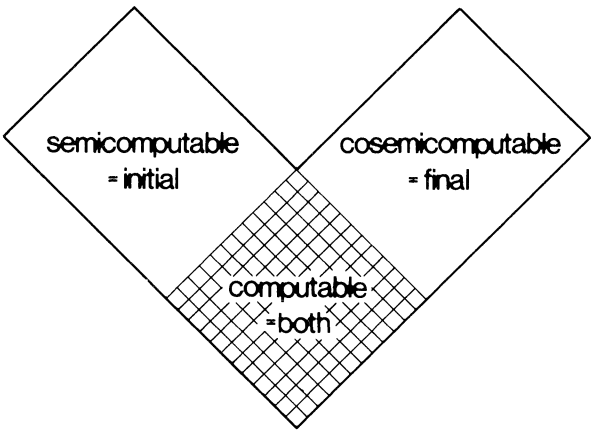


FIG. 4.1

We will conclude this paper by explaining the extent to which its methods fail to establish our conjecture.

Assuming A to be semicomputable, we can first of all dispense with the finite case because we proved the existence of a bounded conditional equational specification for it in [5]. (One hidden function, one identification and two conditional equations are sufficient for any finite data type!) Now, if A is infinite then it turns out that a small adaptation to the proof of Proposition 3.2 will settle Conjecture 4.1 under the hypothesis that A has a computable partition. Let us explain this.

The first change in the proof of Proposition 3.2 is made at the relatively late stage of the construction of the last axiom e_α from a diophantine definition of the r.e. set J_α. As A is semicomputable we want to consider the complement of J_α instead: since

$$\neg J_\alpha = \{(n, m) \in \omega \times \omega : \text{enum}\,(\text{succ}^n\,(0_R)) \equiv_\alpha \text{enum}\,(\text{succ}^m\,(0_R))\}$$

is r.e. we can define it, via the Diophantine Theorem, as

$$\{(n, m) \in \omega \times \omega : \exists z \in \omega^{k(\alpha)} \,.\, [p_\alpha(n, m, z) = q_\alpha(n, m, z)]\}$$

for (new) polynomials p_α, q_α. Taking P_α, Q_α as formal versions of p_α, q_α we take, as the new e_α, the axiom

$$P_\alpha(X, Y, Z_1, \cdots, Z_{k(\alpha)}) = Q_\alpha(X, Y, Z_1, \cdots, Z_{k(\alpha)}) \rightarrow \text{ENUM}\,(\text{PROJ}\,(X)) = \text{ENUM}\,(\text{PROJ}\,(Y)).$$

The redefined specification (Σ_0, E_α) specifies $R/\equiv_\alpha$ under its initial algebra semantics: a fact which can be readily verified and is much easier than Lemma 3.7. Thus, we know this next fact which improves our earlier bounded adequacy theorem for computable data types in [4], and obtains for us the Second Characterization Theorem stated in the Introduction.

THEOREM 4.2. *Let A be an infinite semicomputable algebra, finitely generated by elements named in its signature. If A has a computable partition then A possesses a conditional equation specification, involving 5 hidden functions and $15 + |\Sigma|$ conditional equations, which defines A as a hidden enrichment under its initial algebra semantics.*

Unfortunately our strategy for the semicomputable case breaks down at the last minute:

THEOREM 4.3. *There exists a finitely generated semicomputable algebra (having an initial algebra specification without hidden functions and with only 3 equations!) which does not possess a computable partition.*

The algebra in question is that in Example 2.8 and Theorem 4.3 is merely a rephrasing of Scott's theorem about the term model of combinatory logic: Scott has shown that one cannot even computably partition TMCL into *two* sets, see Barendregt [1], Theorem 2.21.

REFERENCES

[1] H. P. BARENDREGT, *The type free lambda calculus*, in Handbook of Mathematical Logic, J. K. Barwise, ed., North-Holland, Amsterdam, 1977, pp. 1091–1132.

[2] J. A. BERGSTRA AND J. V. TUCKER, *Algebraic specifications of computable and semicomputable data structures*, Department of Computer Science Research Report IW 115, Mathematical Centre, Amsterdam, 1979.

[3] ———, *A characterisation of computable data types by means of a finite, equational specification method*, in Automata, Languages and Programming, Seventh Colloquium, Noordwijkerhout, 1980, J. W. de Bakker and J. van Leeuwen, eds., Springer-Verlag, Berlin, 1980, pp. 76–90.

[4] ———, *Equational specifications for computable data types: six hidden functions suffice and other sufficiency bounds*, Dept. Computer Science Research Report IW 128, Mathematical Centre, Amsterdam, 1980.

[5] J. A. BERGSTRA AND J. V. TUCKER, *On bounds for the specification of finite data types by means of equations and conditional equations*, Dept. Computer Science Research Report IW 131, Mathematical Centre, Amsterdam, 1980.

[6] ———, *A natural data type with a finite equational final semantics specification but no effective equational initial semantics specification*, Bull. EATCS, 11 (1980), pp. 23–33.

[7] ———, *On the adequacy of finite equational methods for data type specification*, ACM-SIGPLAN Notices, 14 (11) (1979), pp. 13–18.

[8] M. BROY, W. DOSCH, H. PARSCH, P. PEPPER AND M. WIRSING, *Existential quantifiers in abstract data types*, in Automata Languages and Programming, Sixth Colloquium, Graz, 1980, H. Maurer, ed., Springer-Verlag, Berlin, 1979, pp. 72–87.

[9] J. A. GOGUEN, J. W. THATCHER, E. G. WAGNER AND J. B. WRIGHT, *Abstract data types as initial algebras and correctness of data representations*, in Proc. ACM Conference on Computer Graphics, Pattern Recognition and Data Structure, ACM, New York, 1975, pp. 89–93.

[10] J. A. GOGUEN, J. W. THATCHER AND E. G. WAGNER, *An initial algebra approach to the specification, correctness and implementation of abstract data types*, Current Trends in Programming Methodology IV, Data Structuring, R. T. Yeh, ed., Prentice-Hall, Englewood Cliffs, NJ, 1978, pp. 80–149.

[11] J. V. GUTTAG, *The specification and application to programming of abstract data types, Ph.D. thesis, Dept. Computer Science, University of Toronto, Toronto*, 1975.

[12] J. V. GUTTAG AND J. J. HORNING, *The algebraic specification of abstract data types*, Acta Informatica, 10 (1978), pp. 27–52.

[13] G. HORNUNG AND P. RAULEFS, *Terminal algebra semantics and retractions for abstract data types*, in Automata, Languages and Programming, Seventh Colloquium, Noordwijkerhout 1980, J. W. de Bakker and J. van Leeuwen, eds., Springer-Verlag, Berlin, 1980, pp. 310–325.

[14] S. KAMIN, *Final data type specifications: a new data type specification method*, in Seventh ACM Principles of Programming Languages Conference, Las Vegas, ACM, 1980, pp. 131–138.

[15] D. KAPUR AND M. K. SRIVAS, *Expressiveness of the operation set of a data abstraction*, in Seventh ACM Principles of Programming Languages Conference, Las Vegas, ACM, 1980.

[16] B. LISKOV AND S. ZILLES, *Specification techniques for data abstractions*, IEEE Trans. Software Engng., 1 (1975), pp. 7–19.

[17] M. MACHTEY AND P. YOUNG, *A Introduction to the General Theory of Algorithms*, North-Holland, New York, 1978.

[18] A. I. MAL'CEV, *Constructive algebras*, I., Russian Mathematical Surveys, 16 (1961), pp. 77–129.

[19] Y. MANIN, *A Course in Mathematical Logic*, Springer-Verlag, New York, 1977.

[20] M. O. RABIN, *Computable algebra, general theory and the theory of computable fields*, Trans. Amer. Math. Soc., 95 (1960), pp. 341–360.

[21] J. W. THATCHER, E. G. WAGNER AND J. B. WRIGHT, *Specification of abstract data types using conditional axioms*, IBM Research Report RC 6214, Yorktown Heights, NY, 1979.

[22] ———, *Data type specification: parametrization and the power of specification techniques*, IBM Research Report RC 7757, Yorktown Heights, NY, 1979.

[23] M. WAND, *Final algebra semantics and data type extensions*, J. Comput. Systems Sci., 19 (1979), pp. 27–44.

[24] M. WIRSING AND M. BROY, *Abstract data types as lattices of finitely generated models*, Mathematical Foundations of Computer Science, Eighth Symposium, Rydzyna 1980, Springer-Verlag, Berlin, 1980.

[25] S. ZILLES, *Algebraic specification of data types*, Project MAC Progress Report 11, M.I.T., Cambridge, MA, 1974.

[26] ———, *An introduction to data algebras*, working paper, IBM Research Laboratory, San Jose, CA, 1975.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0013 $01.25/0

SOME TIME–SPACE TRADEOFF RESULTS CONCERNING SINGLE-TAPE AND OFFLINE TM's*

OSCAR H. IBARRA† AND SHLOMO MORAN‡

Abstract. Fast simulations of time-bounded single-tape TM's and offline TM's (i.e., TM's with a two-way read-only input and one storage tape) by space-bounded TM's of the same type are presented. The following results are shown:

(1) Any language accepted by a single-tape TM in time $T(n) \geqq n^2$ can be accepted by a single-tape TM in space $T^{1/2}(n)$ and time $T^2(n)$.

(2) Any language accepted by an offline TM in time $T(n) \geqq n$ can be accepted by an offline TM in space $(T(n) \log n)^{1/2}$ and time $T^{3/2}(n)(T^{1/2}(n) + n/(\log n)^{1/2})$.

Similar (in fact, in some sense, stronger) results hold for nondeterministic TM's. For example:

(3) Any language accepted by a single-tape nondeterministic TM in time $T(n) \geqq n^2$ can be accepted by a single-tape nondeterministic TM in space $S(n)$ and time $T^2(n)/S(n)$ for any $T^{1/2}(n) \leqq S(n) \leqq T(n)$.

Similar time–space tradeoffs hold for TM's with a multidimensional storage tape. Previously known results on simulation of time bounded by space bounded TM's had exponential (in $T(n)$) time complexity.

Key words. single-tape TM, offline TM, time-bounded, space-bounded, time–space tradeoff

1. Introduction.

A problem of long standing in complexity theory is finding a characterization of time bounded Turing machines (TM's) in terms of space bounded TM's.[1] The "exact" relationship between time and space is not known. It is not even known whether or not polynomial space bounded multitape TM's are strictly more powerful than polynomial time bounded TM's. Currently the best simulation result known for multitape TM's is the following [1]:

(a) Any language accepted by a $T(n)$ time bounded multitape TM can be accepted by a $T(n)/\log T(n)$ space bounded multitape TM.

When the model of computation is a single-tape TM or an offline TM, sharper results are known [3], [6]. (A single-tape TM has a single two-way read-write tape which initially contains the input string. The tape is infinite to the right only. An offline TM has a two-way read-only input tape with endmarkers and a single read-write storage tape which is infinite to the right only. See [2] for normal definitions.) The following results are known [3]:

(b) Any language accepted by a single-tape TM in time $T(n) \geqq n^2$ can be accepted by a single-tape TM in space $T^{1/2}(n)$.

(c) Any language accepted by an offline TM in time $T(n) \geqq n$ can be accepted by an offline TM in space $(T(n) \log n)^{1/2}$.

The space-bounded simulating machines in results (a)–(c) above have the property that they operate in time exponential in $T(n)$. In this paper, we show that in (b) and (c) the simulating machines can, in fact, be made to operate in time polynomial in $T(n)$. Specifically, we show the following:

(1) Any language accepted by a single-tape TM in time $T(n) \geqq n^2$ can be accepted by a single-tape TM in space $T^{1/2}(n)$ and time $T^2(n)$.

(2) Any language accepted by an offline TM in time $T(n) \geqq n$ can be accepted by an offline TM in space $(T(n) \log n)^{1/2}$ and time $T^{3/2}(n)(T^{1/2}(n) + n/(\log n)^{1/2})$.

---

* Received by the editors July 1, 1981, and in revised form August 23, 1982. This research was supported in part by the National Science Foundation under grant MCS8102853.

† Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455.

‡ Present address, Department of Computer Science, Technion, Haifa 32000, Israel.

[1] In the sequel, the abbreviation TM, without an adjective, refers to a deterministic TM.

(3) Any language accepted by a single-tape nondeterministic TM (NTM) in time $T(n) \geqq n^2$ can be accepted by a single-tape NTM in space $S(n)$ and time $T^2(n)/S(n)$ for any $T^{1/2}(n) \leqq S(n) \leqq T(n)$.

(4) Any language accepted by an offline NTM in time $T(n) \geqq n$ can be accepted by an offline NTM in space $S(n)$ and time $T^2(n)(1+n/S(n))/S(n)$ for any $(T(n) \log n)^{1/2} \leqq S(n) \leqq T(n)$.

Similar results hold for TM's with a multidimensional storage tape.

The simulation techniques we present in this paper differ from the ones in [3], where simulation techniques with similar space bounds but exponential time bounds are given, in that instead of trying all of the possible "crossing sequences" in order to decide if an input is accepted, we record the history of the "crossing moves" of the simulated machine, in a way that makes it possible to reconstruct the computation of the machine in a relatively short time.

Our definition of time/space complexity is the following: A TM or an NTM has time complexity $T(n)$ (respectively, space complexity $S(n)$) if it halts on every input of length n after at most $T(n)$ steps (respectively, after visiting at most $S(n)$ tape cells). The results in this paper, however, remain valid for other definitions of time/space complexity.

2. The history of a computation.

DEFINITION 1. A sequence $B = (B_0, B_1, B_2, \cdots)$ is a *legal partition* of a semi-infinite tape F if for each i, B_i is a "block" of a finite number of consecutive cells of F, B_{i+1} is directly to the right of B_i, and each cell of F belongs to a unique B_i. See Fig. 1.

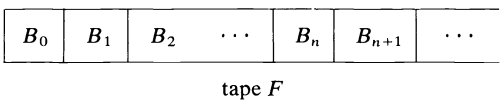


FIG. 1. *A legal partition of F.*

DEFINITION 2. Let M be a single-tape TM and w be in Σ^*. Let B be a legal partition of the tape of M. Then for each nonnegative integer n, the *history of M on input w with respect to B after n steps*, denoted by HIST (M, w, B, n), is a finite sequence of pairs $(d_1, q_1), (d_2, q_2), \cdots, (d_m, q_m)$, where d_i is in $\{-1, +1\}$ and q_i is a state of M, and is defined as follows:

(a) HIST $(M, w, B, 0)$ is the empty sequence.

(b) If M's head does not cross a boundary between two consecutive blocks of B during its nth move on input w, then HIST $(M, w, B, n) =$ HIST $(M, w, B, n-1)$.

(c) If M's head crosses a boundary between two consecutive blocks during the nth move, then HIST $(M, w, B, n) =$ HIST $(M, w, B, n-1), (d, q)$ where d is -1 if T's head moved left, $+1$ otherwise, and q is the state in which M was during this move. (We assume without loss of generality that M changes state before moving its head.)

DEFINITION 3. Let M, w, B be as in Definition 2. Then a (possibly infinite) sequence $(d_1, q_1), \cdots, (d_n, q_n), \cdots$ is the *history of M on input w with respect to B*, denoted by HIST (M, w, B), if the following holds:

(a) For each positive integer n, HIST (M, w, B) contains a prefix which is equal to HIST (M, w, B, n).

(b) Each prefix of HIST (M, w, B) is equal to HIST (M, w, B, n) for some n.

The length of HIST (M, w, B) is the number m of elements in it. (Note that it is possible that $m = \infty$.)

In the sequel we shall be interested in a specific type of legal partitions which we now define.

DEFINITION 4. Let $s \geqq 1$ and $1 \leqq j \leqq s$. Then the *jth legal partition of size s* (see Fig. 2) is the partition $(B_0, B_1, B_2, \cdots)$ defined by:

(a) B_0 consists of cells $1, 2, \cdots, j$.

(b) For $t > 0$, B_t consists of cells $j + (t-1)s + 1$ to $j + ts$.

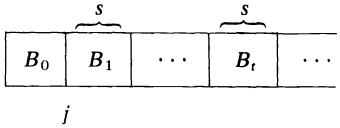


FIG. 2. *The jth partition of size s.*

3. Single-tape TM's. We will need the following lemma which is similar to [3, Lemma 4].

LEMMA 1. *Let M be of time complexity $T(n)$, and let $S(n) = T^{1/2}(n)$. Let w be in Σ^*, $|w|$ = length of $w = n$. For each j, $1 \leqq j \leqq S(n)$, let B^j be the jth partition of size $S(n)$, and let l_j be the length of* HIST (M, w, B^j). *Then for some j_0, $1 \leqq j_0 \leqq S(n)$, $lj_0 \leqq S(n)$.*

Proof. It is not hard to verify that the sum $l_1 + \cdots + l_{S(n)}$ gives the total number of times M's head, on input w, crosses a boundary between two adjacent cells. This number is at most $T(n)$. Hence, $l_1 + \cdots + l_{S(n)} \leqq T(n) \leqq S^2(n)$. This implies that for some j_0, $l_{j_0} \leqq S^2(n)/S(n) = S(n)$. □

DEFINITION 5. $S(n)$ is *fully space constructable in time $T(n)$ by a single-tape* TM if there is a single-tape TM, which on any input of length n, halts after using exactly $S(n)$ cells within time $T(n)$.

THEOREM 1. *Let A be accepted by a single-tape* TM M *in $T(n)$ time, where $T(n) \geqq n^2$. Assume that $S(n) = T^{1/2}(n)$ is fully space constructable in $T^2(n)$ time by a single-tape* TM. *Then A can be accepted by a single-tape* TM M_1 *in $S(n)$ space and $T^2(n)$ time.*

Proof. The TM M_1 simulating M is described below. Informally, on input w, where $|w| = n$, M_1 operates in $S(n)$ stages, where at stage j for $1 \leqq j \leqq S(n)$, M_1 attempts to simulate M on input w by using HIST (M, w, B^j) (B^j is the jth partition of size $S(n)$), which is recorded on M_1's tape. If the length of HIST (M, w, B^j) is greater than $S(n)$, then M_1 stops the simulation and goes to the next stage. Since by Lemma 1 there is at least one j_0 for which the length of HIST (M, w, B^{j_0}) is at most $S(n)$, the simulation eventually succeeds.

M_1 has a 4-track tape, where the track names are COUNT, INPUT, STACK and SCRATCH. COUNT will contain a number j, $1 \leqq j \leqq S(n)$, which indicates the stage M_1 is in. INPUT stores the input string w, STACK and SCRATCH are used in the simulation of M during stage j as described below.

For a given j, the computation of M is (conceptually) divided into time segments by the following rules:

1) Initially M is at time segment 0.

2) M goes from time segment i to time segment $i+1$ immediately after M's head crosses a boundary between two consecutive blocks of B^j. (Note that the total number of time segments is equal to 1 + length of HIST (M, w, B^j).)

Let n_i be the number of steps M makes before it enters time segment i. During the simulation of time segment i, STACK will contain HIST $(M, w, B^j, n_i + 1)$, while SCRATCH will contain the unique block B_k of B^j visited by M's head during this time segment. Initially STACK contains the empty list and SCRATCH contains B_0.

The simulation of time segment $i-1$ is terminated when M's head attempts to cross the boundary between two consecutive blocks. In order to simulate time segment i, M_1 does the following:

Step 1. ‖Update history‖. If there are already $S(n)$ pairs in STACK then terminate the simulation of this stage; else add the appropriate pair (d, q) to STACK.

Step 2. ‖Find current block‖. By scanning STACK from left to right, find the unique integer k such that B_k is the block visited by M's head during the ith time segment. (Note that if STACK $= (d_1, q_1) \cdots (d_t, q_t)$, then $k = d_1 + d_2 + \cdots + d_t$.)

Step 3. ‖Mark previous time segments when B_k was visited‖. Scan again STACK from left to right, and mark each pair (d_m, q_m) such that B_k was visited during the mth time segment.

Step 4. ‖Reconstruct the contents of B_k‖.

4.1. Using INPUT, write on SCRATCH the contents of B_k at the beginning of the computation. If $k = 0$, recompute the value of B_k after the 1st time segment.

4.2. Repeat 4.2.a and 4.2.b until there are no more marked pairs on the STACK.

4.2.a. Find the first marked pair (d_m, q_m). Delete the mark from it.

4.2.b. Using the information (d_m, q_m) above, recompute the contents of B_k at the end of the mth time segment by direct simulation of M.

Verifying the correctness of the above simulation is a straightforward induction, and is left to the reader. Since during the whole computation SCRATCH always contains the contents of a single block, only $S(n)$ space is needed for SCRATCH. Also, STACK contains at most $S(n)$ pairs, and the length of each pair (d, q) is bounded by a constant independent of the input. Hence, $S(n)$ space is enough for STACK. It follows that the computation of M_1 can be carried out in $S(n)$ space.

To prove the time complexity, we first prove that for each j and i, the simulation of time segment i at stage j, as described above, takes $O(T(n))$ time:

Step 1 requires $O(S(n))$ time. Step 2 requires scanning STACK and computing k, and this can be done by a single-tape TM, by a technique similar to the one in [4, Thm. 10.11], in $O(S(n) \log S(n))$ time. Steps 3 and 4.1 can also be computed, by a similar technique, in $O(S(n) \log S(n))$ time.[2]

Each time step 4.2.a is executed requires $O(S(n))$ time, and this step is executed less than $S(n)$ times, which gives a total of $S^2(n) = O(T(n))$ time. The total time needed to simulate the executions of step 4.2.b is bounded by the time complexity of M, which is $T(n)$.[2] Altogether, step 4 requires $O(T(n))$ time, and steps 1, 2, 3 require less than $T(n)$ time. Hence, $O(T(n))$ time is enough to simulate time segment i at stage j.

Since at each stage at most $S(n)$ time segments are simulated, and there are at most $S(n)$ stages, we have that the total time required by M_1 to simulate M is at most $O(S^2(n)T(n)) = O(T^2(n))$. □

The assumption of space constructability of $S(n)$ in Theorem 1 can be removed if we are willing to increase the time of simulation slightly:

COROLLARY 1. *Let A be accepted by a single-tape* TM M *in $T(n)$ time, where $T(n) \geqq n^2$. Let $S(n) = T^{1/2}(n)$. Then A can be accepted by a single-tape* TM M_1 *in $S(n)$ space and $T^{5/2}(n)$ time.*

Proof. M_1 does the simulation for $S(n) = 1, 2, 3, \cdots$. □

One can verify that the simulation technique of Theorem 1 generalizes to nondeterministic computations. (The only difficulty may arise from the fact that when

[2] The time needed in step 4.1 to recompute the value of B_k after the 1st time segment when $k = 0$ is counted in the time requirement of step 4.2.b.

using nondeterminism, each time the contents of a block B_k is reconstructed by M_1, M_1 may use a different computation which satisfies the same history with respect to B^j. However, one can check that this does not affect the correctness of the proof.) Now when the simulating machine M_1 is nondeterministic, it does not have to go through all of the $S(n)$ stages of the simulation. It can guess a j_0, $0 < j_0 \leqq S(n)$, such that the length of HIST (M, w, B^{j_0}) is small, and then simulate M only at stage j_0. This decreases the time complexity of the simulation by a factor of $S(n)$, which results in a "trade-off" phenomenon, stated in the following:

THEOREM 2. *Let A be a set accepted by a single-tape* NTM M *in $T(n)$ time, and let $S(n)$ be such that $T^{1/2}(n) \leqq S(n) \leqq T(n)$. Assume that $S(n)$ is fully space constructable in $T^2(n)/S(n)$ time. Then there is a single-tape* NTM M_1 *accepting A in $S(n)$ space and $T^2(n)/S(n)$ time.*

Proof (sketch). Let M and $S(n)$ be given, and let w be an input to M, $|w| = n$. Let B^j be the jth partition of size $S(n)$. As in Lemma 1, one can prove that there is a j_0 such that the length of HIST (M, w, B^{j_0}) is at most $T(n)/S(n)$. M_1 guesses this j_0, and then carries out the simulation of stage j_0 as in the proof of Theorem 1. Since $T(n)/S(n) \leqq S(n)$, $S(n)$ space is enough to restore HIST (M, w, B^{j_0}). As in the proof of Theorem 1, one can show that the time needed to simulate time segment i is $O(T(n))$. Since there are at most $T(n)/S(n)$ time segments, we have that the total time needed for the simulation is $O(T^2(n)/S(n))$. □

It is unlikely that the time bound of $T^2(n)/S(n)$ in Theorem 2 can be improved. For otherwise, by setting $S(n) = T(n)$, we would have that every set accepted in $T(n)$ time by a single-tape NTM can also be accepted in less than $O(T(n))$ time by a single-tape NTM, which is very unlikely.

4. Offline TM's. An offline TM is a TM with a two way read-only input tape and a single work-tape. (See [2] for a formal definition.) The technique used to prove Theorem 1 can be used to prove

THEOREM 3. *Let A be a set accepted by an offline* TM M *in $T(n) \geqq n$ time. Let $T_1(n) = T^{3/2}(n)(T^{1/2}(n) + n/(\log n)^{1/2})$. Let $S(n) = (T(n) \log n)^{1/2}$ be fully space constructable in time $T_1(n)$ by an offline* TM. *Then there is an offline* TM M_1 *accepting A in $S(n)$ space and $T_1(n)$ time.*

Outline of proof. M_1 simulates M in a manner similar to that described in the proof of Theorem 1, with the following modifications:

(a) The history of a computation of an offline TM M on input w with respect to a legal partition B is a sequence of triples (d, q, k), where d and q are as in Definition 2, and k is an integer, $1 \leqq k \leqq n$ $(n = |w|)$, which indicates the location of the read-only head on the input tape immediately after M's storage head crossed a boundary between two blocks. Thus, a history of length m can be stored in $O(m \log n)$ space.

(b) Let B^j be the jth partition of size $S(n) = (T(n) \log n)^{1/2}$. Then one can prove, in a way similar to the proof of Lemma 1, that for some j_0, $1 \leqq j_0 \leqq S(n)$, the length of HIST (M, w, B^{j_0}) is at most $(T(n)/\log n)^{1/2}$ and hence it can be stored in $(T(n) \log n)^{1/2}$ space.

(c) In steps 4.1 and 4.2.a of the simulation, M_1 has also to move and reposition the input head. This can be done in $O(n)$ additional steps at each time.

Space and time analysis. Since each block contains at most $O((T(n) \log n)^{1/2})$ cells, and we have to store only histories of length $(T(n)/\log n)^{1/2}$, which can be done in $(T(n) \log n)^{1/2}$ space, only $(T(n) \log n)^{1/2}$ space is needed. The total time required for repositioning the input head during the simulation of time segment i at stage j is at most $n(T(n)/\log n)^{1/2}$, since there are at most $(T(n)/\log n)^{1/2}$ time

segments. It can be shown that this implies that the time required to simulate time segment i at stage j is $O(T(n)+n(T(n)/\log n)^{1/2})$. Since there are at most $(T(n)/\log n)^{1/2}$ time segments at each stage, and there are $(T(n)\log n)^{1/2}$ stages, the total time needed for the simulation is $O(T(n)(T(n)+n(T(n)/\log n)^{1/2}))=O(T^{3/2}(n)(T^{1/2}(n)+n/(\log n)^{1/2}))$. □

The analogue of Theorem 2 for offline NTM's is:

THEOREM 4. *Let A be accepted by an offline* NTM *in $T(n)$ time, and let $S(n)$ be such the $(T(n)\log n)^{1/2}\leqq S(n)\leqq T(n)$. Assume that $S(n)$ is fully space constructable in $T^2(n)(1+n/S(n))/S(n)$ time. Then there is an offline* NTM *accepting A in $S(n)$ space and $T^2(n)(1+n/S(n))/S(n)$ time.*

Proof. Omitted. □

5. Related results. We have presented time–space tradeoff results concerning single-tape and offline TM's. These results can be used to sharpen some known hierarchy results. For example, the following result follows from Theorem 1 and diagonalization: Let $T(n)\geqq n^2$. Let $S(n)$ be a space bound such that $\lim_{n\to\infty} T^{1/2}(n)/S(n)=0$. Then there is a positive integer k such that the class of languages accepted by single-tape TM's operating in space $S(n)$ and time $T^k(n)$ properly contains the class of languages accepted by single-tape TM's operating in time $T(n)$.

The simulation technique described in this paper can be generalized to hold for offline TM's whose storage tape is multidimensional. (See [2], [5] for a formal definition.) An r-dimensional storage tape is a tape where each cell is identified with an element $\bar{x}=(x_1,\cdots,x_r)$ of Z^r. (Z is the set of all integers.) Two cells $\bar{x}$ and $\bar{y}$ are adjacent if for some $1\leqq i_0\leqq r$, $|x_{i_0}-y_{i_0}|=1$, and $x_i=y_i$ otherwise. Thus, the storage head of the machine can move in $2r$ directions.

The following is a generalization of Theorem 3. A similar result, but with exponential time complexity, has been shown earlier in [5].

THEOREM 5. *Let A be a set accepted by an offline* TM *M with an r-dimensional storage tape in $T(n)\geqq n$ time. Let $T_1(n)=T^2(n)(n/(T(n)\log T(n))^{1/(r+1)}+\log T(n))$. Let $S(n)=(T(n)\log T(n))^{r/(r+1)}$ be fully space constructable in $T_1(n)$ time by a 1-dimensional offline* TM. *Then A can be accepted by an offline* TM *M_1 with an r-dimensional storage tape in $S(n)$ space and $T_1(n)$ time.*

We shall sketch the proof of Theorem 5 for the case when $r=2$. Let $s\geqq 1$ and $1\leqq j\leqq s$. Then the jth legal partition of size s of the 2-dimensional space, denoted by B^j, is a partition of the plane to blocks, where each block is a square which for some u and v, contains all the cells (x, y) satisfying $us+j<x\leqq(u+1)s+j$, $vs+j<y\leqq(v+1)s+j$. (See Fig. 3.) Let s be given, and let $\bar{x}$, $\bar{y}$ be 2 adjacent cells of the 2-dimensional plane. Then it is easy to verify that for some unique j, $1\leqq j\leqq s$, $\bar{x}$ and $\bar{y}$ belong to 2 distinct blocks of the jth legal partition of size s of the plane.

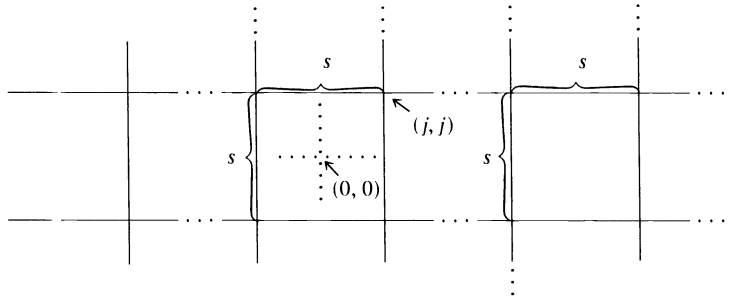


FIG. 3. *The jth legal partition of size s of the plane.*

The (2-dimensional) history of the computation of M on input w with respect to B^j is a sequence of 4-tuples (d, q, k, p), where d is one of four directions, q and k are as in the definition of the history of an offline TM, and p is an integer, $1 \leqq p \leqq s$, which indicates the exact location of M's storage head immediately after it crossed a boundary between 2 adjacent blocks. (Note that the history of an r-dimensional TM will be a sequence of $(r+2)$-tuples.) A history of length m can be stored in $O(m(\log s + \log n))$ space.

The analogue of Lemma 1 for 2-dimensional space is the following:

Lemma 2. *Let M be an offline* TM *with a 2-dimensional storage tape operating in time $T(n)$. Let w be in Σ^*, $|w| = n$,* and let $s \leqq T(n)$. *Then for some $1 \leqq j \leqq s$, the length of* HIST (M, w, B^j) *is at most $T(n)/s$.*

Clearly, the space required to store an s by s block is $O(s^2)$. The space required to store a history of length $T(n)/s$ is $O((T(n)/s)(\log s + \log n))$. Substituting $s = (T(n) \log T(n))^{1/3}$, we get that the space required for both tasks, and hence for the simulation, is $O(T(n) \log T(n))^{2/3}$. We omit the time analysis.

Using a well-known technique for simulating multidimensional TM's by one-dimensional TM's (see [2]) we get:

COROLLARY 2. *Let A be accepted by an offline* TM *M with an r-dimensional storage tape in $T(n) \geqq n$ time. Then A can be accepted by a one-dimensional offline* TM *in $O(T(n) \log T(n))^{r/(r+1)}$ space and $O(T^e(n))$ time for some e.*

In the corollary above, the exponent e depends on the dimension r of the TM M. It would be interesting to know whether there is a fixed e such that $(T(n) \log T(n))^{r/(r+1)}$ space and $T^e(n)$ time are sufficient for all r's.

REFERENCES

[1] J. E. HOPCROFT, W. J. PAUL AND L. G. VALIANT, *On time versus space*, J. Assoc. Comput. Mach., 24 (1977), pp. 332–337.

[2] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[3] ———, *Relations between time and tape complexities*, J. Assoc. Comput. Mach., 15 (1968), pp. 414–427.

[4] ———, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, MA, 1969.

[5] M. C. LOUI, *A space bound for one-tape multidimensional Turing machines*, Tech. Memo. TM-145, Lab. for Computer Science, Massachusetts Institute of Technology, Cambridge, 1979; Theoret. Computer Sci. to appear.

[6] M. S. PATERSON, *Tape bounds for time-bounded Turing machines*, J. Comput. System Sci., 6 (1972), pp. 116–124.

SIAM J. COMPUT.
Vol. 12, No. 2, May 1983

0097-5397/83/1202-0014 $01.25/0

GENERALIZED p-ADIC CONSTRUCTIONS*

MARKUS LAUER†

Abstract. In this paper "abstract lifting algorithms" for polynomial equations over a commutative ring with identity element are developed. They lift solutions modulo some ideal I to solutions modulo another ideal $J \subset I$ (e.g. $J = I^T$). These algorithms are obtained by applying Newton's method to the polynomial equations and include for example the Hensel-type polynomial factorization algorithms as special cases.

Key words. algebraic algorithms, modular algorithms, p-adic constructions

1. Introduction. Hensel's lemma is a strong and widely used tool in algebraic computation; see, for example, [Mus71], [Mus75], [Yun74a], [Yun74b], [W&R75]. It is applied to factorization, gcd calculation, division with remainder and squarefree decomposition of integral polynomials. It provides, in the case of univariate polynomials, a method to compute (or "lift") from a solution of the given problem mod p^T a solution mod p^{T+1}, where p is a prime integer, and can be generalized to multivariate polynomials. Zassenhaus in [Zas69] proposed a "quadratic" version of Hensel's lemma where factors of a polynomial mod p^T are lifted to factors mod p^{2T}.

Yun in [Yun75] pointed out the similarities between Hensel's lemma and Newton's method; more precisely he showed that Hensel's lemma is an application of Newton's method in an algebraic setting, namely polynomial factorization.

This Hensel–Newton technique can be applied to more general problems such as to the solution of polynomial equations. Lewis in [Lew69] used it to lift solutions of (nonlinear) diophantine equations modulo prime powers. In fact our basic Theorem 4.2 is a fairly straightforward generalization of Lewis's Theorem D to commutative rings with identity and finitely generated ideals.

Our theorems lead to abstract algorithms in the sense of Musser in [Mus75]: an abstract algorithm works over abstract algebraic domains (in our case commutative rings with identity), and must terminate. No consideration is given to the effectiveness of its steps; this has to be regarded if it is specialized to a concrete algorithm by specializing the abstract algebraic domain to a concrete one, for example to the ring of integral polynomials.

After the next two (preliminary) sections we present in § 4 the generalization of the "Lewis–Hensel–Newton method" to polynomial equations over a commutative ring with identity which leads to an abstract linear lifting algorithm. In § 5 we apply Zassenhaus's technique to obtain a quadratic lifting algorithm, and Musser's iterative method to get our most general abstract lifting algorithm. Section 6 shows some of the difficulties which arise when the abstract lifting algorithms are specialized to concrete ones.

2. Newton's method and Hensel's lemma. It is well known that Hensel's lemma can be regarded as an application of Newton's (numerical) method in an algebraic setting; see [Yun75] for a detailed discussion. Let us briefly recapitulate this view.

Newton's method provides an iteration formula for approximating a root of a differentiable function $f(x)$: Let $f(\bar{x}) = 0$ and let x_0 be an approximation for $\bar{x}$, i.e.,

* Received by the editors May 17, 1978, and in revised form September 1, 1980.
† Institut für Informatik I, Universität Karlsruhe, Karlsruhe, West Germany.

$\bar{x} = x_0 + \delta$ where δ is sufficiently small. Assume that $f'(x_0) \neq 0$, where f' denotes the first derivative of f. Taylor series expansion of f at x_0 yields

$$0 = f(\bar{x}) = f(x_0 + \delta) = f(x_0) + \delta \cdot f'(x_0) + O(\delta^2).$$

Since δ is small we have

$$0 \approx f(x_0) + \delta f'(x_0);$$

since $f'(x_0) \neq 0$ we can solve this linear equation for δ:

$$\delta \approx -\frac{f(x_0)}{f'(x_0)},$$

and so get a new approximation

$$\bar{x} \approx x_1 = x_0 + \delta = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Hensel's lemma, in its formulation for factoring univariate integral polynomials, reads as follows:

LEMMA 2.1. (Hensel). *Let $A_1, A_2, C \in \mathscr{Z}[x]$, where $\mathscr{Z}$ denotes the ring of integers, $p \in \mathscr{Z}$ a prime, T a positive integer. Suppose that*

$$A_1 A_2 \equiv C \bmod p^T,$$

and that A_1 and A_2 are relatively prime mod p. *Then there exist $A_1^*, A_2^* \in \mathscr{Z}[x]$ such that*

$$A_1^* A_2^* \equiv C \bmod p^{T+1}.$$

Proof. Let

$$A_1 A_2 = C + V p^T, \qquad V \in \mathscr{Z}[x];$$

set

$$A_1^* = A_1 + B_1 p^T, \qquad A_2^* = A_2 + B_2 p^T.$$

Then

$$\begin{aligned} A_1^* A_2^* &= A_1 A_2 + (B_1 A_2 + B_2 A_1) p^T + B_1 B_2 p^{2T} \\ &\equiv C + (V + B_1 A_2 + B_2 A_1) p^T \bmod p^{T+1} \\ &\equiv C \bmod p^{T+1}, \end{aligned}$$

if

$$V + B_1 A_2 + B_2 A_1 \equiv 0 \bmod p.$$

Since A_1 and A_2 are relatively prime mod p, this linear congruence can be solved for B_1 and B_2.

To see how this is an application of Newton's method we note that (A_1, A_2) is a mod p^T-solution ("approximation") of the equation

$$x_1 x_2 - C = 0$$

over $\mathscr{Z}[x]$. We get the new approximation by adding an increment: $A_i^* = A_i + B_i p^T$, $i = 1, 2$. (In the "numerical" Newton's method: $x_1 = x_0 + \delta$.) Plugging the new approximation (with indeterminate increment) into the equation, expanding, and cutting off high order terms we get a linear equation for the increment. This can be solved if the

old approximation satisfies a "solvability condition" (A_1 and A_2 relatively prime mod p for Hensel's lemma, $f'(x_0) \neq 0$ for the "numerical" Newton's method).

One difference, however, should be noted: the numerical Newton's method can be used only if a root of f is known to exist; in Hensel's lemma one only needs a factorization mod p to start with.

After establishing conventions for notation in the next section we will generalize that approach to arbitrary polynomial equations.

3. Notation. We use the standard basic facts of ring theory as described for example in [Lün73]. In all that follows R denotes a commutative ring with identity element; $\mathscr{Z}$ denotes the ring of integers.

DEFINITION 3.1. If k, m are positive integers, $R_{k\times m}$ is the set of all $k \times m$ matrices $W = (w_{ij})$ over R, i.e. with $w_{ij} \in R, i = 1, \cdots, k, j = 1, \cdots, m$. If $W \in R_{k\times m}$, W^* denotes the transposed matrix, an element of $R_{m\times k}$. We write $R^{(n)}$ for $R_{n\times 1}$ and $\bar{a} = (a_1, \cdots, a_n)^*$ for the elements of $R^{(n)}$.

DEFINITION 3.2. If I is an ideal in R we write $I \trianglelefteq R$. If $r_1, \cdots, r_m \in R$, $(r_1, \cdots, r_m)$ denotes the ideal in R generated by $\{r_1, \cdots, r_m\}$. If $R_1, \cdots, R_m$ are subsets of R, $(R_1, \cdots, R_m)$ denotes the ideal in R generated by $\bigcup_{i=1}^{m} R_i$. If $V, W \in R_{k\times m}$ and $I \trianglelefteq R$, $V \equiv W \bmod I$ means $v_{ij} \equiv w_{ij} \bmod I, i = 1, \cdots, k, j = 1, \cdots, m$.

DEFINITION 3.3. Let $f \in R[x_1, \cdots, x_r]$, $r \geqq 1$. Let $y_1, \cdots, y_r$ be new indeterminates. Then $\partial f/\partial x_j$ denotes the coefficient of the term y_j in $f(x_1+y_1, \cdots, x_r+y_r) \in R[x_1, \cdots, x_r][y_1, \cdots, y_r]$. ($\partial f/\partial x_j \in R[x_1, \cdots, x_r]$.)

4. Linear lifting. We now generalize the linear Hensel construction. That is, we apply Newton's method to polynomial equations over R and lift solutions modulo powers of ideals.

LEMMA 4.1 (Taylor expansion). *Let $f \in R[x_1, \cdots, x_r]$, $r \geqq 1$; $y_1, \cdots, y_r$ new indeterminates. Then*

$$f(x_1+y_1, \cdots, x_r+y_r) = f(x_1, \cdots, x_r) + \sum_{j=1}^{r} \frac{\partial f}{\partial x_j} y_j + F,$$

where $F \in R[x_1, \cdots, x_r][y_1, \cdots, y_r]$, and

$$F \equiv 0 \bmod (y_1, \cdots, y_r)^2 \trianglelefteq R[x_1, \cdots, x_r][y_1, \cdots, y_r].$$

Proof. Obvious from Definition 3.3. □

Now we are ready for a first generalization of the Hensel construction.

THEOREM 4.2. *Let I be a finitely generated ideal in R; $f_1, \cdots, f_n \in R[x_1, \cdots, x_r]$, $r \geqq 1$; $a_1, \cdots, a_r \in R$ with*

$$f_i(a_1, \cdots, a_r) \equiv 0 \bmod I, \qquad i = 1, \cdots, n.$$

For

$$u_{ij} = \frac{\partial f_i}{\partial x_j}(a_1, \cdots, a_r) \in R, \qquad i = 1, \cdots, n, \qquad j = 1, \cdots, r,$$

let each system of linear equations with coefficient matrix $U = (u_{ij}) \in R_{n\times r}$ have a solution mod I. *Then for each positive integer T there exist $a_1^{(T)}, \cdots, a_r^{(T)} \in R$ with*

$$f_i(a_i^{(T)}, \cdots, a_r^{(T)}) \equiv 0 \bmod I^T, \qquad i = 1, \cdots, n,$$

and

$$a_j^{(T)} \equiv a_j \bmod I, \qquad j = 1, \cdots, r.$$

Proof. We proceed by induction on T. If $T=1$, we set $a_j^{(T)}=a_j$, $j=1,\cdots,r$. So let $T\geqq 1$ and assume that the proposition is true for T. So there exist $a_1^{(T)},\cdots,a_r^{(T)}\in R$ with

$$f_i(a_1^{(T)},\cdots,a_r^{(T)})\equiv 0 \bmod I^T, \qquad i=1,\cdots,n, \tag{1}$$

and

$$a_j^{(T)}\equiv a_j \bmod I, \qquad j=1,\cdots,r. \tag{2}$$

Since I is finitely generated, I^T is finitely generated. Let

$$I^T=(q_1,\cdots,q_m). \tag{3}$$

By (1) and (3) there exist $v_{ik}\in R$, $i=1,\cdots,n$, $k=1,\cdots,m$, with

$$f_i(a_1^{(T)},\cdots,a_r^{(T)})=\sum_{k=1}^{m} v_{ik}q_k. \tag{4}$$

We set

$$a_j^{(T+1)}=a_j^{(T)}+B_j, \qquad j=1,\cdots,r, \tag{5}$$

with

$$B_j=\sum_{k=1}^{m} b_{jk}q_k\in I^T, \qquad b_{jk}\in R, \quad j=1,\cdots,r, \quad k=1,\cdots,m, \tag{6}$$

where we want to choose the "unknowns" b_{jk} such that the proposition of the theorem holds for $a_j^{(T+1)}$, $j=1,\cdots,r$.

Let

$$u_{ij}^{(T)}=\frac{\partial f_i}{\partial x_j}(a_1^{(T)},\cdots,a_r^{(T)}), \qquad i=1,\cdots,n, \quad j=1,\cdots,r. \tag{7}$$

We then have, by (5),

$$\begin{aligned} f_i(a_1^{(T+1)},\cdots,a_r^{(T+1)}) &= f_i(a_1^{(T)}+B_1,\cdots,a_r^{(T)}+B_r) \\ &\equiv f_i(a_1^{(T)},\cdots,a_r^{(T)})+\sum_{j=1}^{r} u_{ij}^{(T)}B_j \bmod I^{T+1} \\ &\qquad \text{(by Lemma 4.1. and (7))} \\ &\equiv \sum_{k=1}^{m} v_{ik}q_k+\sum_{j=1}^{r} u_{ij}^{(T)}\sum_{k=1}^{m} b_{jk}q_k \qquad \text{(by (4) and (6))} \\ &\equiv \sum_{k=1}^{m}\Big(v_{ik}+\sum_{j=1}^{r} u_{ij}^{(T)}b_{jk}\Big)q_k \bmod I^{T+1}. \end{aligned} \tag{8}$$

Now by (3), (8) $\equiv 0 \bmod I^{T+1}$ if

$$v_{ik}+\sum_{j=1}^{r} u_{ij}^{(T)}b_{jk}\equiv 0 \bmod I, \qquad i=1,\cdots,n, \quad k=1,\cdots,m. \tag{9}$$

These are m systems of linear equations, each with coefficient matrix $(u_{ij}^{(T)})$, and we want solutions mod I. By (2) and (7),

$$u_{ij}^{(T)}\equiv u_{ij} \bmod I, \qquad i=1,\cdots,n, \quad j=1,\cdots,r.$$

So (9) is equivalent to

$$v_{ik} + \sum_{j=1}^{r} u_{ij} b_{jk} \equiv 0 \bmod I, \qquad i = 1, \cdots, n, \quad k = 1, \cdots, m. \tag{10}$$

By the assumption on (u_{ij}), b_{jk} which satisfy (10) do exist, and so, for these b_{jk}

$$f_i(a_1^{(T+1)}, \cdots, a_r^{(T+1)}) \equiv 0 \bmod I^{T+1}, \qquad i = 1, \cdots, n.$$

Clearly, by (2) and (5),

$$a_j^{(T+1)} \equiv a_j \bmod I, \qquad j = 1, \cdots, r. \qquad \square$$

As a sample application let us see how we get the Hensel construction as a specialization of that theorem. Let $R = \mathscr{Z}[y]$ and $I = (p) \trianglelefteq R$, where $p \in \mathscr{Z}$ is a prime. Let $C \in R$ and

$$f_1 = x_1 x_2 - C \in R[x_1, x_2].$$

Let $A_1, A_2 \in R$ with

$$f_1(A_1, A_2) = A_1 A_2 - C \equiv 0 \bmod I = (p).$$

Then

$$u_{11} = \frac{\partial f_1}{\partial x_1}(A_1, A_2) = A_2, \qquad u_{12} = \frac{\partial f_1}{\partial x_2}(A_1, A_2) = A_1.$$

Each system of linear equations with coefficient matrix $U = (A_2, A_1) \in R_{1\times 2}$ has a solution mod (p) if and only if A_1 and A_2 are relatively prime mod (p). If $A_1^{(T)}, A_2^{(T)}$ satisfy

$$A_1^{(T)} A_2^{(T)} - C = V p^T \equiv 0 \bmod (p)^T = (p)^T,$$

we get $A_1^{(T+1)} = A_1^{(T)} + B_1 p^T$, $A_2^{(T+1)} = A_2^{(T)} + B_2 p^T$, by solving (for B_1 and B_2)

$$V + B_1 A_2 + B_2 A_1 \equiv 0 \bmod (p).$$

The proof of Theorem 4.2. is constructive and leads to our first abstract lifting algorithm. For each lifting step we have to solve the linear systems (10). The coefficient matrix $U = (u_{ij})$ remains always the same, so we can do a "preconditioning" which not only increases the efficiency of the algorithm (though we are not considering the efficiency of an abstract algorithm, preconditioning does increase efficiency if we specialize it to a concrete algorithm), but also turns out to be useful in the next section.

If $U \in R_{n\times r}$, each linear system with coefficient matrix U has a solution mod I if and only if U has a "right inverse" mod I, i.e. there exists a $W \in R_{r\times n}$ such that

$$UW \equiv E \bmod I,$$

where E denotes the $n \times n$ identity matrix. Then, if $\bar{a} \in R^{(n)}$, the system

$$U\bar{x} = \bar{a}$$

has the solution

$$\bar{b} = W\bar{a} \in R^{(r)}.$$

So our abstract lifting algorithm takes as an additional input a right inverse of U.

ALGORITHM H (Hensel's linear lifting).

Inputs: $P=\{p_1,\cdots,p_e\}\subset R$ with $I=(p_1,\cdots,p_e)\trianglelefteq R$;
$f_1,\cdots,f_n\in R[x_1,\cdots,x_r]$, $r\geqq 1$;
$a_1,\cdots,a_r\in R$ with
$$f_i(a_1,\cdots,a_r)\equiv 0 \bmod I,\ i=1,\cdots,n;$$
$W\in R_{r\times n}$ with
$$UW\equiv E \bmod I, \text{ where}$$
$$U=(u_{ij})\in R_{n\times r} \text{ and}$$
$$u_{ij}=(\partial f_i/\partial x_j)(a_1,\cdots,a_r),\ i=1,\cdots,n,\ j=1,\cdots,r;$$
T, a positive integer.

Outputs: $A_1,\cdots,A_r\in R$ with
$$f_i(A_1,\cdots,A_r)\equiv 0 \bmod I^T,\ i=1,\cdots,n, \text{ and}$$
$$A_j\equiv a_j \bmod I,\ j=1,\cdots,r.$$

(1)[Initialize.]
set $t\leftarrow 1$, $\bar{a}^{(t)}=(a_1^{(t)},\cdots,a_r^{(t)})^*\leftarrow(a_1,\cdots,a_r)^*\in R^{(r)}$.
(2)[Lift.]
while $t<T$ do
begin compute $q_1,\cdots,q_m\in R$ with
$$I^t=(q_1,\cdots,q_m)\trianglelefteq R;$$
$$\bar{v}_k=(v_{1k},\cdots,v_{nk})^*\in R^{(n)},\ k=1,\cdots,m, \text{ with}$$
$$f_i(a_1^{(t)},\cdots,a_r^{(t)})=\textstyle\sum_{k=1}^{m} v_{ik}q_k,\ i=1,\cdots,n;$$
set $\bar{b}_k\leftarrow -W\bar{v}_k\in R^{(r)}$, $k=1,\cdots,m$;
$\bar{a}^{(t+1)}\leftarrow\bar{a}^{(t)}+\sum_{k=1}^{m}\bar{b}_k q_k\in R^{(r)}$;
$t\leftarrow t+1$
end.
(3)[Finish.]
set $A_1\leftarrow a_1^{(t)},\cdots,A_r\leftarrow a_r^{(t)}$; exit

5. Quadratic lifting and another generalization. In [Zas69] Zassenhaus proposed a "quadratic Hensel construction" which can be carried over to the construction in Theorem 4.2. If we re-examine the proof of Theorem 4.2 we see that by Lemma 4.1 the congruence (8) holds not only mod I^{T+1} but even mod I^{2T}. So we may try to determine the $a_j^{(T+1)}$ such that (8) $\equiv 0 \bmod I^{2T}$. By (3) this holds if

$$v_{ik}+\sum_{j=1}^{r} u_{ij}^{(T)}b_{jk}\equiv 0 \bmod I^T. \tag{11}$$

Since in general $u_{ij}^{(T)}\not\equiv u_{ij} \bmod I^T$ we cannot pass from $u_{ij}^{(T)}$ to u_{ij} as we did in (10). But the next theorem shows that (11) can be solved.

THEOREM 5.1. *Let I, f_i, a_j, $U=(u_{ij})$ be as in theorem* 4.2; *let $W\in R_{r\times n}$ with*

$$UW\equiv E \bmod I,$$

where E denotes the $n\times n$ identity matrix over R. Let S be a nonnegative integer. Then there exist $a_1^{(S)},\cdots,a_r^{(S)}\in R$ with

$$f_i(a_1^{(S)},\cdots,a_r^{(S)})\equiv 0 \bmod I^{2^S},\qquad i=1,\cdots,n,$$
$$a_j^{(S)}\equiv a_j \bmod I,\qquad j=1,\cdots,r.$$

Let

$$u_{ij}^{(S)}=\frac{\partial f_i}{\partial x_j}(a_1^{(S)},\cdots,a_r^{(S)})\in R,\qquad i=1,\cdots,n,\quad j=1,\cdots,r,$$

$$U^{(S)}=(u_{ij}^{(S)})\in R_{n\times r}.$$

Then there exists a $W^{(S)} \in R_{r\times n}$ *with*

$$U^{(S)}W^{(S)} \equiv E \bmod I^{2^S}.$$

Proof. Though the existence of $a_1^{(S)}, \cdots, a_r^{(S)}$ is guaranteed by Theorem 4.2 with $T = 2^S$, we will prove it again, now using a quadratic construction. We proceed by induction on S. If $S = 0$, we set $a_j^{(S)} = a_j$, $j = 1, \cdots, r$, and $W^{(S)} = W$. So let $S \geqq 0$ and assume that the proposition is true for S. So there exist $a_1^{(S)}, \cdots, a_r^{(S)} \in R$ with

$$f_i(a_1^{(S)}, \cdots, a_r^{(S)}) \equiv 0 \bmod I^{2^S}, \qquad i = 1, \cdots, n, \tag{12}$$

and

$$a_j^{(S)} \equiv a_j \bmod I, \qquad j = 1, \cdots, r,$$

and $W^{(S)} \in R_{r\times n}$ with

$$U^{(S)}W^{(S)} \equiv E \bmod I^{2^S}. \tag{13}$$

Let

$$I^{2^S} = (q_1, \cdots, q_m) \trianglelefteq R. \tag{14}$$

By (12) and (14) there exist $v_{ik} \in R$, $i = 1, \cdots, n$, $k = 1, \cdots, m$, with

$$f_i(a_1^{(S)}, \cdots, a_r^{(S)}) = \sum_{k=1}^{m} v_{ik}q_k. \tag{15}$$

We set

$$a_j^{(S+1)} = a_j^{(S)} + B_j, \qquad j = 1, \cdots, r, \tag{16}$$

with

$$B_j = \sum_{k=1}^{m} b_{jk}q_k \in I^{2^S}, \qquad b_{jk} \in R, \quad j = 1, \cdots, r, \quad k = 1, \cdots, m. \tag{17}$$

We have by (16)

$$\begin{aligned} f_i(a_1^{(S+1)}, \cdots, a_r^{(S+1)}) &= f_i(a_1^{(S)} + B_1, \cdots, a_r^{(S)} + B_r) \\ &\equiv f_i(a_1^{(S)}, \cdots, a_r^{(S)}) + \sum_{j=1}^{r} u_{ij}^{(S)}B_j \bmod I^{2^{S+1}} \\ &\qquad \text{(by Lemma 4.1. and (7))} \\ &\equiv \sum_{k=1}^{m} v_{ik}q_k + \sum_{j=1}^{r} u_{ij}^{(S)} \sum_{k=1}^{m} b_{jk}q_k \qquad \text{(by (15) and (17))} \\ &\equiv \sum_{k=1}^{m} \left(v_{ik} + \sum_{j=1}^{r} u_{ij}^{(S)}b_{jk}\right)q_k \bmod I^{2^{S+1}}. \end{aligned} \tag{18}$$

By (13) the m systems of linear "equations"

$$v_{ik} + \sum_{j=1}^{r} u_{ij}^{(S)}b_{jk} \equiv 0 \bmod I^{2^S}$$

have solutions $b_{jk} \bmod I^{2^S}$, and so by (14)

$$(18) \equiv 0 \bmod I^{2^{S+1}}.$$

Clearly by (16)

$$a_j^{(S+1)} \equiv a_j^{(S)} \equiv a_j \bmod I, \qquad j = 1, \cdots, r.$$

Now, by (16) and (17)

$$U^{(S+1)} \equiv U^{(S)} \bmod I^{2^S},$$

so by (13)

$$U^{(S+1)}W^{(S)} \equiv E \bmod I^{2^S}. \tag{19}$$

Thus by (14) there exist $D_k \in R_{n\times n}$, $k = 1, \cdots, m$, with

$$U^{(S+1)}W^{(S)} = E + \sum_{k=1}^{m} D_k q_k. \tag{20}$$

We set

$$W^{(S+1)} = W^{(S)} + \sum_{k=1}^{m} Z_k q_k, \tag{21}$$

where we want to choose the "unknowns" $Z_k \in R_{r\times n}$ such that the proposition of the theorem holds for $W^{(S+1)}$. Then, by (21),

$$U^{(S+1)}W^{(S+1)} = U^{(S+1)}W^{(S)} + \sum_{k=1}^{m} U^{(S+1)}Z_k q_k$$

$$= E + \sum_{k=1}^{m} (D_k + U^{(S+1)}Z_k)q_k \qquad \text{by (20).} \tag{22}$$

By (14), (22) $\equiv E \bmod I^{2^{S+1}}$ if

$$D_k + U^{(S+1)}Z_k \equiv 0 \bmod I^{2^S}, \qquad k = 1, \cdots, m.$$

By (19) this holds if we set

$$Z_k = -W^{(S)}D_k. \qquad \square$$

This constructive proof leads to the next abstract lifting algorithm. It lifts not only the $a_j^{(S)}$ but also the $W^{(S)}$. Since the final $W^{(S)}$ will be needed in another algorithm, it belongs to the output.

ALGORITHM Z (Zassenhaus' quadratic lifting).

Inputs: $P = \{p_1, \cdots, p_e\} \subset R$ with $I = (p_1, \cdots, p_e) \trianglelefteq R$;
$f_1, \cdots, f_n \in R[x_1, \cdots, x_r]$, $r \geqq 1$;
$a_1, \cdots, a_r$ with
$f_i(a_1, \cdots, a_r) \equiv 0 \bmod I$, $i = 1, \cdots, n$;
$W \in R_{r\times n}$ with
$UW \equiv E \bmod I$, where E is the $n \times n$ identity matrix,
$U = (u_{ij}) \in R_{n\times r}$ and
$u_{ij} = (\partial f_i/\partial x_j)(a_1, \cdots, a_r)$, $i = 1, \cdots, n$, $j = 1, \cdots, r$;
S, a nonnegative integer.

Outputs: $A_1, \cdots, A_r \in R$ with
$f_i(A_1, \cdots, A_r) \equiv 0 \bmod I^{2^S}$, $i = 1, \cdots, n$, and
$A_j \equiv a_j \bmod I$, $j = 1, \cdots, r$;
$\bar{W} \in R_{r\times n}$ with
$\bar{U}\bar{W} \equiv E \bmod I^{2^S}$, where
$\bar{U} = (\bar{u}_{ij}) \in R_{n\times r}$ and
$\bar{u}_{ij} = (\partial f_i/\partial x_j)(A_1, \cdots, A_r)$, $i = 1, \cdots, n$, $j = 1, \cdots, r$.

(1)[Initialize.]
set $s \leftarrow 0$, $\bar{a}^{(s)} = (a_1^{(s)}, \cdots, a_r^{(s)})^* \leftarrow (a_1, \cdots, a_r)^* \in R^{(r)}$, $W^{(s)} \leftarrow W$.

(2)[Lift.]
while $s < S$ do
begin compute $q_1, \cdots, q_m \in R$ with
$$I^{2^s} = (q_1, \cdots, q_m) \trianglelefteq R;$$
$\bar{v}_k = (v_{1k}, \cdots, v_{nk})^* \in R^{(n)}$, $k = 1, \cdots, m$, with
$$f_i(a_1^{(s)}, \cdots, a_r^{(s)}) = \sum_{k=1}^m v_{ik} q_k, \quad i = 1, \cdots, n;$$
set $\bar{b}_k \leftarrow -W^{(s)} \bar{v}_k \in R^{(r)}$, $k = 1, \cdots, m$;
$\bar{a}^{(s+1)} \leftarrow \bar{a}^{(s)} + \sum_{k=1}^m \bar{b}_k q_k \in R^{(r)}$;
compute $U^{(s+1)} \in R_{n \times r}$ with
$$u_{ij}^{(s+1)} = (\partial f_i / \partial x_j)(a_1^{(s+1)}, \cdots, a_r^{(s+1)}),$$
$$i = 1, \cdots, n, \ j = 1, \cdots, r;$$
$D_k \in R_{n \times n}$, $k = 1, \cdots, m$, with
$$U^{(s+1)} W^{(s)} = E + \sum_{k=1}^m D_k q_k;$$
set $Z_k \leftarrow -W^{(s)} D_k \in R_{r \times n}$, $k = 1, \cdots, m$;
$W^{(s+1)} \leftarrow W^{(s)} + \sum_{k=1}^m Z_k q_k$;
$s \leftarrow s + 1$
end.
(3)[Finish.]
set $A_1 \leftarrow a_1^{(s)}, \cdots, A_r \leftarrow a_r^{(s)}$, $\bar{W} \leftarrow W^{(s)}$; exit.

Musser in [Mus71, 75] generalized the Hensel construction in a different way to nonprincipal ideals: the factors are not lifted modulo powers of an ideal but modulo ideals which are generated by powers of the basis elements of the starting ideal. We apply this technique to the solution of polynomial equations, again replacing powers of principal ideals by powers of finitely generated ideals. To do so we make use of the second isomorphism theorem for rings (see for example [Lün73]):

If $I, J \trianglelefteq R$ and $I \subset J$, then

$$R/J \simeq (R/I)/(J/I) \tag{23}$$

via the mapping

$$r + J \mapsto (r + I) + (J/I).$$

THEOREM 5.2. *Let $P_1, \cdots, P_m$ be finite subsets of R and $I = (P_1, \cdots, P_m) \trianglelefteq R$; $f_1, \cdots, f_n \in R[x_1, \cdots, x_r]$, $r \geqq 1$; $a_1, \cdots, a_r \in R$ with $f_i(a_1, \cdots, a_r) \equiv 0 \bmod I$, $i = 1, \cdots, n$. Let $W \in R_{r \times n}$ with*

$$UW \equiv E \bmod I,$$

where E denotes the $n \times n$ identity matrix over R, $U = (u_{ij}) \in R_{n \times r}$ and

$$u_{ij} = \frac{\partial f_i}{\partial x_j}(a_1, \cdots, a_r) \in R, \qquad i = 1, \cdots, n, \quad j = 1, \cdots, r.$$

Then for each sequence of nonnegative integers $j_1, \cdots, j_m$ there exist $A_1, \cdots, A_r \in R$ with

$$f_i(A_1, \cdots, A_r) \equiv 0 \bmod ((P_1)^{2^{j_1}}, \cdots, (P_m)^{2^{j_m}}), \qquad i = 1, \cdots, n,$$

and

$$A_j \equiv a_j \bmod I, \qquad j = 1, \cdots, r,$$

and $\bar{W} \in R_{r \times n}$ with

$$\bar{U}\bar{W} \equiv E \bmod ((P_1)^{2^{j_1}}, \cdots, (P_m)^{2^{j_m}}),$$

where $\bar{U} = (\bar{u}_{ij}) \in R_{n\times r}$ *and*

$$\bar{u}_{ij} = \frac{\partial f_i}{\partial x_j}(A_1, \cdots, A_r), \qquad i = 1, \cdots, n, \qquad j = 1, \cdots, r.$$

Proof. We proceed by induction on m. If $m = 1$, we get $A_1, \cdots, A_r$ and $\bar{W}$ from Theorem 5.1. with $I = (P_1)$ and $S = j_1$. So let $m > 1$ and assume that the proposition is true for $m-1$. Let $R' = R/(P_m)$; for $r \in R$ let r' denote the image of r under the canonical homomorphism from R onto R', i.e. $r' = r + (P_m)$; similarly for subsets of R and elements of $R_{k\times l}$ and $R[x_1, \cdots, x_r]$. Then

$$I' = I/(P_m) = (P_1', \cdots, P_{m-1}') \trianglelefteq R'.$$

$$R/I \simeq R'/I' \tag{24}$$

by (23), and therefore

$$f_i'(a_1', \cdots, a_r') \equiv 0 \bmod I', \qquad i = 1, \cdots, n,$$

and

$$U'W' \equiv E' \bmod I'.$$

So, by the induction hypothesis for R', I', $f_1', \cdots, f_n'$, $a_1', \cdots, a_r'$, W', there exist $\hat{A}_1, \cdots, \hat{A}_r \in R'$ with

$$f_i'(\hat{A}_1, \cdots, \hat{A}_r) \equiv 0 \bmod ((P_1')^{2i_1}, \cdots, (P_{m-1}')^{2i_{m-1}}), \qquad i = 1, \cdots, n, \tag{25}$$

and

$$\hat{A}_j \equiv a_j' \bmod I', \qquad j = 1, \cdots, r, \tag{26}$$

and $\hat{W} \in R'_{r\times n}$ with

$$\hat{U}\hat{W} \equiv E' \bmod ((P_1')^{2i_1}, \cdots, (P_{m-1}')^{2i_{m-1}}) \tag{27}$$

where $\hat{U} = (\hat{u}_{ij}) \in R'_{n\times r}$ and

$$\hat{u}_{ij} = \frac{\partial f_i'}{\partial x_j}(\hat{A}_1, \cdots, \hat{A}_r) \in R', \qquad i = 1, \cdots, n, \quad j = 1, \cdots, r.$$

Now we choose $\tilde{A}_1, \cdots, \tilde{A}_r \in R$ such that

$$\tilde{A}_j' = \hat{A}_j, \qquad j = 1, \cdots, r, \tag{28}$$

and $\tilde{W} \in R_{r\times n}$ such that

$$\tilde{W}' = \hat{W}. \tag{29}$$

By (23), $R'/((P_1')^{2i_1}, \cdots, (P_{m-1}')^{2i_{m-1}}) \simeq R/((P_1)^{2i_1}, \cdots, (P_{m-1})^{2i_{m-1}}, P_m)$, and so by (25) and (28)

$$f_i(\tilde{A}_1, \cdots, \tilde{A}_r) \equiv 0 \bmod ((P_1)^{2i_1}, \cdots, (P_{m-1})^{2i_{m-1}}, P_m), \quad i = 1, \cdots, n, \tag{30}$$

and by (27) and (29)

$$\tilde{U}\tilde{W} \equiv E \bmod ((P_1)^{2i_1}, \cdots, (P_{m-1})^{2i_{m-1}}, P_m), \tag{31}$$

where $\tilde{U} = (\tilde{u}_{ij}) \in R_{n\times r}$ and

$$\tilde{u}_{ij} = \frac{\partial f_i}{\partial x_j}(\tilde{A}_1, \cdots, \tilde{A}_r), \qquad i = 1, \cdots, n, \quad j = 1, \cdots, r.$$

By (26), (28) and (24)

$$\tilde{A}_j \equiv a_j \bmod I, \qquad j = 1, \cdots, r. \tag{32}$$

Let $R'' = R/((P_1)^{2i_1}, \cdots, (P_{m-1})^{2i_{m-1}})$; for $r \in R$ let r'' denote the image of r under the canonical homomorphism from R onto R'', similarly for subsets of R and elements of $R_{k\times l}$ and $R[x_1, \cdots, x_r]$. Let

$$I^* = (P_m'') \trianglelefteq R''. \tag{33}$$

$$R/((P_1)^{2i_1}, \cdots, (P_{m-1})^{2i_{m-1}}, P_m) \simeq R''/I^* \tag{34}$$

by (23), and therefore, by (30),

$$f_i''(\tilde{A}_1'', \cdots, \tilde{A}_r'') \equiv 0 \bmod I^*, \qquad i = 1, \cdots, n,$$

and by (31),

$$\tilde{U}''\tilde{W}'' \equiv E'' \bmod I^*.$$

So, by Theorem 5.1, with R'' instead of R, and with I^* for I, f_i'' for f_i, $i = 1, \cdots, n$, $\tilde{A}_j''$ for a_j, $j = 1, \cdots, r$, and j_m for S, there exist $\hat{\hat{A}}_1, \cdots, \hat{\hat{A}}_r \in R''$ with

$$f_i''(\hat{\hat{A}}_1, \cdots, \hat{\hat{A}}_r) \equiv 0 \bmod (I^*)^{2i_m} \tag{35}$$

and

$$\hat{\hat{A}}_j \equiv \tilde{A}_j'' \bmod I^*, \qquad j = 1, \cdots, r, \tag{36}$$

and

$$\hat{\hat{W}} \in R''_{r\times n} \quad \text{with } \hat{\hat{U}}\hat{\hat{W}} \equiv E'' \bmod (I^*)^{2i_m}, \tag{37}$$

where $\hat{\hat{U}} = (\hat{\hat{u}}_{ij}) \in R''_{n\times r}$ and

$$\hat{\hat{u}}_{ij} = \frac{\partial f_i''}{\partial x_j}(\hat{\hat{A}}_1, \cdots, \hat{\hat{A}}_r) \in R'', \qquad i = 1, \cdots, n, \quad j = 1, \cdots, r.$$

We choose $A_1, \cdots, A_r \in R$ such that

$$A_j'' = \hat{\hat{A}}_j, \qquad j = 1, \cdots, r, \tag{38}$$

and $\bar{W} \in R_{r\times n}$ such that

$$\bar{W}'' = \hat{\hat{W}}. \tag{39}$$

By (33)

$$(I^*)^{2i_m} = (P_m'')^{2i_m},$$

and by (23)

$$R''/(I^*)^{2i_m} \simeq R/((P_1)^{2i_1}, \cdots, (P_m)^{2i_m}),$$

and hence by (35) and (38)

$$f_i(A_1, \cdots, A_r) \equiv 0 \bmod ((P_1)^{2i_1}, \cdots, (P_m)^{2i_m}), \qquad i = 1, \cdots, n,$$

and by (37) and (39)

$$\bar{U}\bar{W} \equiv E \bmod ((P_1)^{2i_1}, \cdots, (P_m)^{2i_m}).$$

By (33) $I^* \subseteq I'' = (P_1'', \cdots, P_m'') \trianglelefteq R''$, so by (36)

$$\hat{\hat{A}}_j \equiv \tilde{A}_j'' \bmod I'', \qquad j = 1, \cdots, r. \tag{40}$$

By (23) $R/I \simeq R''/I''$, so by (38) and (40) $A_j \equiv \tilde{A}_j \bmod I$, $j = 1, \cdots, r$. Finally, by (32) $A_j \equiv a_j \bmod I$, $j = 1, \cdots, r$. □

According to our custom, Theorem 5.2. is followed by an abstract algorithm,

ALGORITHM M (Musser's lifting).

Inputs: $P_1, \cdots, P_m$, finite subsets of R, with $I = (P_1, \cdots, P_m) \trianglelefteq R$;
$f_1, \cdots, f_n \in R[x_1, \cdots, x_r]$, $r \geqq 1$;
$a_1, \cdots, a_r \in R$ with
$$f_i(a_1, \cdots, a_r) \equiv 0 \bmod I,\ i = 1, \cdots, n;$$
$W \in R_{r\times n}$ with
$$UW \equiv E \bmod I,$$
where $U = (u_{ij}) \in R_{n\times r}$
and $u_{ij} = (\partial f_i/\partial x_j)(a_1, \cdots, a_r)$, $i = 1, \cdots, n$, $j = 1, \cdots, r$;
$j_1, \cdots, j_m$, nonnegative integers.

Outputs: $A_1, \cdots, A_r \in R$ with
$$f_i(A_1, \cdots, A_r) \equiv 0 \bmod ((P_1)^{2^{j_1}}, \cdots, (P_m)^{2^{j_m}}),\ i = 1, \cdots, n$$
and $A_j \equiv a_j \bmod I$, $j = 1, \cdots, r$;
$\bar{W} \in R_{r\times n}$ with
$$\bar{U}\bar{W} \equiv E \bmod ((P_1)^{2^{j_1}}, \cdots, (P_m)^{2^{j_m}}),$$
where $\bar{U} = (\bar{u}_{ij}) \in R_{n\times r}$
and $\bar{u}_{ij} = (\partial f_i/\partial x_j)(A_1, \cdots, A_r)$, $i = 1, \cdots, n$, $j = 1, \cdots, r$.

(1) [$m = 1$.] if $m = 1$, apply Algorithm Z with inputs
$P_1, f_1, \cdots, f_n, a_1, \cdots, a_r, W, j_1$; obtain outputs
$A_1, \cdots, A_r$ and $\bar{W}$; exit.

(2) [$m > 1$; recursive call for $P_1, \cdots, P_{m-1}$.]
set $R' \leftarrow R/(P_m)$;
let h_1 denote the canonical homomorphism from R onto R'; apply this algorithm recursively, with R' instead of R, to inputs $h_1(P_1), \cdots, h_1(P_{m-1})$, $h_1(f_1), \cdots, h_1(f_n)$, $h_1(a_1), \cdots, h_1(a_r)$, $h_1(W), j_1, \cdots, j_{m-1}$; obtain outputs $\hat{A}_1, \cdots, \hat{A}_r \in R'$, $\hat{W} \in R'_{r\times n}$;
choose
$\tilde{A}_1, \cdots, \tilde{A}_r \in R$ with $h_1(\tilde{A}_j) = \hat{A}_j$, $j = 1, \cdots, r$, and $\tilde{W} \in R_{r\times n}$ with $h_1(\tilde{W}) = \hat{W}$.

(3) [Algorithm Z for P_m.]
set $R'' \leftarrow R/((P_1)^{2^{j_1}}, \cdots, (P_{m-1})^{2^{j_{m-1}}})$;
let h_2 denote the canonical homomorphism from R onto R''; apply Algorithm Z (with R'' instead of R), to inputs $h_2(P_m), h_2(f_1), \cdots, h_2(f_n), h_2(\tilde{A}_1), \cdots, h_2(\tilde{A}_r)$, $h_2(\tilde{W}), j_m$;
obtain outputs $\hat{\hat{A}}_1, \cdots, \hat{\hat{A}}_r \in R''$, $\hat{\hat{W}} \in R''_{r\times n}$;
choose
$A_1, \cdots, A_r \in R$ with $h_2(A_j) = \hat{\hat{A}}_j$, $j = 1, \cdots, r$,
and $\bar{W} \in R_{r\times n}$ with $h_2(\bar{W}) = \hat{\hat{W}}$;
exit

It should be noted that Algorithm H (Hensel's linear lifting, § 4) can easily be modified such that it also lifts (linearly) a right inverse $W^{(T)}$ of $U^{(T)}$ mod I^T, and that in Algorithm M this algorithm could be used instead of Algorithm Z, thereby achieving iterated linear lifting. Then all essential operations are done mod I (instead of mod I^{2^S}), and this may turn out to be more efficient for some concrete lifting algorithms; see [M&Y74].

6. Application problems. In this last section we will specialize the abstract lifting algorithms into some concrete ones. We don't want to go into the details; several sophisticated implementations are described elsewhere ([Mus71, 75], [Yun74], [W&R75]). We rather want to point out some general problems which arise.

The most commonly used domain R in which the concrete lifting algorithms work is $\mathscr{Z}[y_1, \cdots, y_n]$, $n \geqq 1$, the ring of integral polynomials in n variables. The most commonly used lifting algorithm is Algorithm M (§ 5). The starting ideal $I = (P_1, \cdots, P_m)$ is chosen such that $R/I \simeq GF(p)[y_n]$, because polynomial operations can be carried out rather efficiently over $GF(p)$. There are different ways to choose $P_1, \cdots, P_m$ in Algorithm M to achieve this. Musser in [Mus71, 75] proposed $P_1 = \{p\}$, $p \in \mathscr{Z}$ prime, $P_i = \{y_{i-1} - a_{i-1}\}$, $a_{i-1} \in \mathscr{Z}$, $i = 2, \cdots, n$. Yun in [Yun74] and Wang and Rothschild in [W&R75] use $P_1 = \{p\}$, $P_2 = \{y_1 - a_1, \cdots, y_{n-1} - a_{n-1}\}$. In both cases $p, a_1, \cdots, a_{n-1}$ are chosen such that I is "lucky" in a sense which has to be specified according to the problem to be solved; for factorization it means that the polynomial C to be factored has the same degree in y_n as its image C' mod I, and that C' is squarefree if C is. So the choice of I depends on the particular instance of the problem.

The final aim of computation in a domain R is in general a solution over R, not a solution modulo some ideal J. So one needs a method to obtain a solution over R, if it exists, from a solution mod J for a suitable J. It is intuitively clear that this implies that, if $A_1^*, \cdots, A_r^*$ is a solution over R, and $A_1, \cdots, A_r$ is a solution mod J obtained by a lifting algorithm, then

$$A_j^* \equiv A_j \bmod J, \qquad j = 1, \cdots, r. \tag{41}$$

This is by no means obvious, as the following example shows.

We take the factorization problem for univariate integral polynomials. So $R = \mathscr{Z}[y]$, $I = (p)$, $p \in \mathscr{Z}$ prime. Let

$$C = y^3 - 5y^2 + 4y - 20 = (y^2 + 4)(y - 5),$$

so

$$A_1^* = y^2 + 4, \qquad A_2^* = y - 5.$$

We want to solve $f = x_1 x_2 - C = 0$ over R and choose $p = 3$. A solution mod $I = (3)$ is

$$a_1^{(1)} = y^2 + 1, \qquad a_2^{(1)} = y + 1,$$

and we have

$$f(a_1^{(1)}, a_2^{(1)}) = a_1^{(1)} a_2^{(1)} - C = (2y^2 - y + 7) \cdot 3.$$

So, for a solution $a_1^{(2)} = a_1^{(1)} + b_1 \cdot 3$, $a_2^{(2)} = a_2^{(1)} + b_2 \cdot 3$ mod (9), we have to solve

$$(2y^2 - y + 7) + b_1(y + 1) + b_2(y^2 + 1) \equiv 0 \bmod (3). \tag{42}$$

One possible solution is

$$b_1 = y^2 - 1, \qquad b_2 = -y,$$

and so

$$a_1^{(2)} = 4y^2 - 2, \qquad a_2^{(2)} = -2y + 1.$$

We have

$$f(a_1^{(2)}, a_2^{(2)}) = (-y^3 + y^2 + 2) \cdot 9 \equiv 0 \bmod (9),$$

but

$$a_1^{(2)} \not\equiv A_1^* \bmod (9), \qquad a_2^{(2)} \not\equiv A_2^* \bmod (9)!$$

Another solution of (42) is

$$b_1 = 1, \qquad b_2 = 1;$$

for this we get

$$a_1^{(2)} = y^2 + 4, \qquad a_2^{(2)} = y + 4.$$

Now we have

$$a_1^{(2)} \equiv A_1^* \bmod (9), \qquad a_2^{(2)} \equiv A_2^* \bmod (9).$$

The reason for this flaw is that (42), or more generally (10), does in general not have a unique solution, and we need a way to uniquely choose a particular solution which leads the lifting to a solution which satisfies (41).

If there exists a solution $A_1^*, \cdots, A_r^*$ over R then one usually can specify an ideal J such that A_j^* is a canonical representative of its residue class $A_j^* + J \in R/J$, for example, if $R = \mathscr{Z}[y_1, \cdots, y_n]$, by giving bounds on the degrees and norms of the A_j^*: if it is known that $|A_j^*|_\infty < q$ and $\deg_{y_i}(A_j^*) < n_i$, $i = 1, \cdots, n-1$, then we may choose $I = (p, y_1 - a_1, \cdots, y_{n-1} - a_{n-1})$ and $J = ((p^k), (y_1 - a_1)^{n_1}, \cdots, (y_{n-1} - a_{n-1})^{n_{n-1}})$, where k is such that $p^k > 2q$, or $J = ((p^k), (y_1 - a_1, \cdots, y_{n-1} - a_{n-1})^s)$ where $s = \max\{n_1, \cdots, n_{n-1}\}$. It is easy to see that during the lifting process the lifted solutions can be chosen in canonical form modulo the ideal which is currently used, so that by (41)

$$A_j^* = A_j, \qquad j = 1, \cdots, r,$$

holds.

So, if it is known that a solution $A_1^*, \cdots, A_r^*$ over R exists, two points have to be regarded in order to compute it by a lifting algorithm: first, an ideal J has to be specified such that the A_j^* are canonical representatives of $A_j^* + J \in R/J$, and $K \subset J \subset I$, where I is the (lucky) starting ideal and K is the final ideal of a lifting algorithm, and second, one must be able to solve the linear congruences at each lifting step such that finally (41) holds.

The situation becomes more difficult if a solution modulo the starting ideal I exists and the lifting can be carried out, but the existence of a solution over R is not guaranteed, as it is for factorization. Of course each polynomial $C \in R = \mathscr{Z}[y_1, \cdots, y_n]$ has a complete factorization (which is unique up to units), but the number of factors is not known in advance. For the lifting, these factors have to be given as solutions of an equation, and in that equation the number of variables is given by the number of irreducible factors mod I which may be greater than the number of irreducible factors over R. So usually (41) will not hold. The equivalent of (41) in that case is that the lifted solution $A_1, \cdots, A_r$ constitutes a complete factorization of C mod J, i.e. (see [Mus71]):

(a) $C \equiv cA_1 \cdots A_r \bmod J$ for some $c \in \mathscr{Z}[y_1, \cdots, y_{n-1}]$;

(b) for every factorization $C \equiv AB \bmod J$ such that $\deg_{y_n}(C) = \deg_{y_n}(A) + \deg_{y_n}(B)$, there exists a unique $H = \{A_{i_1}, \cdots, A_{i_s}\} \subset \{A_1, \cdots, A_r\}$ such that $A \equiv aA_{i_1} \cdots A_{i_s} \bmod J$ for some $a \in \mathscr{Z}[y_1, \cdots, y_{n-1}]$.

(c) The leading coefficient with respect to y_n of each A_j is not a zero divisor mod J.

If this holds, the true factors of C can be obtained by trying all products of A_j's.

So one has to specify a solution for the congruence arising from (10) (or its equivalent in the quadratic lifting) which leads to a complete factorization mod J. Let us look somewhat closer at this particular problem.

If $a_1, \cdots, a_r$ are the irreducible factors of the squarefree polynomial $C \in R = \mathscr{Z}[y_1, \cdots, y_n] \bmod I = (p, y_1 - z_1, \cdots, y_{n-1} - z_{n-1}), z_1, \cdots, z_{n-1} \in \mathscr{Z}$, where I is lucky, then $a_1, \cdots, a_r$ are solutions mod I of the equation

$$f = x_1 \cdots x_r - C = 0.$$

To lift this solution we need a right inverse $W \in R_{r\times 1}$ of $U \in R_{1\times r} \bmod I$, where

$$U = (u_1, \cdots, u_r) \quad \text{and} \quad u_j = \frac{\partial f}{\partial x_j}(a_1, \cdots, a_r) = \prod_{\substack{i=1 \\ i\neq j}}^{r} a_i.$$

Since $a_1, \cdots, a_r$ are pairwise relatively prime mod I,

$$\gcd(u_1, \cdots, u_r) \equiv 1 \bmod I,$$

and hence $W = (w_1, \cdots, w_r)^*$ exists with $UW \equiv 1 \bmod I$, so that the input assertions for the lifting algorithms hold. We now need a way to modify Algorithm Z in such a way that it (or Algorithm M) leads to a complete factorization modulo the final ideal. For each lifting step we have to solve (see (18))

$$\bar{v}_k + U^{(S)}\bar{b}_k \equiv 0 \bmod I^{2^S}, \qquad k = 1, \cdots, m, \tag{43}$$

where

$$U^{(S)} = (u_j^{(S)}) \quad \text{and} \quad u_j^{(S)} = \prod_{\substack{i=1 \\ i\neq j}}^{r} a_i^{(S)}$$

and the $a_j^{(S)}$ are solutions (the factors) mod I^{2^S}.

In Algorithm Z we used

$$\bar{b}_k = -W^{(S)}\bar{v}_k,$$

where $W^{(S)}$ was a right inverse of $U^{(S)} \bmod I^{2^S}$ obtained by lifting. Let $\bar{b}_k = (b_{1k}, \cdots, b_{rk})^*$; then $b_{jk} \in \mathscr{Z}[y_1, \cdots, y_n]$, $j = 1, \cdots, r$. Suppose that the leading coefficient of $a_j^{(S)}$ with respect to y_n is a unit mod I^{2^S}, $j = 1, \cdots, r$; let then q_{jk}, $b'_{jk} \in \mathscr{Z}[y_1, \cdots, y_n]$, $j = 1, \cdots, r-1$, be defined by

$$b_{jk} \equiv q_{jk}a_j^{(S)} + b'_{jk} \bmod I^{2^S},$$

$$b'_{jk} = 0 \quad \text{or} \quad \deg_{y_n}(b'_{jk}) < \deg_{y_n}(a_j^{(S)}),$$

and

$$b'_{rk} = \left(\sum_{j=1}^{r-1} q_{jk}\right) a_r^{(S)} + b_{rk}.$$

Then $\bar{b}'_k = (b'_{1k}, \cdots, b'_{rk})^*$ is also a solution of (43). If this is used,

$$\operatorname{ldcf}_{y_n}(a_j^{(S+1)}) = \operatorname{ldcf}_{y_n}(a_j^{(S)}) = \cdots = \operatorname{ldcf}_{y_n}(a_j), \qquad j = 1, \cdots, r-1.$$

Since mod I each nonzero element is a unit and since each unit mod I is a unit mod I^{2^S}, one can always choose the solutions $\bar{b}'_k$ as just described, i.e. with $\deg_{y_n}(b'_{jk}) < \deg_{y_n}(a_j)$. Furthermore, $\bar{b}'_k$ is uniquely determined mod I^{2^S} by this degree condition. This is a rather straightforward generalization of the classical lifting to more than 2 factors, and generalizing Musser's proofs in [Mus71] (see also [Mus75]) one can show that

this choice of solutions of (43) leads to a complete factorization modulo the final ideal, in both Algorithms Z and M.

In the case where $R = \mathscr{Z}[y_1, \cdots, y_n]$, such degree constraints on the solutions of (43) seem to be the general method to lead the lifting algorithms to solutions modulo some ideal J from which solutions over R can be obtained.

As a final remark let us note that for the validity of the lifting theorems and algorithms we required an input assertion which may be somewhat too strong: we assumed that each system of linear equations with coefficient matrix U has a solution mod I; it would be sufficient to require that each such system which actually shows up during the lifting has a solution mod I. (To include this would make the formulation of Theorem 4.2. longer than its proof.)

Acknowledgments. I want to express my sincere gratitude to Professor R. Loos for his guidance and encouragement throughout this work. I also want to thank Professor G. E. Collins for helpful discussions.

REFERENCES

[Lew69] D. J. LEWIS, *Diophantine equations: p-adic methods*, in Studies in Number Theory, W. J. LeVeque, ed., Math. Assoc. Amer., 1969, pp. 25–75.

[Lün73] H. LÜNEBURG, *Einführung in die Algebra*, Springer Hochschultext, Springer-Verlag, Berlin, 1973.

[Mus71] D. R. MUSSER, *Algorithms for polynomial factorization*, Technical Rep. 134 (Ph.D. Thesis), Computer Sciences Dept., University of Wisconsin, Madison, WI, 1971.

[Mus75] ———, *Multivariate polynomial factorization*, J. Assoc. Comput. Mach., 22 (1975), pp. 291–308.

[M&Y74] A. MIOLA AND D. Y. Y. YUN, *The computational aspects of Hensel-type univariate polynomial greatest common divisor algorithms*, in Proceedings of EUROSAM'74, Stockholm, 1974, pp. 46–54.

[W&R75] P. S. WANG AND L. P. ROTHSCHILD, *Factoring multivariate polynomials over the integers*, Math. Comp., 29 (1975), pp. 935–950.

[Yun74a] D. Y. Y. YUN, *The Hensel lemma in algebraic manipulation*, Ph.D. Thesis, Dept. Mathematics, Project MAC Report TR-138, Massachusetts Institute of Technology, Cambridge, MA, 1974.

[Yun74b] ———, *A p-adic division with remainder algorithm*, ACM SIGSAM Bulletin, 8 (1974) (Issue 32), pp. 27–32.

[Yun75] ———, *Hensel meets Newton-algebraic constructions in an analytic setting*, in Analytic Computational Complexity, Proc. CMU Symposium, J. Traub, ed., Academic Press, New York, 1975.

[Zas69] H. ZASSENHAUS, *On Hensel Factorization, I*, J. Number Theory, 1 (1969), pp. 291–311.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0001 $01.25/0

ON CERTAIN POLYNOMIAL-TIME TRUTH-TABLE REDUCIBILITIES OF COMPLETE SETS TO SPARSE SETS*

YAACOV YESHA†

Abstract. Let Σ be a finite alphabet. A set $S \subset \Sigma^*$ is called *sparse* if the number of members of S having length at most n is bounded above by a polynomial in n. Let $\leqq_m^P$ denote polynomial-time many-one reducibility, and let $\leqq_{bptt}^P$ denote the more general polynomial-time bounded positive truth-table reducibility. We prove: (1) $\leqq_{bptt}^P$ reducibility of a coNP-hard set to a sparse set implies NP = P, and (2) $\leqq_{bptt}^P$ reducibility of an NP-hard set to a sparse set which is itself in NP implies NP = P. (1) generalizes Fortune's result [F]. He proved it for the case of $\leqq_m^P$ reducibility. (2), for the case of $\leqq_m^P$ reducibility, was proved by Mahaney [M], even without assuming that the sparse set itself is in NP. Our results imply that if a coNP-hard set is a finite union of sets which are $\leqq_m^P$ reducible to sparse sets, then NP = P. We then investigate a certain nonpositive polynomial-time truth-table reducibility of NP-hard sets to sparse sets, and obtain new results regarding the structure of sets hard for NP or for ETIME. Finally, we investigate the possibility of existence of sets in NP − coNP which are $\leqq_m^P$ reducible to sparse sets. Some of our techniques involve generalizations of and variations on the techniques of Berman [B], Fortune [F] and Mahaney [M].

Key words. NP-complete sets, sparse sets, polynomial-time reducibility, truth-table reducibility

1. Introduction. This work is concerned with the implications of the existence of certain polynomial-time truth-table reducibilities of NP-complete and other sets to h-sparse sets (to be defined below). The questions resolved have natural interpretations in terms of the structure of NP-hard and other sets. For example, we show that if NP ≠ P, no NP-hard set A can be "very close" to a set B in P in the sense that the symmetric difference of A and B is "very small", namely, is h-sparse (see definition below) for $h(n) = O(\log \log (n))$.

Let Σ be a finite alphabet. A set S, $S \subset \Sigma^*$, is called *h-sparse* if its *census function* c_S, defined by $c_S(n) = \|\Sigma^{(n)} \cap S\|$ (where for any set S, $\|S\|$ denotes its cardinality, and $\Sigma^{(n)}$ is the set of all strings over Σ of length at most n) satisfies $c_S(n) = O(h(n))$. In particular, S is called *sparse* if there is some polynomial h such that S is h-sparse. Several recent papers deal with the structure of NP-hard and other related sets. From [L], [S] we know that a set which is polynomial-time Turing reducible ($\leqq_T^P$) to *arbitrarily* sparse sets (according to a different notion of sparseness) is in P (first proved by Lynch [L] for $\leqq_m^P$, then extended by Solovay [S] to $\leqq_T^P$). For A NP-complete, coNP-complete, or PSPACE-complete, $A \leqq_m^P S$ (A polynomial-time many-one reducible to S), where S is sparse, implies $A \in$ P ([B], [M], [F], [MP], see also [C1] for some generalizations). Related questions are treated in Landweber, Lipton and Robertson [LLR] and in Karp and Lipton [KL]. We now sketch our main results:

1) As pointed out in [M], the question whether $A \leqq_T^P S$ for A NP-complete and S sparse implies NP = P is an important open question. Motivated by this question, we treat some special cases. In what follows let S be sparse. We prove that if A is coNP-complete or PSPACE-complete and $A \leqq_{bptt}^P S$ (A is polynomial-time bounded positive truth-table reducible to S) then $A \in$ P. We then prove that if A is NP-complete, $A \leqq_{bptt}^P S$, and S itself is in NP then $A \in$ P.

We also show that $A \leqq_{bptt}^P S$ where S is sparse if and only if $A = \bigcup_{i=1}^t A_i$, $A_i \leqq_m^P S_i$ ($1 \leqq i \leqq t$) and the S_i are sparse. Thus, if a coNP-complete or PSPACE-complete set A is a finite union of sets which are polynomial-time many-one reducible to sparse

* Received by the editors July 9, 1981, and in revised form June 10, 1982. This work was supported in part by Natural Science and Engineering Research Council grants A7671 and A8651.

† Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A7.

sets, then $A \in \mathrm{P}$. For example, take $A = \Sigma^* \# S \cup S \# \Sigma^*$. This case already generalizes polynomial-time many-one reducibility to sparse sets, since A does not seem to be polynomial-time many-one reducible to a sparse set, but $A \leqq^P_{bptt} S$.

2) We also investigate some polynomial-time bounded truth-table reducibilities which are not necessarily positive. We prove: If A is coNP-complete and for all $x \in \Sigma^*$,

$$x \in A \Leftrightarrow f(x) \in S_1 \wedge g(x) \notin S_2,$$

where f, g are computable in deterministic polynomial-time, and S_1, S_2 are "sparse enough" (e.g., c_{S_1}, c_{S_2} being $O(\log \log (n))$), then $A \in \mathrm{P}$. This result has the following consequences which we find interesting:

2-1) It is known [B], [MP] that if an NP-hard set A can be recognized by an algorithm which runs in polynomial-time on all inputs except for inputs from some sparse set S, then NP = P. The hypothesis is equivalent to the existence of a recognizer for A which always runs in polynomial-time, but for inputs from S halts without giving any answer. So, in particular S is in P.

We now ask ourselves what happens if we weaken the hypothesis in the sense that there is an algorithm for an NP-hard set which always runs in polynomial-time, for each input in $\Sigma^* - S$ gives the correct answer, but for each input in S gives a wrong answer. Note that in this case S is not necessarily in P! Using 2) above, we show that if S is sparse enough ($c_S(n) = O(\log \log (n))$ is sufficient) we can conclude NP = P. One way to interpret the hypothesis is by saying that A is "close enough" to a set in P, namely, the set $B = \{x \in \Sigma^* | \text{the above polynomial algorithm answers "yes" on } x\}$, where "close enough" means that S, the symmetric difference of A and B has a sufficiently slowly growing census function. Using a result of Meyer, appearing in [KL], we obtain as a corollary that if D is ETIME-hard (with respect to polynomial-time many-one reducibility) there is no polynomial-time algorithm which recognizes D correctly outside a set S with census $O(\log \log (n))$.

2-2) Let ISO be the set of all pairs (G_1, G_2) where G_1, G_2 are graphs and G_1 is isomorphic to G_2. Let INTFAC be the set of all triples (m, a, b) where $1 < a \leqq b \leqq m$ and m has a prime factor between a and b. ISO and INTFAC are in NP, but not known to be in P. [M] and [MP] leave open the question whether $A \leqq^P_m S$ where A is ISO or INTFAC and S is sparse imply $A \in \mathrm{P}$. Using 2) above we show that for S sparse enough (for instance $c_S(n) = O(\log \log (n))$) and A = INTFAC we can conclude $A \in \mathrm{P}$. For A = ISO we can only conclude the existence of a sub-exponential algorithm under the above hypotheses.

3) [HM] raises the question whether NP $\neq$ P implies the existence of sparse sets in NP − P. We show that the existence of a set in NP − coNP which is polynomial-time many-one reducible to a sparse set implies ESPACE $\neq$ NETIME.

The main tools used in obtaining the results mentioned in 1) and 2) above are generalizations of and variations on Fortune's tree searching method [F], [M].

2. Preliminaries and notation. We assume familiarity with the classes P, NP, coNP, [C], [K] and PSPACE = NPSPACE. We also assume familiarity with [M] and [MP]. For standard definitions and results from computational complexity, see [HU].

Let

$$\mathrm{ETIME} = \bigcup_{c>0} \mathrm{DTIME}(2^{cn}), \qquad \mathrm{NETIME} = \bigcup_{c>0} \mathrm{NTIME}(2^{cn}),$$

$$\mathrm{ESPACE} = \bigcup_{c>0} \mathrm{DSPACE}(2^{cn}).$$

For each set $A \subset \Sigma^*$, A^c denotes $\Sigma^* - A$. We will need the following sets, which are in NP:

SAT—the set of all satisfiable Boolean formulas.

ISO—the set of all pairs (G_1, G_2) of isomorphic graphs.

INTFAC $= \{(m, a, b) | 1 < a \leqq b \leqq m$ and m has a prime factor between a and $b\}$.

INTFAC $\in$ P if and only if there is an integer factoring algorithm which runs in polynomial-time [MP].

For any set C let χ_C denote its characteristic function.

We need the following types of polynomial-time reducibilities:

1) $\leqq^P_m$-*polynomial-time many-one reducibility* [K].
2) $\leqq^P_T$-*polynomial-time Turing reducibility* [C].
3) Various types of polynomial-time truth-table reducibilities, essentially defined in Ladner, Lynch and Selman [LLS].

DEFINITION [LLS]. Let $A, B \subset \Sigma^*$. Then

$$A \leqq^P_{tt} B \ (A \text{ is polynomial-time truth-table reducible to } B)$$

if the following hold:

1) There exists a polynomial-time mapping g, called a *tt-condition generator*, from Σ^* into $\Delta^* \# (\# \Sigma^*)^*$, where Δ is a fixed alphabet and $\# \notin \Sigma \cup \Delta$. The output of the mapping is called a *tt-condition*.

2) There exists a polynomial-time mapping e from $\Delta^* \# \{0, 1\}^*$ into $\{0, 1\}$ called a *tt-condition evaluator.*

3) $(\forall x \in \Sigma^*)(x \in A \Leftrightarrow g(x)$ is *e-satisfied* by $B)$, where "$g(x)$ is e-satisfied by B" means that if $g(x) = w \# \# \alpha_1 \# \alpha_2 \cdots \# \alpha_k$ then $e(w \# \chi_B(\alpha_1)\chi_B(\alpha_2) \cdots \chi_B(\alpha_k)) = 1$.

Thus $w \in \Delta^*$ represents a Boolean function which is applied to the variables $x_1, x_2, \cdots, x_k$ where $x_i = 1$ if $\alpha_i \in B$, and 0 otherwise. w and the α_i's depend on x by g.

DEFINITION. A *tt*-condition generator g is *bounded* if there exists a constant k such that for any $x \in \Sigma^*$, $g(x)$ contains at most k #'s. This means that all the Boolean functions obtained have at most k variables.

A *tt*-condition evaluator e is *positive* if it has the property:

$$[e(w \# \sigma_1 \sigma_2 \cdots \sigma_k) = 1 \wedge (\sigma_i = 1 \Rightarrow \tau_i = 1)] \Rightarrow [e(w \# \tau_1 \tau_2 \cdots \tau_k) = 1].$$

The meaning is that the Boolean function represented by w is positive (i.e., monotone increasing).

DEFINITION.

$A \leqq^P_{btt} B$ (A is polynomial-time bounded truth-table reducible to B) if the generator g is bounded.

$A \leqq^P_{ptt} B$ (A is polynomial-time positive truth-table reducible to B) if the evaluator e is positive.

$A \leqq^P_{bptt} B$ (A is polynomial-time bounded positive truth-table reducible to B) if g is bounded and e is positive.

We are now going to discuss self-reducibility [M], [MP], [KL].

DEFINITION. $A \subset \Sigma^*$ is called *disjunctively-self-reducible* if:

1) There exists $W \subset \Sigma^*$, $W \in$ P such that $A \cap W \in$ P.
2) For each $x \notin W$ a list $L(x) = (x_1, x_2, \cdots, x_k)$ can be computed in polynomial-time, such that $|x_i| < |x|$ $(1 \leqq i \leqq k)$.
3) For $X \notin W$, $x \in A \Leftrightarrow \vee_{i=1}^{k} (x_i \in A)$, where $L(x) = (x_1, x_2, \cdots, x_k)$.

DEFINITION. The *tree of self-reductions* of x is the tree with x as a root, the sons of each node $y \notin W$ being the members of $L(y)$ from left to right, and the leaves being members of W.

A is called *conjunctively-self-reducible* if in 3) above $\wedge_{i=1}^{k}(x_i \in A)$ replaces $\vee_{i=1}^{k}(x_i \in A)$. Clearly A is disjunctively-self-reducible if and only if its complement A^c is conjunctively-self-reducible, L, W and the tree of self-reductions being the same.

A is called *strongly disjunctively (conjunctively)-self-reducible* if it is disjunctively (conjunctively)-self-reducible and there exists a constant d such that for every $x \notin W$ $L(x)$ contains at most d members.

It is known [F], [M], [MP], [KL] that SAT and INTFAC are strongly disjunctively-self-reducible, and there exists a set which is polynomial-time many-one equivalent to ISO which is disjunctively-self-reducible. For SAT take W to be the set $\{false, true\}$ of constant Boolean formulas, and for a formula $x \notin W$ if $x = \beta(x_1, x_2, \cdots, x_n)$ let $L(x) = (\beta(0, x_2, \cdots, x_n), \beta(1, x_2, \cdots, x_n))$ [F], [M], [MP].

We know the following results [F], [M], [MP]:

1) If $A \leqq_m^P S$ where S is sparse and A is conjunctively-self-reducible, then $A \in \mathrm{P}$. In particular (Fortune's theorem [F]): A being coNP-complete implies $\mathrm{NP} = \mathrm{P}$.

2) If $A \leqq_m^P S$ where S is sparse then:

i) A NP-complete $\Rightarrow \mathrm{NP} = \mathrm{P}$ (Mahaney's theorem [M]).

ii) A PSPACE-complete $\Rightarrow \mathrm{PSPACE} = \mathrm{P}$ (Fortune [F], see also Meyer and Paterson [MP]). Cook [C1] gives generalizations to h-sparse sets where h is not necessarily a polynomial.

We also note that P. Berman [B] first proved a special case of the above results, motivated by a conjecture by L. Berman and Hartmanis [BH]. His technique provided a basis for the later results of [F], [MP], [M]. In particular, he proved that if $\mathrm{NP} \neq \mathrm{P}$, no set over a one-letter alphabet can be NP-hard.

3. Polynomial-time bounded positive truth-table reducibility to sparse sets. First we show that $A \leqq_{bptt}^P S$ where S is sparse implies a particularly simple way of expressing this reducibility.

LEMMA 3.1. *The following are equivalent*:

(1) $A \leqq_{bptt}^P S$, *where* S *is sparse.*

(2) *There exist* t, *a sparse set* S_1 *and* f_i *computable in deterministic polynomial-time* $(1 \leqq i \leqq t)$ *such that for all* $x \in \Sigma^*$,

$$x \in A \Leftrightarrow \bigvee_{i=1}^{t} (f_i(x) \in S_1).$$

(3) *For some* t, $A = \bigcup_{i=1}^{t} E_i'$, $E_i' \leqq_m^P E_i$, E_i *sparse* $(1 \leqq i \leqq t)$.

Proof. (1) $\rightarrow$ (2). Let r be the bound on the number of variables appearing in any output of the *tt*-condition generator. Given x, use the *tt*-condition generator and evaluator to find the truth-table of the corresponding Boolean function β, by evaluating its value for the finitely many combinations of the variable values. Now,

$$x \in A \Leftrightarrow \beta(\chi_S(y_1), \chi_S(y_2), \cdots, \chi_S(y_k)) = 1,$$

where y_i $(1 \leqq i \leqq k \leqq r)$ are the arguments computed by the generator. Now, represent β in positive disjunctive-normal-form (positive DNF), i.e., a Boolean formula of the form $c_1 \vee c_2 \vee \cdots \vee c_q$ where each c_i is a conjunction of some of the variables (no negations allowed). Define $f_{ij}(x) = y_m$ if the jth variable in c_i is $\chi_S(y_m)$. Also $f_{ij}(x) = \mu_0$, μ_0 a fixed member of S (we assume $S \neq \varnothing$) for any $j \leqq r$ which is greater than the number of variables in c_i. Let

$$f_i(x) = f_{i1}(x) \# f_{i2}(x) \# \cdots \# f_{ir}(x) \quad \text{for } i \leqq q$$

and

$$f_i(x) = \mu_1 \# \mu_1 \# \cdots \# \mu_1 \ (r \text{ times}) \quad \text{for } q < i \leqq 2^r = t,$$

where μ_1 is a fixed element not in S. Then $x \in A \Leftrightarrow \vee_{i=1}^{t} (f_i(x) \in S_1)$, where

$$S_1 = \{w_1 \# w_2 \# \cdots \# w_r | w_i \in S (1 \leqq i \leqq r)\}.$$

(2)→(3). Let $E_i' = \{x | f_i(x) \in S_1\}$, $E_i = S_1 (1 \leqq i \leqq t)$.

(3)→(1). Let $S = \{w_1 \# w_2 \# \cdots \# w_t | w_i \in E_i (1 \leqq i \leqq t)\}$, $f_i(x) = (h_{i1}(x), \cdots, h_{it}(x))$, where for $j \neq i$, $h_{ij}(x) = \mu_i$, μ_i being a fixed member of E_i and $h_{ii}(x) = q_i(x)$, where $E_i' \leqq_m^P E_i$ by q_i. Then for all $x \in \Sigma^*$, $x \in A \Leftrightarrow \vee_{i=1}^{t} (f_i(x) \in S)$, hence $A \leqq_{bptt}^P S$. Q.E.D.

Before we prove that for a coNP-hard set D, $D \leqq_{bptt}^P S$, where S is sparse implies NP = P, we need some combinatorial lemmas, and an approximation algorithm for minimal hitting sets.

DEFINITION. A collection W of sets is called *K-compact* if there are no distinct $s_1, s_2, \cdots, s_{K+1}$ in W such that $s_i = \alpha \cup \beta_i (1 \leqq i \leqq K+1)$ where α and the β_i are pairwise disjoint.

LEMMA 3.2. *If W is a K-compact collection of sets, each of cardinality at most t, then there are at most $I(K, t) = \sum_{j=0}^{t} (t!/j!) K^{t-j}$ sets in W.*

Proof. We use induction on t. Clearly the assertion is true for $t = 0$. If $t > 0$, choose a maximal collection U of pairwise disjoint sets of W. By K-compactness, U has at most K sets in it. Let T be the union of the sets in U. T has at most Kt elements, and each nonempty set in W contains some element of T. Let

$$T = \{a_1, a_2, \cdots, a_r\} \qquad (r \leqq Kt),$$

$$W_i = \{s \in W | a_i \in s\},$$

$$W_i' = \{s - \{a_i\} | s \in W_i\} \qquad (1 \leqq i \leqq r).$$

Then $W - \{\varnothing\} = \cup_{i=1}^{r} W_i$. Also each W_i' is a K-compact collection of sets, each of cardinality at most $t-1$. By the induction hypothesis there are at most $I(K, t-1)$ sets in each W_i', hence in each W_i. Hence there are at most $1 + Kt \cdot I(K, t-1) = I(K, t)$ sets in W. Q.E.D.

LEMMA 3.3. *Suppose W is a collection of sets, each of cardinality at most t, and there exists a constant $K > 1$ and an ordering $s_1, s_2, \cdots, s_m$ of the sets in W with the following property: For each $2 \leqq j \leqq m$ there is a set of at most K elements which has nonempty intersection with each nonempty $s_i - s_j$ for $1 \leqq i < j$. Then W is $K+2$ compact, and hence has at most $I(K+2, t)$ sets by Lemma 3.2.*

Proof. If W is not $(K+2)$-compact, choose a minimal n such that $W' = \{s_1, s_2, \cdots, s_n\}$ is not $(K+2)$-compact. Then there are sets α and $\beta_i (1 \leqq i \leqq K+3)$ all pairwise disjoint and the β_i all different, and $t_1, t_2, \cdots, t_{K+3}$ in W' such that $t_i = \alpha \cup \beta_i (1 \leqq i \leqq K+3)$ and $t_{K+3} = s_n$. Now $t_i - t_{K+3} = \beta_i - \beta_{K+3} = \beta_i$ for $1 \leqq i \leqq K+2$, and at most one of the β_i for $1 \leqq i \leqq K+2$ is empty. Thus among $t_i - t_{K+3} (1 \leqq i \leqq K+2)$ there are $K+1$ nonempty disjoint sets, hence no set of K elements can intersect each one of them, a contradiction. Q.E.D.

A set with a nonempty intersection with each set in a collection W is called a *hitting set* for W. We now show that there exists a polynomial-time algorithm which finds a "small enough" hitting set for a collection W.

LEMMA 3.4. *There exists a polynomial-time algorithm that, given a collection W of nonempty sets, produces a hitting set for W with at most $k \log_e \|W\| + 2$ elements, where k is the cardinality of the smallest hitting set for W which has at least two elements.*

Proof. We use the following "greedy algorithm":

(1) Initialize: $W_0 \leftarrow W$, $H_0 \leftarrow \varnothing$, $r \leftarrow 0$.

(2) Iteratively, while $W_r \neq \varnothing$, let y be an element which appears in a maximal number of sets in W_r. Let $H_{r+1} \leftarrow H_r \cup \{y\}$, $W_{r+1} \leftarrow W_r - \{s | s \in W_r \wedge y \in s\}$, $r \leftarrow r+1$.

Clearly after the algorithm terminates with $r = p$, H_p is a hitting set for W, and H_p has p elements. Suppose there exists a hitting set for W with k elements ($k > 1$). Then for each W_i there exists a hitting set with k elements, hence an element y_i exists such that $\{s | s \in W_i \wedge y_i \in s\}$ contains at least $\|W_i\|/k$ sets. Hence

$$\|W_{i+1}\| \leqq \left(1 - \frac{1}{k}\right)\|W_i\| \qquad (0 \leqq i \leqq p-1),$$

and thus by induction on i

$$\|W_i\| \leqq \left(1 - \frac{1}{k}\right)^i \|W\| \qquad (0 \leqq i \leqq p).$$

We now claim that

$$p \leqq \left\lceil \frac{\log_e \|W\|}{-\log_e (1-1/k)} \right\rceil + 1.$$

To see this, suppose

$$p \geqq \left\lceil \frac{\log_e \|W\|}{-\log_e (1-1/k)} \right\rceil + 1.$$

Now, let

$$i_0 = \left\lceil \frac{\log_e \|W\|}{-\log_e (1-1/k)} \right\rceil;$$

then

$$\|W_{i_0}\| \leqq \left(1 - \frac{1}{k}\right)^{i_0} \|W\|.$$

Taking $\log_e$ of both sides gives

$$\log_e \|W_{i_0}\| \leqq i_0 \log_e \left(1 - \frac{1}{k}\right) + \log_e \|W\| \leqq 0.$$

Hence $\|W_{i_0}\| \leqq 1$, and since $\|W_{i_0+1}\| < \|W_{i_0}\|$ we conclude that $\|W_{i_0+1}\| = 0$, hence $p = i_0 + 1$. Thus in any case

$$(*) \qquad p \leqq \left\lceil \frac{\log_e \|W\|}{-\log_e (1-1/k)} \right\rceil + 1.$$

Now consider the real-valued function

$$\phi(x) = x + \log_e (1-x) \qquad (0 \leqq x < 1),$$

$\phi(0) = 0$ and

$$\phi'(x) = \frac{-x}{1-x} \qquad (0 < x < 1).$$

Since ϕ' is negative in (0, 1) we conclude that $\phi(x)<0$ for $0<x<1$, hence

$$x<-\log_e(1-x) \qquad (0<x<1).$$

Hence also

$$\frac{1}{k}<-\log_e\left(1-\frac{1}{k}\right) \quad \text{for } k>1$$

or

$$\frac{1}{-\log_e(1-1/k)}<k \quad \text{for } k>1.$$

Using (*) we finally get

$$p \leqq k \log_e \|W\|+2. \qquad \text{Q.E.D.}$$

Finally, we can prove

THEOREM 3.1. *If* $D \leqq^P_{bptt} S$, *where* S *is sparse and* D coNP-*hard, then* NP = P.

Proof. It is enough to consider $D = \mathrm{SAT}^c$. As in Fortune's original method [F], [M], we perform a depth-first search on the tree of self-reductions of the given Boolean formula G, using the reducibility to a sparse set to prune the tree. By Lemma 3.1 we may assume that there exist f_i computable in deterministic polynomial-time ($1 \leqq i \leqq t$) such that for all $x \in \Sigma^*$,

$$x \in \mathrm{SAT}^c \Leftrightarrow \bigvee_{i=1}^{t} (f_i(x) \in S).$$

The tree of self-reductions of G is as defined in § 2. Its nodes are all Boolean formulas, its root is G, and the leaves are constant formulas. The tree is not constructed in advance. Only nodes which are searched are constructed, as the search proceeds. As we shall see below, the reducibility to a sparse set will enable us to search only polynomially (in $|G|$) many nodes. Note also that for any nonsink H in the tree (which is also a nonconstant Boolean formula), its left son is easily constructed by substituting 0 for the first variable appearing in H, and making the appropriate simplifications. Its right son is similarly constructed by substituting 1 for the same variable.

We have the following rules for the search:

(Q-1) If a leaf which is the constant formula *true* is encountered, stop: $G \in \mathrm{SAT}$.

(Q-2) Every encountered leaf which is the constant formula *false* is marked F ("unsatisfiable").

(Q-3) If the two sons of a node (see § 2) are marked F the node itself is marked F.

(Q-4) If the root G is marked F, stop: $G \in \mathrm{SAT}^c$.

(Q-5) For each new node H encountered, a label $l(H) = \bigcup_{i=1}^{t} \{f_i(H)\}$ is computed.

(Q-6) If for a newly encountered node H, $l(H) \supset l(H')$, where H' is already marked F, mark H by F and never search below H.

(Q-7) Let $\hat{f}_i$ be a monotone polynomial bounding the increase in size under the transformation $f_i (1 \leqq i \leqq t)$. Let $k = \max_{1 \leqq i \leqq t} h(\hat{f}_i(|G|))$ where h is a monotone polynomial bound on the census of S. Let $K = k \cdot (\log_e 2 \cdot |G|+2)$. For each newly encountered node H for which (Q-6) does not apply form the collection

$$W_1 = \{l(H') - l(H) | H' \text{ is already marked } F\},$$

and apply the algorithm of Lemma 3.4 to find a hitting set for W_1. If the hitting set produced has more than K elements—mark H by F and never search below H.

For proving the validity of the algorithm, we only have to justify rule (Q-7). We claim that if the above hitting set has J elements where $J > K$, then $l(H)$ contains an element in S, which implies $H \in \mathrm{SAT}^c$. To see this, note that since the whole tree has at most $2^{|G|}$ nodes, $\log_e \|W_1\|$ is bounded above by $|G| \cdot \log_e 2$. By Lemma 3.4, the size q of a minimal hitting set for W_1 satisfies

$$q \geqq \frac{J}{\log_e \|W_1\| + 2} \geqq \frac{J}{|G| \cdot \log_e 2 + 2} > \frac{K}{|G| \cdot \log_e 2 + 2} = k.$$

Since there are at most k different elements in S of the form $f_i(H)(1 \leqq i \leqq t)$, where H is a node of the tree, S is not a hitting set for W_1, hence there exists some H', already marked F, such that $l(H') - l(H)$ does not have a member in S. Since $l(H')$ must have a member in S, we conclude that $l(H)$ has a member in S, hence $H \in \mathrm{SAT}^c$.

We now prove that the number of steps is bounded by a polynomial in $|G|$. We claim that there are at most $I(K+2, t)$ distinct paths from the root to nodes interior in the *pruned tree* (i.e., the tree without the subtrees which are never searched), which are marked F. To see this consider any sequence of interior nodes $H_1, H_2, \cdots, H_r$ (r any integer satisfying $r \geqq 1$) which are marked F, such that for $1 \leqq i \leqq r-1$, H_i was marked F before H_{i+1}, and such that if $i \neq j$, H_i and H_j are not on the same path from the root. By rule (Q-7), the hypotheses of Lemma 3.3 apply to the collection W of distinct (by rule (Q-6)) sets $\{l(H_i) | 1 \leqq i \leqq r\}$. Hence $r \leqq I(K+2, t)$. Thus, for some d, the total number of nodes visited is $O(|G|^d)$, since each path has at most $|G|$ nodes, and there is at most one visited path leading to a leaf with value 1. Q.E.D.

We now prove a similar result for NP.

THEOREM 3.2. *If* $D \leqq^P_{bptt} S$, *where* D *is* NP*-hard and* S *is a sparse set in* NP, *then* NP = P.

Proof. It is enough to consider $D = \mathrm{SAT}$, and to assume the existence of f_i computable in deterministic polynomial-time $(1 \leqq i \leqq t)$ such that for all $x \in \Sigma^*$

$$x \in \mathrm{SAT} \Leftrightarrow \bigvee_{i=1}^{t} (f_i(x) \in S).$$

Note that, in Lemma 3.1, if S in (1) is in NP then also S_1 in (2) is in NP. Now, letting $\tilde{S} = (S^c)^t$, we have:

$$x \in \mathrm{SAT}^c \Leftrightarrow (f_1(x), f_2(x), \cdots, f_t(x)) \in \tilde{S}.$$

Following Mahaney's method [M] we define a set $\hat{S} \in \mathrm{NP}$ which is called "pseudo $\tilde{S}$". $\hat{S}$ being in NP implies the existence of a function g computable in deterministic polynomial-time such that for all $x \in \Sigma^*$,

$$x \in \hat{S} \Leftrightarrow \bigvee_{i=1}^{t} (f_i(g(x)) \in S).$$

Now, if we can reduce SAT^c to $\hat{S}$ in polynomial-time, we clearly can use the method of Theorem 3.1 to decide SAT^c in polynomial-time. As in [M] it turns out that indeed SAT^c can be reduced to $\hat{S}$ if we correctly guess the census function. Our $\hat{S}$ is a

generalization of Mahaney's "pseudo complement", and is defined by

$$\hat{S} = \{((y_1, y_2, \cdots, y_t), \#^k, \#^n) \mid |y_j| \leqq n \text{ for } 1 \leqq j \geqq t,$$
$$k \leqq h(n), \text{ and there exist distinct strings } \xi_1, \xi_2, \cdots, \xi_k$$
$$\text{such that } \xi_i \in S, |\xi_i| \leqq n, \xi_i \neq y_j \ (1 \leqq i \leqq k, 1 \leqq j \leqq t)\}.$$

Here h is a monotone polynomial bounding the census function c_S. Intuitively, for $k = c_S(n)$, $\hat{S}$ coincides with $\tilde{S}$. Thus, using an adaptation of Mahaney's method of trying all possible values of the census function and using the tree search method we indeed can decide SAT^c in polynomial-time. We omit the details, which we believe can be filled in by the reader who is familiar with [M]. Q.E.D.

Remarks. (1) Theorem 3.1 generalizes Fortune's result [F], who proved that if a coNP-hard set is $\leqq_m^P$ reducible to a sparse set then NP = P. Theorem 3.2, for the special case of $\leqq_m^P$ reducibility, was proved by Mahaney [M], even without assuming that the sparse set itself is in NP. We prove the result for the more general $\leqq_{bptt}^P$ reducibility, but we need the extra assumption that the sparse set is in NP. It follows from Theorem 3.2 that if NP $\neq$ P, no set complete for NP under $\leqq_{bptt}^P$ reducibility can be sparse.

(2) If D is any conjunctively-self-reducible set, and $D \leqq_{bptt}^P S$, S a sparse set, then $D \in$ P. The proof is almost the same as for Theorem 3.1.

(3) If D is PSPACE-hard and $D \leqq_{bptt}^P S$, S a sparse set, then PSPACE = P. The proof is similar to the proof of Theorem 3.1.

We can assume $D =$ QBF (the set of true quantified Boolean formulas [ST]). Then we generalize the proof of Fortune's result for the case QBF $\leqq_m^P S$ in the same way that Theorem 3.1 generalizes the case $\mathrm{SAT}^c \leqq_m^P S$.

4. Nonpositive *tt* reducibilities and applications. We investigate the consequences of the existence of a certain nonpositive polynomial-time bounded truth-table reducibility of a conjunctively-self-reducible set to a sparse set.

For a monotone unbounded function h, let $\mathrm{DN}(h)$ denote the class of all subsets of Σ^* with census function bounded above by $O(h)$.

THEOREM 4.1. *If D is strongly conjunctively-self-reducible, and there exist h_1, h_2, f, g computable in deterministic polynomial-time, $S_1 \in \mathrm{DN}(h_1)$, $S_2 \in \mathrm{DN}(h_2)$ such that for all $x \in \Sigma^*$,*

$$x \in D \Leftrightarrow f(x) \in S_1 \wedge g(x) \notin S_2,$$

then

$$D \in \mathrm{DTIME}\,(n^v \cdot (t_1(n) \cdot t_2(n))^{t_3(n)})$$

where v is an integer and

$$t_1(n) = p_1 \circ h_1 \circ q_1(n), \quad t_2(n) = p_2 \circ h_2 \circ q_2(n), \quad t_3(n) = p_3 \circ h_2 \circ q_3(n)$$

for some polynomials p_1, p_2, p_3, q_1, q_2, q_3. If D is just conjunctively-self-reducible, then under the above hypotheses

$$D \in \mathrm{DTIME}\,((n^v t_1(n) t_2(n))^{t_3(n)}),$$

t_1, t_2, t_3 as above.

Proof. Let G be an instance of the decision problem for D. Consider the tree of self-reductions of G. This tree is defined in § 2. Its set of directed edges is:

$$\{(H, H') \mid H' \text{ is a son of } H \text{ in the tree}\}.$$

Before proceeding with the proof of the theorem, we prove the following:

LEMMA 4.1. *Let* $r \geqq 1$, $H_i (1 \leqq i \leqq r)$ *nodes in the above tree such that there exists a directed path in the tree from* H_i *to* $H_{i+1} (1 \leqq i \leqq r-1)$, $f(H_i) \in S_1 (1 \leqq i \leqq r)$, *and for some* j, $1 \leqq j \leqq r$ $g(H_j) \notin S_2$. *Then* $H_r \in D$.

Proof. $H_j \in D$ since $f(H_j) \in S_1 \wedge g(H_j) \notin S_2$. Conjunctive-self-reducibility implies that if a node is in D then its sons are all in D. Hence $H_r \in D$. Q.E.D.

We now proceed with the proof of Theorem 4.1. Let $\hat{f}$, $\hat{g}$ be monotone polynomials bounding the increase in size under the transformations f, g, respectively. Let

$$k_1 = h_1(\hat{f}|G|), \qquad k_2 = h_2(\hat{g}|G|).$$

We have:

(1) There are at most k_1 distinct values of the form $f(H)$, where H is a node in the tree and $f(H) \in S_1$. There are at most k_2 distinct values of the form $g(H)$, where H is a node in the tree and $g(H) \in S_2$.

We now describe the algorithm for deciding whether $G \in D$. As in Fortune's method, we perform a depth-first search on the tree. We have the following rules:

(R-1) If a leaf in D^c is encountered, stop: $G \in D^c$.

(R-2) Every leaf in D is marked F (Base case).

(R-3) If all sons of a node are marked F, the node itself is marked F.

(R-4) If G is marked F, stop: $G \in D$.

(R-5) For each node H encountered, $f(H)$ and $g(H)$ are computed.

(R-6) If for a newly encountered node H, $(f(H), g(H)) = (f(H'), g(H'))$ for a node H' already marked F—mark H by F and never search below H.

(R-7) For each newly encountered node H let $m(H)$ be 1 if there exists a node H', already marked F, such that $f(H) = f(H')$. Let $m(H)$ be 0 otherwise.

(R-8) For each newly encountered node H compute a set

$$l(H) = \{g(H') | H' \text{ is on the path from the root to } H, \text{ and } m(H') = 1\}.$$

(R-9) If for a newly encountered node H, $l(H)$ contains more than k_2 elements—mark H by F and never search below H. $H \in D$.

For proving the validity of the algorithm, note that (R-9) is justified by (1) above and Lemma 4.1. The other details are obvious.

We will now prove the bound on the number of steps, by establishing a bound on the total number of nodes encountered. We call a node H a *pivot* if either H is the root G, or $l(H)$ has more elements than $l(H')$, where H' is the father of H. Clearly in the latter case $l(H)$ is the union of $l(H')$ with a one element set. For each pivot H let $T(H)$ be the pruned subtree with root H. Also let $\sigma(H)$, *the skeleton of* H be defined as follows: $\sigma(H)$ is the part of $T(H)$ with nodes H' such that either $l(H')$ has the same number of elements as $l(H)$, or H' is a pivot with $l(H')$ having one more element than $l(H)$. Clearly $\sigma(H)$ is also a tree. We have the following property:

(2) If H is a pivot, then in $\sigma(H)$ there are at most $k_1(k_2+1)$ distinct paths from H to interior nodes in $\sigma(H)$ marked F.

This is seen as follows: If H' is marked F, then $f(H') \in S_1$, hence there are at most k_1 possible values for $f(H')$. For a fixed $\alpha \in S_1$, suppose that there are k_2+2 distinct paths leading from the root H to interior nodes $H_1, H_2, \cdots, H_{k_2+2}$, marked F in this order, such that $f(H_i) = \alpha (1 \leqq i \leqq k_2+2)$. By rule (R-6), the values $g(H_i)$ for $2 \leqq i \leqq k_2+2$ are all different—otherwise some H_i would not be interior. Hence not all the $g(H_i)$ for $2 \leqq i \leqq k_2+2$ are in $l(H)$, by (R-9) above. Since H_1 is marked F first, $m(H_i) = 1$ for $2 \leqq i \leqq k_2+2$. (Note that H_1 is already marked F when each H_i for

$2 \leqq i \leqq k_2+2$ is first encountered.) Thus for some $2 \leqq i \leqq k_2+2$ $l(H_i)$ has more elements than $l(H)$, hence H_i cannot be interior in $\sigma(H)$.

From (2) we have the following:

(3) For H as in (2), there are at most $d(1+k_1(k_2+1))$ distinct paths from H to the leaves of $\sigma(H)$, where d is the constant bounding the degree of the tree.

To see this, note that in the whole pruned tree there can be only one path leading from G to nodes which are never marked F. This, in the case that $G \notin D$, is the unique path from G to the first leaf in D^c which is encountered. Hence by (2) above, the number of distinct paths from H to interior nodes of $\sigma(H)$ is at most $1+k_1(k_2+1)$. Each such path can split to at most d paths in $\sigma(H)$.

(4) If H is a pivot, with $l(H)$ having r elements, then the pruned subtree with root H has at most $[d(1+k_1(k_2+1))]^{k_2+1-r}$ distinct paths from H to the leaves. We prove this by backward induction on r. By rule (R-9), the assertion is true for $r = k_2+1$. Now consider a pivot H with $l(H)$ having $r < k_2+1$ elements. By (3) there are at most $d(1+k_1(k_2+1))$ distinct paths from H to the leaves of $\sigma(H)$. Also, each leaf H' of $\sigma(H)$ is either a leaf of the whole tree, or a pivot with $l(H)'$ having $r+1$ elements, hence there are, by the induction hypothesis, at most $[d(1+k_1(k_2+1))]^{k_2-r}$ paths from H' to the leaves. Thus from H there are at most $[d(1+k_1(k_2+1))] \cdot [d(1+k_1(k_2+1))]^{k_2-r} = [d(1+k_1(k_2+1))]^{k_2-r+1}$ paths to the leaves.

Now, since $l(G) = \varnothing$, there are at most $[d(1+k_1(k_2+1))]^{k_2+1}$ distinct paths from the root G to the leaves of the pruned tree. Each such path contains at most $z(|G|)$ nodes, where z is a fixed polynomial bounding the height of the tree derived from G, and the result follows.

Now, if D is just conjunctively-self-reducible, the constant d above has to be replaced by some polynomial in $|G|$. Q.E.D.

Since every complement of a (strongly) conjunctively-self-reducible set is (strongly) disjunctively-self-reducible, we have the following corollary:

COROLLARY 4.1. If $D \leqq_m^P S$, $S \in \mathrm{DN}(h)$, *h computable in deterministic polynomial-time, then*:

(i) *If D is strongly disjunctively-self-reducible then*

$$D \in \mathrm{DTIME}(n^v \cdot t_1(n)^{t_2(n)})$$

for some integer v and where for some polynomials p_1, p_2, q_1, q_2,

$$t_1(n) = p_1 \circ h \circ q_1(n), \qquad t_2(n) = p_2 \circ h \circ q_2(n).$$

In particular, if $t_1(n)^{t_2(n)}$ is bounded above by a polynomial (for instance, if $h(n) = O(\log\log(n))$) then $D \in \mathrm{P}$.

(ii) *If D is disjunctively-self-reducible then*

$$D \in \mathrm{DTIME}((n^v \cdot t_1(n))^{t_2(n)}),$$

v, t_1, t_2 as above.

As applications, in (i) above D could be INTFAC, and in (ii) above D could be the disjunctively-self-reducible set which is polynomial-time many-one equivalent to ISO.

It should be noted that any set in NP, and in particular any disjunctively-self-reducible set, has a strongly disjunctively-self-reducible set associated with it. If $E \in \mathrm{NP}$, let M be a nondeterministic Turing machine which accepts E in

nondeterministic polynomial-time. Let

$$A = \{(x, (\#\#)^k) | x \text{ is a configuration of } M \text{ from which an accepting configuration can be reached in at most } k \text{ steps}\}.$$

Clearly, $E \leqq_m^P A$. A is strongly disjunctively-self-reducible. Let d be a bound on the number of nondeterministic choices of M in one step. If $k = 0$ $(x, (\#\#)^k) \in A$ if and only if x is an accepting configuration, and this can be checked in deterministic polynomial-time. If $k > 0$, let $x_1', x_2', \cdots, x_r'$ $(r \leqq d)$ be the configurations of M which can be reached from x in one step. Then

$$(x, (\#\#)^k) \in A \Leftrightarrow \bigvee_{i=1}^{r} (x_i', (\#\#)^{k-1}) \in A.$$

Since $|x_i'| \leqq |x| + 1$, $|(x_i', (\#\#)^{k-1})| < |(x, (\#\#)^k)| (1 \leqq i \leqq r)$.

We note, however, that a reducibility of E to a sparse set does not seem to imply a reducibility of A to a sparse set.

We now turn to the second application of Theorem 4.1. Define an *S-approximate algorithm* for a set D as a polynomial-time algorithm which decides D correctly on $\Sigma^* - S$, and gives a wrong answer on any input from S $(S \subset \Sigma^*)$. The closeness of approximation is measured by the census function of S. For S sparse, the assumption of the existence of the above algorithm is weaker than the existence of an "almost polynomial-time" algorithm for D, [B], [MP], which never gives a wrong result, but requires more than polynomial-time on a sparse set S. For example, the latter assumption implies $S \in \mathrm{P}$. It is known [B], [MP] that if there exists an almost polynomial-time algorithm to solve an NP-hard question then NP = P. Now, if one could settle the basic question regarding reducibility to sparse sets, by showing that if an NP-complete set is $\leqq_T^P$ reducible to a sparse set then NP = P, this will also show that unless NP = P, no S-approximate algorithm according to our definition exists for NP-hard sets, with sparse S.

Now, the above basic question is still open, nevertheless we are able to apply Theorem 4.1 to show that if S as above is "sparse enough" (and having census $O(\log \log (n))$ is enough) then NP = P. This also can be interpreted as follows: Suppose we measure the "distance" between two sets by the census of their symmetric difference. How "close" (according to this criterion) can be an NP-hard set to a set in P? The answer is partly given in the following theorem.

THEOREM 4.2. *Suppose D is* NP*-hard, f_1 computable in deterministic polynomial-time, $f_1: \Sigma^* \to \{0, 1\}$ such that the set $\{x | f_1(x) \neq \chi_D(x)\}$ is in* DN (h), *where h is computable in deterministic polynomial-time. Then every set in* NP is in DTIME $(n^v t_1(n)^{t_2(n)})$ *for some integer v and where for some polynomials p_1, p_2, q_1, q_2,*

$$t_1(n) = p_1 \circ h \circ q_1(n), \qquad t_2(n) = p_2 \circ h \circ q_2(n).$$

Hence in particular, if $h(n) = O(\log \log (n))$ then NP = P.

Proof. Let

$$S = \{x | f_1(x) \neq \chi_D(x)\}.$$

Then for all $x \in \Sigma^*$,

$$x \in D \Leftrightarrow (f_1(x) = 0 \wedge x \in S) \vee (f_1(x) = 1 \wedge x \notin S).$$

Let f_2 (which is computable in deterministic polynomial-time) satisfy: For all $x \in \Sigma^*$,

$$x \in \mathrm{SAT} \Leftrightarrow f_2(x) \in D.$$

Then for all $x \in \Sigma^*$,

$$x \in \mathrm{SAT} \Leftrightarrow (f_1(f_2(x)) = 0 \wedge f_2(x) \in S) \vee (f_1(f_2(x)) = 1 \wedge f_2(x) \notin S).$$

Hence since for any x, $f_1(f_2(x)) \in \{0, 1\}$, we also have for all $x \in \Sigma^*$,

$$x \in \mathrm{SAT}^c \Leftrightarrow (f_1(f_2(x)) = 0 \wedge f_2(x) \notin S) \vee (f_1(f_2(x)) = 1 \wedge f_2(x) \in S).$$

Also, if $S = \varnothing$ or if $S = \Sigma^*$ then $\mathrm{SAT}^c \in \mathrm{P}$. So suppose there exist μ_0, μ_1 such that $\mu_0 \in S$ and $\mu_1 \notin S$. Define f, g computable in deterministic polynomial-time as follows:

$$f(x) = \begin{cases} f_2(x) & \text{if } f_1(f_2(x)) = 1, \\ \mu_0 & \text{if } f_1(f_2(x)) = 0, \end{cases}$$

$$g(x) = \begin{cases} f_2(x) & \text{if } f_1(f_2(x)) = 0, \\ \mu_1 & \text{if } f_1(f_2(x)) = 1. \end{cases}$$

Then for all $x \in \Sigma^*$,

$$x \in \mathrm{SAT}^c \Leftrightarrow f(x) \in S \wedge g(x) \notin S.$$

The result now follows by Theorem 4.1. Q.E.D.

We now use Theorem 4.2 and a result due to Meyer to prove the following theorem.

THEOREM 4.3. *For no* ETIME*-hard (under* $\leqq_m^P$ *reducibility) set* D *does there exist* f *computable in deterministic polynomial-time* $f: \Sigma^* \to \{0, 1\}$ *such that* $\{x \mid f(x) \neq \chi_D(x)\}$ *is in* DN (log log (n)).

Proof. Let

$$S = \{x \mid f(x) \neq \chi_D(x)\}.$$

D is $\leqq_T^P$ reducible to the sparse set S. By a result due to Meyer (see [KL]), if an ETIME-hard set is $\leqq_T^P$ reducible to a sparse set, then NP $\neq$ P. On the other hand, since D is NP-hard, NP = P follows by Theorem 4.2, resulting in a contradiction. Q.E.D.

This should be compared with Meyer's result (see [BH]) that no ESPACE-complete set can be $\leqq_T^P$ reducible to a sparse set. So in particular an ESPACE-hard set cannot have a sparse symmetric difference with a set in P.

5. Do there exist sets in NP − coNP which are polynomial-time many-one reducible to sparse sets? In [HM] the question whether NP $\neq$ P implies the existence of sparse sets in NP − P is raised. We are concerned with a similar question: Assuming NP $\neq$ coNP, what are the implications of the existence of a sparse set in NP − coNP? The following result is essentially from [BO].

LEMMA 5.1. ESPACE $\neq$ NETIME *if and only if there exists a tally set (a set over one symbol) in* PSPACE − NP.

Proof. See [BO] for the proof of similar results. Q.E.D.

We now show the following:

THEOREM 5.1. *If there exists a set* $D \in$ NP − coNP *such that* $D \leqq_m^P S$, *where* S *is sparse, then there exists a tally set in* PSPACE − NP, *hence* ESPACE $\neq$ NETIME.

Proof. Using a method of Mahaney [M], we first show that we may assume that S itself is in NP − coNP. Let f be computable in deterministic polynomial-time such that for all $x \in \Sigma^*$, $x \in D \Leftrightarrow f(x) \in S$. Let $f_1(x) = f(x)\#^{|x|}$, where $\#$ is a new symbol, $S_1 = f_1(D)$. Then S_1 is sparse. Also, to decide whether $y \in S_1$, guess an x satisfying $|x| \leqq |y|$, verify that $f_1(x) = y$ and that $x \in D$. Thus $S_1 \in$ NP. Also $S_1 \notin$ coNP, since $D \leqq_m^P S_1$ and $D \notin$ coNP. Now let $H = \{0^{\phi(i,j)} \mid c_S(i) = j, i = 0, 1, \cdots\}$, where $\phi(i, j)$ is the

conventional pairing function $\phi(i,j) = i + \frac{1}{2}(i+j)(i+j+1)$, and c_S is the census function of S. We now show that $H \in$ PSPACE, by exhibiting a nondeterministic polynomial-space algorithm for H. Given 0^r, guess i and j and check whether $\phi(i,j) = r$. Then check all $x \in \Sigma^*$ with $|x| \leqq i$ for membership in S, thus determining $c_S(i)$. Since $S \in$ NP, this can be done in polynomial-space. Finally, verify that $c_S(i) = j$. Now we show that if $H \in$ NP then $S \in$ coNP, resulting in a contradiction. The method is essentially in [M].

Suppose $H \in$ NP. Let h be some polynomial bound on c_S. To decide whether $x \in S^c$ in nondeterministic polynomial-time, do the following:

1) Guess a j satisfying $j \leqq h(|x|)$. Verify that $0^{\phi(|x|,j)} \in H$ in nondeterministic polynomial-time, thus proving that $c_S(|x|) = j$.

2) Guess distinct $x_1, x_2, \cdots, x_j$ such that $|x_i| \leqq |x|$ and $x_i \neq x (1 \leqq i \leqq j)$, verify in nondeterministic polynomial-time that for $1 \leqq i \leqq j$, $x_i \in S$. Since $c_S(i) = j$, $x \in S^c$ if and only if such $x_1, x_2, \cdots, x_j$ exist. Q.E.D.

The above result might imply that it will be very difficult to show [NP $\neq$ coNP $\Rightarrow$ there exists a sparse set in NP $-$ coNP] because this will also imply [NP $\neq$ coNP $\Rightarrow$ NETIME $\neq$ ESPACE]. Whether the latter implication holds is an open and apparently difficult question.

6. Concluding remarks. We have obtained new results regarding the structure of sets which are hard for NP and other classes with respect to certain polynomial-time truth-table reducibilities, and sets in NP $-$ coNP (assuming NP $\neq$ coNP). The results are applied to the question how "close" can be difficult or apparently difficult sets to sets in P. We also answer new restricted forms of very basic open problems which are:

1) Does $\leqq_T^P$ reducibility of NP complete sets to sparse sets imply NP = P?

Note that $A \leqq_{bptt}^P S$ is a special case of $A \leqq_T^P S$, in which the polynomial-time oracle Turing machine which accepts A with oracle S can ask only a bounded number of questions, and furthermore, the Boolean formula which describes the outcome of the computation as a function of the oracle answers is monotone increasing.

2) Can ETIME-complete sets be $\leqq_T^P$ reducible to sparse sets?

Clearly a set with census $O(\log \log (n))$ is a special case of a sparse set. Also if A has an S-approximate algorithm, then A is accepted by a polynomial-time oracle Turing machine which accepts A with an oracle S, such that given input x, only two questions, which are prepared in advance, are presented to the oracle, and the outcome of the computation as a function of the oracle answers is described by a Boolean formula in two variables of a specific form (see the proof of Theorem 4.2).

Acknowledgments. I would like to thank Professor Stephen A. Cook for his encouragement, very helpful suggestions which helped to improve the results of this work, and patient and careful reading of the manuscript, resulting in remarks which lead to an improved presentation, also for bringing [S] to my attention and supplying me with [S] and [MP]. I am grateful to the referee for pointing out a mistake in the previous version of Theorem 3.2.

REFERENCES

[B] P. BERMAN, *Relationship between density and deterministic complexity of* NP*-complete languages*, Fifth International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 62, Springer Verlag, Berlin, 1978, pp. 63–71.

[BH] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of* NP *and other complete sets*, this Journal, 6 (1977), pp. 305–322.

[BO] R. BOOK, *Tally languages and complexity classes*, Inform. Control, 26 (1974), pp. 186–194.

[C] S. A. COOK, *The complexity of theorem proving procedures*, Proc. 3rd Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[C1] ———, unpublished lecture notes, CSC2428F 1980, Dept. Computer Science, Univ. Toronto, Toronto, Ontario.

[F] S. FORTUNE, *A note on sparse complete sets*, this Journal, 8 (1979), pp. 431–433.

[HM] J. HARTMANIS AND S. MAHANEY, *An essay about research on sparse* NP *complete sets*, Computer Science Dept. Technical Report 80-422, Cornell Univ., Ithaca, NY, 1980.

[HU] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

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

[KL] R. KARP AND R. LIPTON, *Some connections between non-uniform and uniform complexity classes*, Proc. 12th ACM Symposium on Theory of Computing, May 1980, pp. 302–309.

[L] N. LYNCH, *On reducibility to complex or sparse sets*, J. Assoc. Comput. Mach., 22 (1975), pp. 341–345.

[LLS] R. LADNER, N. LYNCH AND L. SELMAN, *Comparison of polynomial-time reducibilities*, Proc. 6th ACM Symposium on Theory of Computing, 1974, pp. 110–121; Theoret. Comput. Sci., 1 (1975), pp. 103–123.

[LPR] L. LANDWEBER, R. LIPTON AND E. ROBERTSON, *On the structure of sets in* NP *and other complexity classes*, Computer Science Dept. Technical Report #342, Univ. of Wisconsin-Madison, 1978.

[M] S. R. MAHANEY, *Sparse complete sets for* NP: *Solution of a conjecture of Berman and Hartmanis*, IEEE 21st Annual Symposium on Foundations of Computer Science, 1980, pp. 54–60.

[MP] A. MEYER AND M. PATERSON, *With what frequency are apparently intractable problems difficult?* MIT Laboratory for Computer Science TM-126, Massachusetts Institute of Technology, Cambridge, 1979.

[S] R. SOLOVAY, *On sets Cook-reducible to sparse sets*, this Journal, 5 (1976), pp. 646–652.

[ST] L. STOCKMEYER, *The complexity of decision problems in automata theory and logic*, MAC TR-133, Project MAC, Massachusetts Institute of Technology, Cambridge.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0002 $01.25/0

CONSTRUCTION OF OPTIMAL α-β LEAF TREES WITH APPLICATIONS TO PREFIX CODE AND INFORMATION RETRIEVAL*

DAVID M. CHOY† AND C. K. WONG‡

Abstract. In this paper, we study a special class of trees called α-β leaf trees with degree r. These trees arise in information retrieval problems as well as prefix coding problems. An efficient method for constructing optimal trees is presented. The method is combinatorial in nature instead of the integer programming approach followed by other authors to study similar problems. The construction time is linear to the number of leaves of the tree.

Key words. α-β leaf trees, prefix coding, information retrieval, integer programming, analysis of algorithms, combinatorial optimization

1. Introduction. In this paper, we consider a special class of trees, called α-β *leaf trees with degree r.* Given integer $r \geqq 2$, real numbers $\beta \geqq \alpha > 0$, an α-β leaf tree with degree r is a tree such that each of its internal nodes has at most r children. The weights, denoted by $\omega t(\cdot)$, are defined as follows:

(a) $\omega t(\text{root}) = 0$;

(b) $\omega t(i\text{th child from the left}) = \omega t(\text{parent}) + \alpha + (i-1)\beta$, $1 \leqq i \leqq r$;

(c) $\omega t(\text{tree}) = \text{sum of } \omega t(\text{leaf}) \text{ over all leaves.}$

Figure 1a gives an example of an α-β leaf tree with $r = 3$ and $\omega t(\text{tree}) = 11\alpha + 6\beta$. Figure 1b shows a binary tree representation of the same α-β tree with each left path associated with α and each right path associated with β. The weight of a node is equal to the sum of the edges from the root to the node in the binary tree.

The problem we study here can be described as follows: Given integers $n \geqq 1$, $r \geqq 2$, and real numbers $\beta \geqq \alpha > 0$, we want to construct an α-β leaf tree with degree r such that it has n leaves and that $\omega t(\text{tree})$ is minimum. (Note that the assumption $\beta \geqq \alpha$ is not essential. The approach described in this paper can also be adapted for the case of $\beta < \alpha$.)

This problem arises in at least the following two applications [3]:

(i) *Information retrieval.* Given a collection of records for storage and retrieval, we want to assign identifying keys to these records and store these keys along with the addresses of the corresponding records so that the retrieval of records (through search of the keys) can be performed efficiently. The keys are composed of strings of symbols over some given key alphabet of size r. For example, let {1, 2, 3} be the given key alphabet and let $R1$, $R2$, $R3$, $R4$, $R5$ be five records to be stored. We can assign keys to the records as follows:

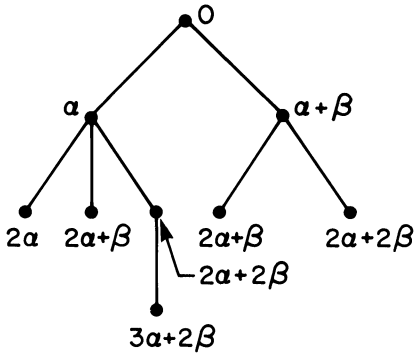
111	$R1$
12	$R2$
13	$R3$
22	$R4$
23	$R5$

and then store these keys in a tree structure such that each key corresponds to a leaf (Fig. 2a, b). To retrieve the address of a record, we have to follow a chain of pointers from the root of the tree to the appropriate leaf. For example, to get the address of

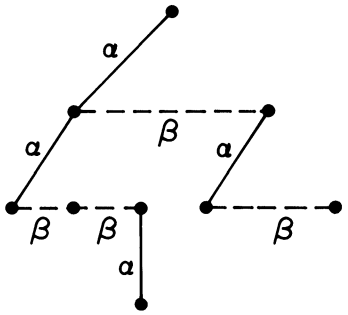
* Received by the editors July 6, 1981, and in revised form March 15, 1982.

† IBM Research Laboratory, San Jose, California 95193.

‡ IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.



(a) *Weights of the nodes.*



(b) *Binary tree representation of the tree in* (a).

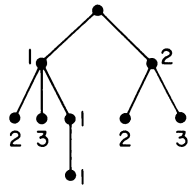
FIG. 1

$R4$, we have to follow one downward pointer followed by a horizontal pointer and two more downward pointers. In general, the downward pointer and horizontal pointer may have different retrieval costs due to the clustering of the nodes in storing the tree. If we assign cost α to the downward pointer and cost β to the horizontal pointer, and r is the alphabet size, then a doubly chained tree can be thought of as an α-β leaf tree with degree r and the weight of the tree as defined above reflects the expected search time for the record addresses assuming all records have the same retrieval probability.

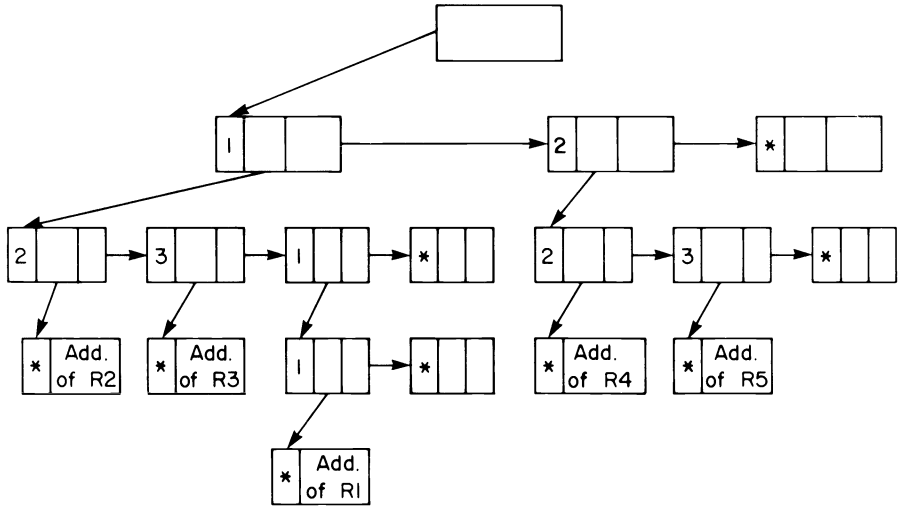
(ii) *Prefix code.* Suppose we have r coding symbols denoted by $1, 2, \cdots, r$ with costs $C_1, C_2, \cdots, C_r$, respectively. Consider any prefix code in which the prefix Sj, S a string and j a symbol, is utilized only if $S1, S2, \cdots, S(j-1)$ are utilized. Figure 2c is an example for $r=4$, where 111, 12, 2, 31, 32, 33, 4 are the code words and the cost of any word is the sum of the costs of the symbols in the word. In the case when $C_i = \alpha + (i-1)\beta$, $i = 1, 2, \cdots, r$, we can regard a prefix code as an α-β leaf tree with degree r, and the weight of the tree represents the average word cost.

Thus an optimal α-β leaf tree with degree r, i.e., one with the minimum weight, corresponds to a key assignment to records such that the expected search time for the keys is minimized, or to a prefix code with minimum average code word cost.

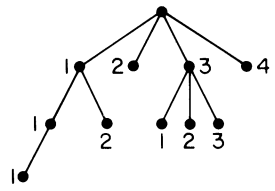
The more general prefix code problem where C_i, $i = 1, 2, \cdots, r$, are arbitrary integers was first solved by Karp [7] using integer programming methods, which may have exponential time complexity. Stanfel used the same approach to solve the special



(a)



(b)



(c)

FIG. 2

case of optimal α-β leaf trees when β/α is an integer [3]. A nearly optimal algorithm that runs in linear time for the general case was proposed by Mehlhorn [8].

In this paper, we propose an algorithm that runs in linear time to construct optimal α-β leaf trees for arbitrary α, β. The method is combinatorial in nature and is based on an enumeration technique previously employed to study other α-β tree problems, namely, α-β node trees [1] and α-β node trees with degree r [2].

2. Tree array. Given r, α, and β, consider any α-β leaf tree with degree r (tree for short). Each node in the tree has a weight $i\alpha + j\beta$ for some nonnegative integers i, j. Suppose we count the number of nodes with the same ordered pair i, j and enter

α \ β	0	1	2	3
0	1	0	0	0
1	1	1	0	0
2	1	2	2	0
3	0	0	1	0

FIG. 3

this number at the ith row and jth column of an array. We then obtain an array of nonnegative integers.

For example, Fig. 3 shows the array corresponding to the tree in Fig. 1a. Notice that such mapping f from a tree to an array always exists and is unique. We shall next discuss the range of f, from which the construction of an optimal tree will result.

Given $r \geqq 2$, consider the following semi-infinite array. Let $e(i, j)$ be the entry at the ith row ($i \geqq 0$) and jth column. Construct the array according to the following rules:

(1) $e(i, j) = 0$ for $j < 0$,

(2) $e(0, 0) = 1$,

(3) $e(0, j) = 0$ for $j > 0$,

(4) $e(i, j) = \sum_{k=0}^{r-1} e(i-1, j-k)$ for $i > 0$.

An example of this array for $r = 4$ is given in Fig. 4. The implication of $e(i, j)$ is that it is possible to construct a tree in which $e(i, j)$ of the nodes (internal nodes or leaves) each has weight $i\alpha + j\beta$. But there is no tree that can map into an array in which the (i, j) entry is larger than $e(i, j)$ for any i and j. This array has been studied in [2]. Our approach here is to select an appropriate number of nodes from certain entries of this array to form an optimal tree. After the selection, we shall actually construct the tree in a natural way from these selected nodes.

While this array is sufficient for constructing α-β *node* trees with degree r as demonstrated in [2], it is, however, insufficient for *leaf* trees. A decomposition of

α \ β	0	1	2	3	4	5	6	7	8
0	1	0	0	0	0	0	0	0	•••
1	1	1	1	1	0	0	0	0	
2	1	2	3	4	3	2	1	0	
3	1	3	6	10	12	12	10	6	
4	1	4	10	20	31	40	44	40	
	⋮								

$r = 4$

FIG. 4. *Array with entry* $e(i, j)$.

each entry into three components is needed. We shall let the entry at row i ($i \geqq 0$) and column j ($j \geqq 0$) be denoted by a triplet $E(i,j) = (e_1(i,j), e_2(i,j), e_3(i,j))$ where each component is a nonnegative integer, and the sum of the components is given by $e(i,j) = e_1(i,j) + e_2(i,j) + e_3(i,j)$. Within the triplet, $e_1(i,j)$ denotes the number of nodes (each with weight $i\alpha + j\beta$) which are the first children of their respective parents. Similarly, $e_2(i,j)$ represents the second children, and $e_3(i,j)$ the third and further children. Clearly, if $r = 2$, then $e_3(i,j) = 0$ for all i, j. Consequently, the third components of an entry can be dropped and we have $E(i,j) = (e_1(i,j), e_2(i,j))$.

Given integer $r \geqq 2$ and $i \geqq 0$, we construct an alternate semi-infinite array as follows:

$$E(0,0) = (1,0,0),$$

$$E(0,j) = (0,0,0) \quad \text{for } j \neq 0,$$

$$E(i,j) = \left(e(i-1,j), e(i-1,j-1), \sum_{k=2}^{r-1} e(i-1,j-k)\right) \quad \text{for } i > 0.$$

By the same approach as used in the derivation of [2, Eq. (2)], we have

$$e_3(i,j) = \sum_{k=2}^{r-1} e(i-1,j-k) = e(i,j-1) - e(i-1,j-r) - e(i-1,j-1).$$

Recall that $e(i,j) = 0$ for $j < 0$. Also, note that $e_1(i,j) = e_2(i,j+1)$ for $i > 0$. This fact will be utilized in our algorithm. Figure 5 shows an example of such an array for $r = 4$. It is a decomposition of the array shown in Fig. 4. For properties of this array, the reader may refer to [2].

3. z-line. We are going to truncate the semi-infinite array defined in § 2, so that the subarray obtained corresponds to an optimal tree.

Let $t_i = \lceil i\beta/\alpha \rceil$, for $i = 0, 1, 2, \cdots$. Then

(1) $$t_i\alpha \geqq i\beta > (t_i - 1)\alpha,$$

(2) $$t_i \geqq i,$$

(3) $$t_{i+j} + 1 \geqq t_i + t_j \geqq t_{i+j}$$

α \ β	0	1	2	3	4	5	6	7	8
0	1 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	• • •
1	1 0 0	0 1 0	0 0 1	0 0 1	0 0 0	0 0 0	0 0 0	0 0 0	
2	1 0 0	1 1 0	1 1 1	1 1 2	0 1 2	0 0 2	0 0 1	0 0 0	
3	1 0 0	2 1 0	3 2 1	4 3 3	3 4 5	2 3 7	1 2 7	0 1 5	
4	1 0 0	3 1 0	6 3 1	10 6 4	12 10 9	12 12 16	10 12 22	6 10 24	
	•								
	•								
	•								

$r = 4$

FIG. 5. *Array with entry $E(i,j)$.*

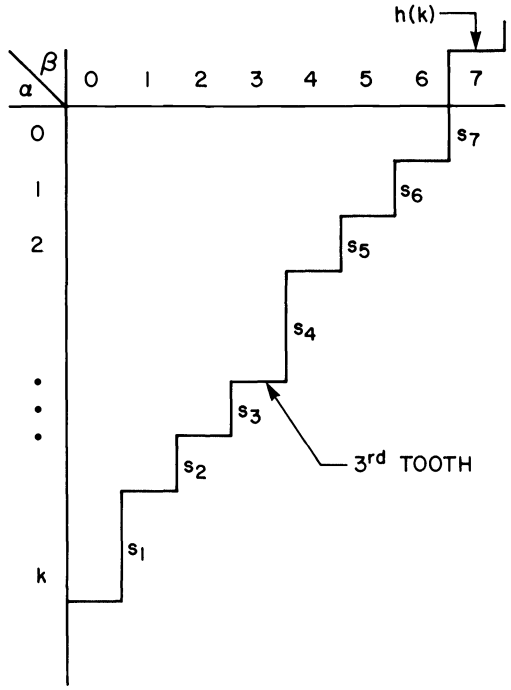


FIG. 6. *An example of a z-line.*

for $i, j \geqq 0$. Furthermore, if i is the smallest positive integer such that $t_i\alpha = i\beta$, then $t_j\alpha = j\beta$ if and only if j is divisible by i for all $j \geqq 0$.

We now define a staircase-like semi-infinite line called a z-line, which will be used to truncate the tree array. A z-line consists of horizontal and vertical segments (Fig. 6). The vertical segments are of length $s_i = t_i - t_{i-1} \geqq 1$, $i = 1, 2, \cdots$, (in terms of the number of array entries), and are arranged upwards. The lower endpoint of s_{i+1} is connected to the upper endpoint of s_i by a horizontal segment of unit length with s_i being to the left of s_{i+1}. This horizontal segment will be referred to as the ith "tooth" of the z-line.

Overlay the z-line on top of the tree array with segment s_i always between columns $i-1$ and i, as shown in Fig. 6. Suppose the head of the z-line (i.e., the 0th tooth) is between rows k and $k+1$ at column 0. Then the jth tooth is at column j and between rows $(k - t_j)$ and $(k - t_j + 1)$, $j = 0, 1, 2, \cdots$. The entry $E(k - t_j, j)$ is said to be *at the* jth *tooth* of the z-line.

Let N_k be the sum of the entries at all teeth plus the third components of the entries immediately below the "teeth":

$$N_k = \sum_{j=0}^{h(k)-1} [e(k - t_j, j) + e_3(k + 1 - t_j, j)]$$

$$= \sum_{j=0}^{h(k)-1} [e_1(k + 1 - t_j, j) + e_3(k + 1 - t_j, j)], \tag{4}$$

where $h(k)$ is the number of teeth inside the array and is given by

$$h(k)=\left\lfloor\frac{k\alpha}{\beta}\right\rfloor+1.$$

Figure 6 shows an example of a z-line with $\alpha=3$, $\beta=4$, which imply $t_0=0$, $t_1=2$, $t_2=3$, $t_3=4$, $t_4=6$, $t_5=7$, $t_6=8$, $t_7=10, \cdots$, and $s_1=2$, $s_2=1$, $s_3=1$, $s_4=2$, $s_5=1$, $s_6=1, \cdots$.

We have the following observation concerning a z-line:

LEMMA 1. *Referring to Fig. 7, let A be the entry at the 0th tooth. Let B be the entry at any tooth. Let C be an entry inside the z-line but not at any tooth. Let D be an entry outside the z-line. Let $\omega t(X)$ denote the weight of a node associated with the entry X. Then*

$$\omega t(D)>\omega t(A)\geqq\omega t(B)>\omega t(A)-\alpha\geqq\omega t(C).$$

Proof. (i) First inequality: Let A be at row k. If D is at row $\geqq k$, then the inequality is obvious. Assume D is above A. Let X be the leftmost entry outside the z-line and at the same row as D. Suppose X is at column j. Let Y be the entry just below the jth tooth. Then

$$\omega t(D)\geqq\omega t(X)\geqq\omega t(Y)$$

and $\omega t(A)=k\alpha$, $\omega t(Y)=(k+1-t_j)\alpha+j\beta$. From (1), $j\beta>(t_j-1)\alpha$, thus $\omega t(Y)>\omega t(A)$. Hence $\omega t(D)>\omega t(A)$.

(ii) From (1), $t_i\alpha\geqq i\beta$, thus $k\alpha\geqq(k-t_i)\alpha+i\beta$ for $i\geqq 0$. Hence $\omega t(A)\geqq\omega t(B)$.

(iii) From (1), $i\beta>(t_i-1)\alpha$, and we have $\omega t(B)>\omega t(A)-\alpha$.

(iv) Suppose C is at column j for some j. Let Z be the entry at the jth tooth. Then $\omega t(Z)\geqq\omega t(C)+\alpha$. By (ii), $\omega t(A)\geqq\omega t(Z)$. Thus $\omega t(A)-\alpha\geqq\omega t(C)$. □

4. Array truncation. We now describe the array truncation algorithm. We shall truncate the array using the z-line in a graphical manner so that a better insight to the algorithm is possible. At the end, the steps are summarized in an arithmetic form.

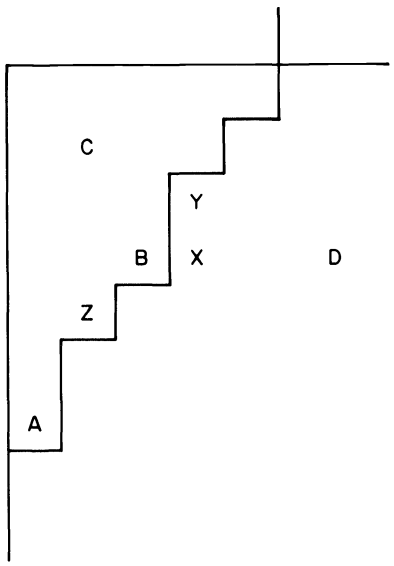


FIG. 7. *Illustration for Lemma* 1.

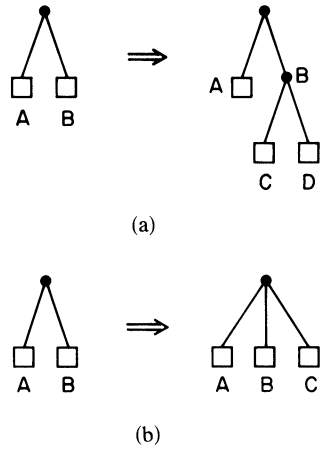


FIG. 8

The first step is to move the z-line down the array so that the 0th tooth is at row m where m is the smallest integer such that $N_m \geqq n$ and n is the number of leaves in the final tree. The entries below the z-line and not enumerated by N_m are set to zeros. The nodes associated with these entries will not be included in the final tree.

If we let all the nodes enumerated by N_m be leaves, we can construct a tree with N_m leaves. Clearly, it may have more leaves than we need, and furthermore, the tree is not optimal. We shall proceed to "locally optimize" the tree as we select n leaves from the entries near the teeth of the z-line.

Progressively we identify one new leaf at a time for the tree by marking the corresponding entry in the array, such that each time, the increase in the weight of the tree due to the additional leaf is minimal. Note that the increase in tree weight may or may not be equal to the weight of the new leaf added. For example, in Fig. 8a, adding a new leaf to B implies turning B into an internal node and creating two new leaves C and D. The weight of the tree is increased by $\omega t(B)+2\alpha+\beta$, which is neither equal to $\omega t(C)$ nor equal to $\omega t(D)$. On the other hand, adding a new leaf C as in Fig. 8b will incur an additional tree weight equal to $\omega t(C)$.

For a tooth at row i and column j, where $i = m - t_j$, let us examine the components of N_m (see Fig. 9). First of all, $e_1(i, j)$ of these nodes are the first children of their

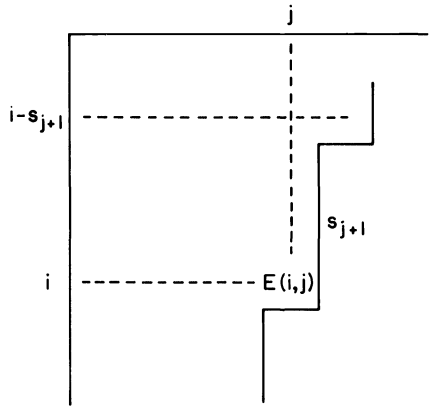


FIG. 9

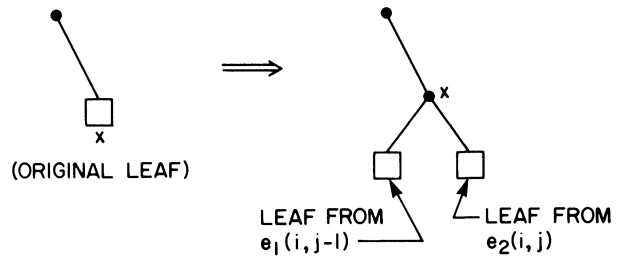
parents. Also by construction, none of the nodes enumerated by $e_2(i, j+1)$ will be present in the tree since they are below those enumerated by N_m. (Recall $e_1(i, j) = e_2(i, j+1)$ by definition.) It follows that each of the $e_1(i, j)$ nodes is the only child of its parent. If these nodes are leaves, we can delete them and let their parents be leaves, i.e., nodes enumerated by $e(i-1, j)$. This will reduce the weight of the tree by $e_1(i, j) \cdot \alpha$ without changing the number of leaves. Similarly, we will not select any node from $e_2(i-k, j+1)$ to be a leaf for $k = 1, 2, \cdots, s_{j+1}-1$, and for the time being we will also not select any node from $e_2(i-s_{j+1}, j+1)$ to be a leaf. Consequently, we change $e_1(i-k, j)$ to zero for $k = 0, 1, \cdots, s_{j+1}$, and mark $e_1(i-s_{j+1}-1, j)$, $e_2(i-k, j)$, $e_3(i-k, j)$ to be leaves for $k = 1, 2, \cdots, s_{j+1}+1$. Note that the sum of all these leaves is exactly $e_1(i, j)$. Also note that the weight of each of these leaves is at most $(i-1)\alpha + j\beta$ because $s_{j+1} \geqq 1$.

Next, consider $e_3(i, j)$ for the tooth at row i and column j. Each of the nodes enumerated by $e_3(i, j)$ has a weight of $(i\alpha + j\beta)$. Since this node is the third or further child of its parent, $(i\alpha + j\beta)$ is also the *additional weight* to the tree. We shall mark $e_3(i, j)$ to be leaves. After we mark $e_3(i, j)$ to be leaves for all teeth, we obtain an array corresponding to a tree of N_{m-1} leaves where

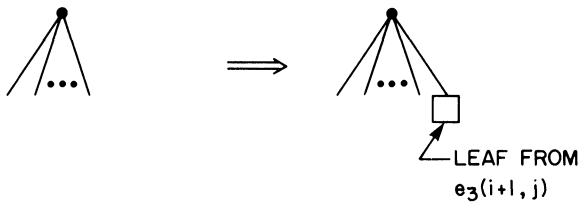
$$N_{m-1} = \sum_{j=0}^{h(m)-1} [e_1(m-t_j, j) + e_3(m-t_j, j)] < n.$$

We shall add $n - N_{m-1}$ more leaves to the tree one by one, each time choosing the node which increases the weight of the tree by a minimum. These leaves will be selected from those enumerated by $e_2(i, j)$ or $e_3(i+1, j)$ for teeth at row i and column j, $j = 0, 1, \cdots, h(m)-1$.

If we select a node from $e_2(i, j)$ as a leaf, we also have to turn one of the leaves from $e(i-1, j-1)$ back to an internal node and mark one of the nodes from $e_1(i, j-1)$ to be a leaf. This is illustrated in Fig. 10a. Thus, adding a leaf from $e_2(i, j)$ incurs an



(a) *Adding a leaf from* $e_2(i, j)$.



(b) *Adding a leaf from* $e_3(i+1, j)$.

FIG. 10

additional tree weight of $(i+1)\alpha+j\beta$. On the other hand, if we add a leaf from $e_3(i+1,j)$ to the existing tree, the leaves of the existing tree need not be changed (Fig. 10b). Thus the additional cost is also $(i+1)\alpha+j\beta$. Therefore, there are $e_2(i,j)+e_3(i+1,j)$ nodes each bearing an additional weight of $(i+1)\alpha+j\beta$.

If we let these nodes at all the teeth be leaves, we obtain a tree of N_m leaves, which may be larger than n. Therefore we shall select $n-N_{m-1}$ of these nodes to be leaves in the order of increasing additional weight. Let us consider the case $\beta>\alpha$ first. Note that at the jth tooth, $i=m-t_j$; thus the additional weight is $u_j=(m-t_j+1)\alpha+j\beta$. Let

$$d_j=\left\lceil\frac{j\beta}{\alpha}\right\rceil-\frac{j\beta}{\alpha}. \tag{5}$$

Then $u_j\geqq u_p$ if and only if $d_j\leqq d_p$. Thus, we can order the teeth by decreasing d_j. We refer to this partial ordering as *priority*, which is a function of j. Clearly, the 0th tooth has the lowest priority. Note that $d_j=d_{j+q}$ for all j, where q is the smallest positive integer such that $t_q\alpha=q\beta$. If β/α is an integer, then all teeth have the same priority.

We now follow this priority on j, $0\leqq j\leqq h(m)-1$, to add leaves to the tree. For each j (each tooth), we first mark $e_2(m-t_j,j)$ to be leaves, change $e(m-t_j-1,j-1)$ back to internal nodes, and restore and mark $e_1(m-t_j,j-1)$ to be leaves. We then for the same j mark $e_3(m-t_j+1,j)$ to be leaves as well. This is possible because $e_2(m-t_j+1,j-1)$ has already been assigned either as internal nodes or as leaves due to the following. Note that $e_2(m-t_j+1,j-1)$ can also be represented by $e_2(m-t_{j-1}-s_j+1,j-1)$. If $s_j>1$ then $e_2(m-t_{j-1}-s_j+1,j-1)$ has been marked as leaves in the previous step. If $s_j=1$, then the $(j-1)$st tooth has a higher priority than the jth tooth:

$$d_{j-1}-d_j=\frac{\beta}{\alpha}-S_j>0$$

since $\beta>\alpha$. So $e_2(m-t_{j-1}-s_j+1,j-1)$ will be marked as leaves before $e_3(m-t_j+1,j)$ will. This loop on j is continued until n leaves are marked.

For the case $\beta=\alpha$, the basic operation is the same except it will be done in a different order. First $e_2(m-t_j,j)$ are marked to be leaves together with changing $e(m-t_j-1,j-1)$ and marking $e_1(m-t_j,j-1)$ as in the previous case. This is done for all j in *any* order. If more leaves are still needed, $e_3(m-t_j+1,j)$ are then marked to be leaves for all j, again in any order.

For both cases, the process stops when n leaves are marked. For the last visited entry in the process, if only p leaves are needed then p of the nodes are marked. In particular, if $p<e_2(m-t_j,j)$, then we change p of $e(m-t_j-1,j-1)$ nodes to internal, change both $e_1(m-t_j,j-1)$ and $e_2(m-t_j,j)$ to p leaves each. If on the other hand, $p<e_3(m-t_j+1,j)$, then we change $e_3(m-t_j+1,j)$ to p leaves.

Finally, we change the e_2 component of all unvisited teeth to zero. All entries below the z-line are assumed to be zero except those e_3 components marked as leaves. We now obtain an array which corresponds to an optimal tree. The weight of the tree is the sum of the weight of the leaves. Proof of optimality will be given later.

Notice that, in reality, there is no need to first construct a semi-infinite array and then truncate it using the z-line. Each array entry can be computed from the previous ones when the entry is needed. Furthermore, the algorithm only uses entries close to the z-line. Therefore, only these entries are saved. It is not necessary to store the entire array, and no "truncation" is actually performed. Furthermore, the actual marking of the array entries as leaves is also unnecessary. It is included here to provide a better insight to the algorithm. What is required, though, is the count of the total

number of leaves selected and the new values of the modified entries. When a tree is finally constructed from the "truncated" array, the nodes without descendants are naturally leaves.

The algorithm is now summarized below. Any "unused" entry is considered to be zero. The operations with asterisk * are only conceptual and can be ignored in implementation.

ALGORITHM *Construction of α-β leaf trees.*

Step 1. Find the smallest m such that $N_m \geqq n$ (see (4)).

Step 2. Set the leaf count to N_{m-1}. For each j, $0 \leqq j \leqq h(m)-1$:

(1) Change $e_1(m-t_j-k, j)$ to zero for $k = 0, 1, \cdots, s_{j+1}$.

(2)* Mark $e_1(m-t_j-s_{j+1}-1, j)$, $e_2(m-t_j-k, j)$, $e_3(m-t_j-k, j)$, $e_3(m-t_j, j)$ to be leaves for $k = 1, 2, \cdots, s_{j+1}+1$.

Step 3

(a) *For $\beta > \alpha$.* Follow the order on j defined by decreasing d_j, $0 \leqq j \leqq h(m)-1$ (see (5)). For each j do the following (until n leaves are counted):

(1)* Change $e(m-t_j-1, j-1)$ back to internal nodes.

(2) Restore $e_1(m-t_j, j-1)$ from zero to its original value.

(3)* Mark $e_1(m-t_j, j-1)$, $e_2(m-t_j, j)$, $e_3(m-t_j+1, j)$ to be leaves.

(4) Increase leaf count by $e_2(m-t_j, j)$ and $e_3(m-t_j+1, j)$.

For the j when the nth leaf is counted, let p be the number of leaves needed for this j:

(5) If $p \leqq e_2(m-t_j, j)$ then set both $e_1(m-t_j, j-1)$ and $e_2(m-t_j, j)$ to p, and set $e_3(m-t_j+1, j)$ to zero.

(6) Otherwise restore $e_1(m-t_j, j-1)$ to its original value and set $e_3(m-t_j+1, j)$ to $p-e_2(m-t_j, j)$.

(7)* Same as (3)* and (4).

For each of the remaining j after n leaves are counted:

(8) Set both $e_2(m-t_j, j)$ and $e_3(m-t_j+1, j)$ to zero.

(b) *For $\beta = \alpha$.*

(1) Follow any order on j, $0 \leqq j \leqq h(m)-1$. For each j do (1)*, (2), (3)*, (4) of (a) except that $e_3(m-t_j+1, j)$ is not marked and counted as leaves.

(2) Again follow any order on j. For each j count and mark $e_3(m-t_j+1, j)$ to be leaves.

(3) If the nth leaf is counted in (1) then do (4) of (a) and set $e_2(m-t_j, j)$ or $e_3(m-t_j+1, j)$ to zero for the remaining j in (1) and (2).

(4) If the nth leaf is counted in (2) then do (5) of (a) and set $e_3(m-t_j+1, j)$ to zero for the remaining j in (2).

Step 4. Construct a tree from the array (to be discussed in the next section).

An example of a truncated array is shown in Fig. 11, where $n = 93$, $r = 4$, $\alpha = 3$, $\beta = 4$. The circled entries are those marked for leaves. An entry with a slash across it and a number at its upper right-hand corner means this entry has been changed to the number in the final array. I, II, III denote the priority of the teeth.

In this case, we start with $N_8 = 58$ and $N_9 = 102$. Thus $m = 9$, $h(m) = 7$, and we have the z-line denoted by the heavy line in Fig. 11. To illustrate Step 2, we consider the tooth at column 3. $e_1(5, 3) = 20$ and we change the entries $e_1(3, 3)$, $e_1(4, 3)$ and $e_1(5, 3)$ to zero. Entries $e_2(4, 3)$, $e_3(4, 3)$, $e_2(3, 3)$, $e_3(3, 3)$ are marked for leaves and $e_1(2, 3)$, $e_2(2, 3)$, $e_3(2, 3)$ are temporarily marked for leaves. $e_3(5, 3)$ is also marked for leaves. After we do this for all teeth, we have marked a total of $N_8 = 58$ nodes for leaves. Thirty-five more leaves are needed.

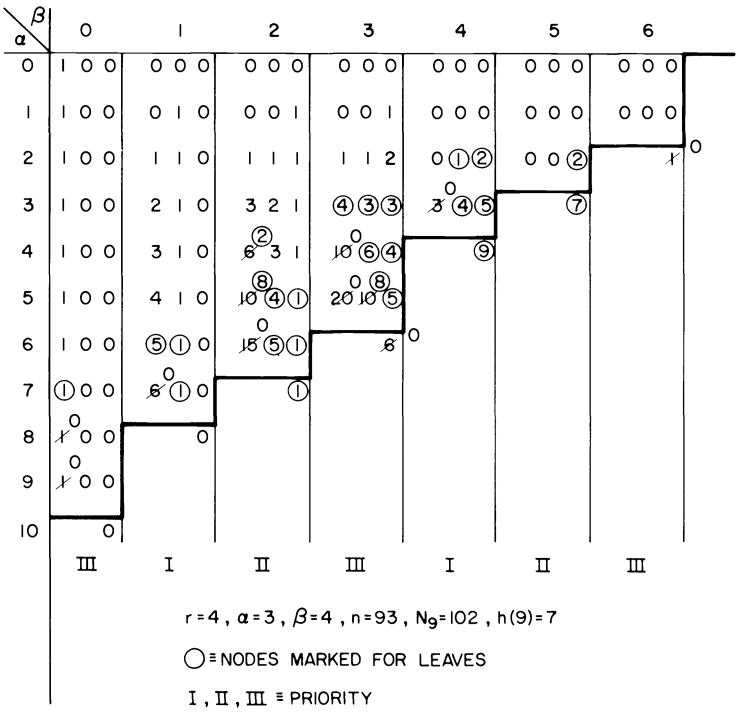


FIG. 11

Step 3 selects leaves from $e_2(i, j)$ at the teeth and from $e_3(i+1, j)$. Since $\beta = 4$ and $\alpha = 3$, columns 1, 4 have priority I, 2, 5 have II, and 0, 3, 6 have III. All the nodes from $e_2(i, j)$ and $e_3(i+1, j)$ in I and II are selected, and 8 from $e_2(5, 3)$ in priority III are selected to make up a total of 93. When a node from $e_2(i, j)$ is selected for a leaf, a node from $e_1(i, j-1)$ must also be selected for a leaf, and one node from $E(i-1, j-1)$ must be turned into an internal node. For example, 8 nodes are selected from $e_2(5, 3) = 10$ as leaves. Thus, we must select 8 of $e_1(5, 2) = 10$ and 2 of $e(4, 2) = 10$ nodes as leaves.

5. Construction of the tree. In this section we shall construct a tree from the "truncated" array obtained in § 4. The entry $e_1(0, 0) = 1$ corresponds to the root of the tree. We shall examine the array one row at a time starting from row 1, and examine individual entries from left to right on that row. For an entry $E(i, j)$ in the array, $e(i, j)$ nodes will be added to the tree. Pointers will be maintained so that these nodes can later be located when i, j and component subscript (1, 2 or 3) are given.

For an entry $e_1(i, j)$, we add the first child to that many nodes associated with $E(i-1, j)$. This is always possible because by construction and truncation of the array, $e_1(i, j) \leqq e(i-1, j)$; see Fig. 12a.

For an entry $e_2(i, j)$, we add the second child to the parents of the nodes associated with $e_1(i, j-1)$. This is possible because $e_2(i, j) = e_1(i, j-1)$ for all i, j; see Fig. 12b.

For an entry $e_3(i, j)$, we add the third child to the nodes associated with $E(i-1, j-2)$, the fourth child to those associated with $E(i-1, j-3), \cdots$, and the rth child to those associated with $E(i-1, j-r+1)$. There are enough parents because

$$e_3(i, j) \leqq \sum_{k=2}^{r-1} e(i-1, j-k).$$

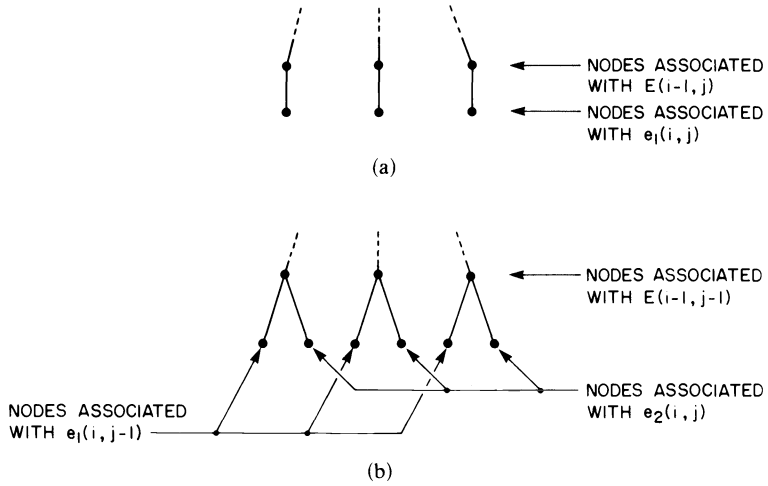


FIG. 12

But what about the existence of the previous children under the same parents? The only case that needs to be considered is when $e_2(i, j-1)$ is at a tooth and $e_3(i, j)$ is just below a tooth, such that the latter may be selected for leaves while the former may not. This case has been proved in § 4.

The process of adding nodes to the tree continues until all nonzero array entries are exhausted. The final tree is an optimal α-β leaf tree of degree r. For the example in Fig. 11, the result is shown in Fig. 13.

Some of the properties of the tree can be observed directly from the truncated array. Since $e_2(i, 0) = e_3(i, 0) = 0$ for all i, we can easily show that the number of levels (depth) of the tree is at most $m - s_1$, which is equal to $m - \lceil \beta/\alpha \rceil$. Furthermore, the

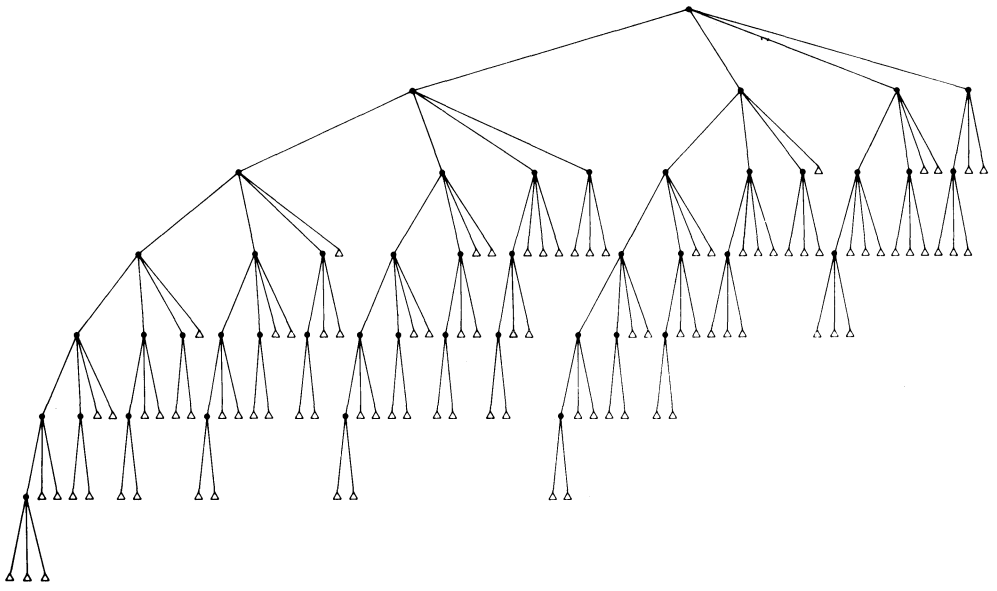


FIG. 13. *An optimal tree of degree* 4 *with* 93 *leaves for* $\alpha = 3$, $\beta = 4$.

number of nodes at level i of the tree is equal to

$$\sum_{j=0}^{h(m)-1} e(i,j),$$

where $e(i,j)$ refers to the entry of the truncated array. Since the weight of a node represented by an entry at row i and column j in the array is $i\alpha+j\beta$, the weight of the final tree can easily be computed during the construction using the coordinates of those entries marked as leaves.

6. Proof of optimality. We prove here that the tree constructed in §§ 4 and 5 is indeed an optimal α-β leaf tree. We shall first prove that the tree thus constructed is locally optimal in the sense that: (1) every internal node has at least two children, and (2) subtracting a leaf from the tree followed by adding a leaf to it will not reduce the total weight. We then show that a locally optimal tree is indeed optimal. We reiterate a few definitions here and introduce some new definitions.

Let T be an α-β leaf tree with degree r, $r \geqq 2$, (*tree* for short) and a finite number of leaves $L(T)$. The *path* of a node A in T is a finite sequence of integers $i_1, i_2, \cdots, i_m$ where m is the depth of A in T and i_j represents the edge leading from the particular node at the $(j-1)$st level to its i_jth child ($1 \leqq i_j \leqq r$) at the jth level of T. The 0th level of T contains the root only. Clearly, A can be uniquely identified by its path in T, and vice versa. The *weight of* A is given by $\omega t(A) = \sum_{j=1}^{m} [\alpha + (i_j - 1)\beta]$. The weight of the root is zero. The *weight of* T is given by $\omega t(T) = \sum \omega t(x)$, where x runs over all leaves in T. T is said to be *optimal* if $\omega t(T) \leqq \omega t(S)$ for all trees S such that $L(S) = L(T)$. To *subtract a leaf* from T we decrease $L(T)$ by one by removing a leaf directly, and, if this results in an internal node with a single child, we remove the single child as well, and make the internal node a new leaf. To *add a leaf* to T we increase $L(T)$ by one by either adding an extra child to an internal node currently with less than r children, or adding two children to an existing leaf thereby making it internal. T is said to be *local optimal* if (i) every internal node of T has at least two children, and (ii) subtracting a leaf from T followed by adding a leaf to it will not reduce $\omega t(T)$ regardless of the particular leaves being modified.

Given two trees U and V, a node A in U is said to *correspond* to a node B in V if the path of A in U is identical to the path of B in V. We may therefore denote B by $V:A$. Clearly $\omega t(A) = \omega t(V:A)$. Next we extend this notion to include nodes outside the trees, for example, B can be a node outside V. In this case B is said to be *nonexistent* in V. Thus, $V:A$ can be either a leaf, an internal node, or nonexistent in V. The *distance between* A *and* $V:A$ is defined as

$$d(A, V:A) = \begin{cases} 0 & \text{if both } A \text{ and } V:A \text{ are leaves or if both are nonexistent,} \\ |c(A) - c(V:A)| & \text{if both } A \text{ and } V:A \text{ are internal nodes,} \\ r & \text{otherwise,} \end{cases}$$

where $c(A)$ is the number of children that A has in U and $c(V:A)$ is the number of children that $V:A$ has in V. The *distance between* U *and* V is defined as

$$\eta(U, V) = \sum d(x, V:x),$$

where the summation is over all nodes x in U and all nodes $V:x$ in V. Clearly $d(A, V:A) = d(V:A, A)$ and $\eta(U, V) = \eta(V, U)$. U is said to be *identical* to V,

denoted by $U = V$, if $V:x$ exists in V for any node x in U and $U:y$ exists in U for any node y in V. Clearly $U = V$ if and only if $V = U$.

THEOREM 1. *The tree constructed by the Algorithm, described in §§ 4 and 5, is local optimal.*

Proof. By construction, $e_1(i,j) = e_2(i,j+1) \leqq e(i-1,j)$ for $i > 0$ and $j \geqq 0$. It follows that for the set of nodes represented by $e(i-1,j)$, if they have p first children among them then they also have p second children. Since this is true for all nodes and since a node having a second child must have a first child, it is obvious that every internal node of the tree must have at least two children.

Suppose we subtract a leaf from the tree, which must be the rightmost leaf of its parent. Since an internal node has at least two children, this leaf corresponds to either $e_2(i,j)$ or $e_3(i,j)$ in the array for some i, j. For the former case, the reduction in the weight of the tree is $(i+1)\alpha + j\beta$. Since this entry is above the z-line, we have $i \leqq m - t_j$. Therefore the maximum reduction in tree weight is $(m - t_j + 1)\alpha + j\beta$ for some j. For the latter case, i.e., $e_3(i,j)$, the entry is either above the z-line or immediately below it. According to Lemma 1, any entry below the z-line corresponds to a larger weight than any entry above the z-line. Therefore $i = m - t_j + 1$, and the maximum reduction in tree weight is also $(m - t_j + 1)\alpha + j\beta$ for some j. Let the Jth tooth be the one in which the nth leaf of the tree was counted. Then the maximum reduction in tree weight is equal to $(m - t_J + 1)\alpha + J\beta$.

Suppose we now add a leaf to the tree. This can be by either (i) adding two leaves under an existing leaf or (ii) adding another leaf under an existing internal node which currently has less than r children.

(i) The minimum additional weight is achieved by adding two leaves under a leaf enumerated by $E(m - t_j - S_{j+1} - 1, j)$ for some j. This weight is $(m - t_j - S_{j+1} + 1)\alpha + (j+1)\beta = (m - t_{j+1} + 1)\alpha + (j+1)\beta$ for some j, $= (m - t_k + 1)\alpha + k\beta$ for some k.

(ii) The only possible case here is to add a leaf which is enumerated by $e_3(m - t_j + 1, j)$ for some j. The additional weight is therefore also

$$(m - t_j + 1)\alpha + j\beta \quad \text{for some } j.$$

For both cases, this additional weight is minimized by $j = J$.

Therefore subtracting a leaf from the tree followed by adding a leaf does not reduce the tree weight. □

LEMMA 2. *An optimal tree with a finite number of leaves is always local optimal.*

Proof. Given an optimal tree T. Suppose T has an internal node A with a single child B. Then we can remove B and hang the subtree under B directly under A. This will reduce the weight of each leaf in that subtree by α, and so will reduce $\omega t(T)$. Since T is optimal, we cannot have such an internal node A. Furthermore, by subtracting a leaf from T and then adding a leaf to T, we obtain a new tree S with the same number of leaves as T. Since T is optimal, we have $\omega t(S) \geqq \omega t(T)$, regardless of the particular leaves subtracted or added. □

LEMMA 3. *Given two trees U and V such that $U = V$, every leaf of U corresponds to a leaf in V, and every internal node of U corresponds to an internal node in V with the same number of children. Furthermore, $L(U) = L(V)$ and $\omega t(U) = \omega t(V)$.*

Proof. Let x be a leaf in U. Since $U = V$, $V:x$ exists in V. If $V:x$ is internal, let y be a child of $V:x$. $U = V$ implies $U:y$ exists in U. This means x has a child and cannot be a leaf. Therefore $V:x$ must be a leaf. Now let x be an internal node of U with p children. Since $U = V$, $V:x$ exists in V, and so do the p corresponding nodes of the children of x. Therefore $V:x$ is an internal node with at least p children. If $V:x$ has more than p children, then the $(p+1)$st child of $V:x$ must correspond to

the $(p+1)$st child of x. By contradiction, we know $V:x$ must have exactly p children. Since every leaf in U corresponds to a leaf in V and vice versa, clearly we have $L(U)=L(V)$ and $\omega t(U)=\omega t(V)$. □

LEMMA 4. *Given two trees U and V,* (i) $\eta(U, V)\geqq 0$ *and* (ii) $\eta(U, V)=0$ *if and only if* $U=V$.

Proof. Statement (i) is obvious from the definition of η.

Suppose $U=V$. By Lemma 3, we know $d(x, V:x)=0$ for any node x in U, and $d(U:y, y)=0$ for any node y in V. Therefore $\eta(U, V)=0$. Now suppose $\eta(U, V)=0$. By the definition of d, we know d cannot be negative. So we have $d(x, V:x)=d(U:y, y)=0$ for any node x in U and any node y in V. This implies that $V:x$ exists in V for any x in U and $U:y$ exists in U for any y in V. Therefore $U=V$. □

LEMMA 5. *Given two local optimal trees U and V in which there exist an internal node A and a leaf B in U such that $V:A$ is a leaf and $V:B$ is an internal node of V (we assume B is not in the subtree under A). If U is optimal then there exists an optimal tree W such that $L(U)=L(W)$ and $\eta(W, V)<\eta(U, V)$. If V is optimal then there exists an optimal tree W such that $L(W)=L(V)$ and $\eta(W, U)<\eta(U, V)$ (see Fig.* 14a).

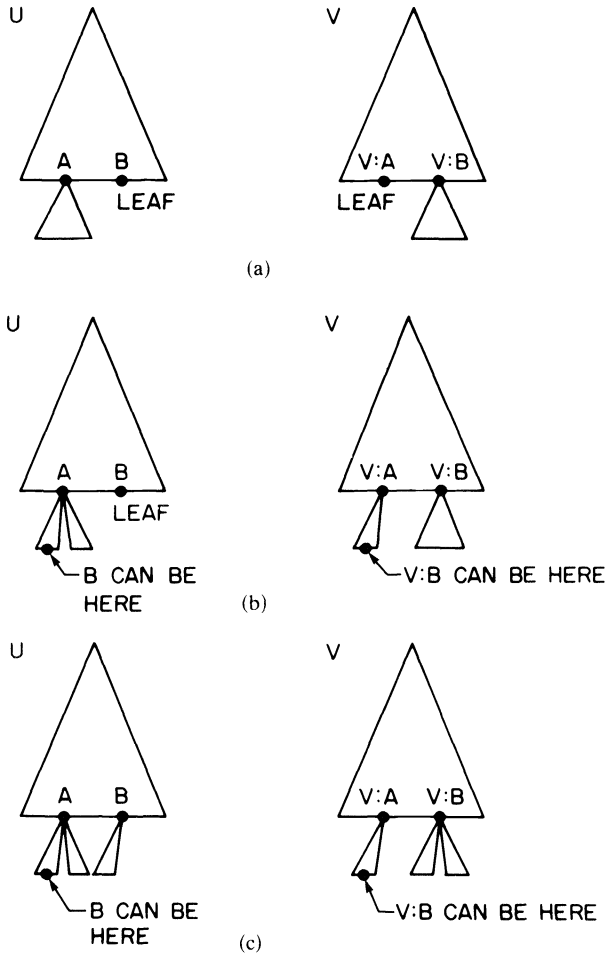


FIG. 14

Proof. Since both U and V are local optimal, the subtree under A and the subtree under $V:B$ each must have at least two leaves. If we subtract one leaf under A, we reduce $\omega t(U)$ by at least $\omega t(A)+2\alpha+\beta$. If we then add a leaf by inserting 2 nodes under B, we increase $\omega t(U)$ by $\omega t(B)+2\alpha+\beta$. Since U is local optimal, we have

$$\omega t(B)+2\alpha+\beta \geqq \omega t(A)+2\alpha+\beta,$$

or

$$\omega t(A) \leqq \omega t(B).$$

On the other hand, we can subtract a leaf under $V:B$ and add one under $V:A$. This implies

$$\omega t(V:A) \geqq \omega t(V:B), \quad \text{or} \quad \omega t(A) \geqq \omega t(B).$$

Consequently

$$\omega t(A) = \omega t(B).$$

If U is optimal, we can remove the entire subtree from under A to under B without affecting the optimality or the number of leaves of U. But the distance between the new optimal tree and V is smaller than $\eta(U, V)$. Similarly, we can move the entire subtree from under $V:B$ to under $V:A$ if V is optimal. □

LEMMA 6. *Given two local optimal trees U and V in which there exist an internal node A and a leaf B in U (B can be in the subtree under A) such that $V:A$ and $V:B$ are internal nodes of V and such that A has more children than $V:A$ has. If U is optimal then there exists an optimal tree W such that $L(W)=L(U)$ and $\eta(W, V)<\eta(U, V)$. If V is optimal then there exists an optimal tree W such that $L(W)=L(V)$ and $\eta(W, U)<\eta(U, V)$ (see Fig. 14b).*

Proof. Let p be the number of children that $V:A$ has. If B is in the subtree under A, then $V:B$ must be one of the p children of $V:A$ or in one of their subtrees. Since V is local optimal, $p \geqq 2$. If we subtract a leaf from the subtree under the $(p+1)$st child of A (note that B cannot be in this particular subtree) and add an extra leaf under B, we first reduce $\omega t(U)$ by at least $\omega t(A)+\alpha+p\beta$ and then increase $\omega t(U)$ by $\omega t(B)+2\alpha+\beta$. Since U is local optimal, we have

$$\omega t(B)+2\alpha+\beta \geqq \omega t(A)+\alpha+p\beta.$$

On the other hand, if we subtract a leaf from the subtree under $V:B$ and add the $(p+1)$st child to $V:A$ as a new leaf, we first reduce $\omega t(V)$ by $\omega t(B)+2\alpha+\beta$ or more and then increase $\omega t(V)$ by $\omega t(A)+\alpha+p\beta$. Since V is local optimal, we have

$$\omega t(A)+\alpha+p\beta \geqq \omega t(B)+2\alpha+\beta.$$

Therefore $\omega t(A)+\alpha+p\beta=\omega t(B)+2\alpha+\beta$. This implies that if U is optimal, we can subtract a leaf from the subtree under the $(p+1)$st child of A and add a leaf under B without affecting the optimality or the number of leaves of U. But the distance between the new optimal tree and V is smaller than $\eta(U, V)$. Similarly, we can subtract a leaf under $V:B$ and add the $(p+1)$st child to $V:A$ if V is optimal. □

LEMMA 7. *Given two local optimal trees U and V in which there exist internal nodes A and B in U (B can be in the subtree under A) such that $V:A$ and $V:B$ are internal nodes of V. Furthermore, A has more children than $V:A$ has and $V:B$ has more children than B has. If U is optimal then there exists an optimal tree W such that $L(W)=L(U)$ and $\eta(W, V)<\eta(U, V)$. If V is optimal then there exists an optimal tree W such that $L(W)=L(V)$ and $\eta(W, U)<\eta(U, V)$ (see Fig. 14c).*

Proof. Let p be the number of children that $V:A$ has and let q be the number of children that B has. If B is in the subtree under A, then $V:B$ must be one of the p children of $V:A$ or in one of their subtrees. Since U and V are local optimal, we have $p \geqq 2$ and $q \geqq 2$. Suppose we subtract a leaf from the subtree under the $(p+1)$st child of A (B cannot be in this particular subtree) and add the $(q+1)$st child under B as a new leaf, we first reduce $\omega t(U)$ by at least $\omega t(A)+\alpha+p\beta$ and then increase $\omega t(U)$ by $\omega t(B)+\alpha+q\beta$. Since U is local optimal, we have $\omega t(B)+\alpha+q\beta \geqq \omega t(A)+\alpha+p\beta$. Similarly, we can subtract a leaf from the subtree under the $(q+1)$st child of $V:B$ and add the $(p+1)$st child under $V:A$ as a new leaf. This shows

$$\omega t(A)+\alpha+p\beta \geqq \omega t(B)+\alpha+q\beta.$$

Therefore $\omega t(A)+\alpha+p\beta=\omega t(B)+\alpha+q\beta$. If U is optimal, we can remove a leaf from the subtree under the $(p+1)$st child of A and add the $(q+1)$st child under B. We then obtain a new optimal tree W such that $L(W)=L(U)$ and $\eta(W, V)<\eta(U, V)$. Similarly we can remove a leaf from the subtree under the $(q+1)$st child of $V:B$ and add the $(p+1)$st child under $V:A$ if V is optimal. □

LEMMA 8. *Given trees U and V such that $L(U)=L(V)=n$ and $U \neq V$, where n is finite. Furthermore, U is optimal and V is local optimal. Then there exists an optimal tree W such that $L(W)=n$ and $\eta(W, V)<\eta(U, V)$.*

Proof. Since $U \neq V$ and $L(U)=L(V)$, there exists a leaf x in U such that $V:x$ is not a leaf in V. So $V:x$ is either an internal code (Case (i)) or nonexistent (Case (ii)).

Case (i). *$V:x$ is an internal node in V.* Let x be B. Since V is local optimal, the subtree under $V:B$ must have more than one leaf. Since $L(U)=L(V)$, if we disregard B and the subtree under $V:B$, we know there must exist another leaf y in U different from B such that $V:y$ is not a leaf in V. If $V:y$ is again an internal node in V, then y is in the same situation as x and we can repeat Case (i) endlessly. But since n is finite, at some point in the recursion $V:y$ must be nonexistent in V for some leaf y in U different from B.

Now consider the parent, grandparent, $\cdots$, of this y until we find ancestor A of y such that $V:A$ exists in V. $V:A$ is either a leaf or an internal node. If $V:A$ is a leaf, then A cannot be an ancestor of B and we can apply Lemma 5. If $V:A$ is internal, then A has more children than $V:A$ has and we can apply Lemma 6. For either situation, we can find an optimal tree W such that $L(W)=n$ and $\eta(W, V)<\eta(U, V)$.

Case (ii). *$V:x$ is nonexistent in V.* Let us consider the parent, grandparent, $\cdots$, of x until we find an internal node y of U such that $V:y$ exists in V. If $V:y$ is a leaf, then we have found a leaf in V and the corresponding node is internal in U. It is the same situation as x and $V:x$ in Case (i) except the roles of U and V are now interchanged. Since we have used only local optimality in proving Case (i) and since Lemmas 5 and 6 are symmetric with respect to which tree is optimal, we can apply the same proof to this situation of Case (ii). So we only need to consider the case where $V:y$ is an internal node of V, and let y be A. Note that A has more children than $V:A$ has. Because $L(U)=L(V)$ and x is a leaf in U but $V:x$ is not in V, V must have a leaf z such that $U:z$ is not a leaf of U.

If $U:z$ is an internal node of U, then this becomes Case (i) again. If $U:z$ is not in U, then we can consider the parent, grandparent, $\cdots$, of z until we find a node $V:B$ such that B exists in U. If B is a leaf, then we can apply Lemma 6. If B is internal, then $V:B$ has more children than B has and we can apply Lemma 7. For either situation, we can find an optimal tree W such that $L(W)=n$ and $\eta(W, V)<\eta(U, V)$. □

THEOREM 2. *Given a local optimal tree V with a finite number of leaves, V is optimal.*

Proof. Let U be an optimal tree such that $L(U)=L(V)$. If $U=V$ then V is optimal because of Lemma 3. Otherwise by Lemma 4 we know $\eta(U, V)>0$. Because of Lemma 8, we can find optimal trees $W_1, W_2, \cdots, W_m$ such that $L(W_1)=L(W_2)=\cdots=L(W_m)=L(U)$ and $\eta(U, V)>\eta(W_1, V)>\eta(W_2, V)>\cdots>\eta(W_m, V)=0$, where m is finite. This implies $W_m=V$ (Lemma 4) and so V is optimal (Lemma 3). □

7. Computation time and storage. In order to estimate the computation time, we need to estimate m.

LEMMA 9. *In the tree array, $e(i, j)=0$ for $i\geqq 0$ and $j\geqq i(r-1)+1$.*

Proof. Suppose there exists some $I\geqq 0$ such that $e(I, j)=0$ for all $j\geqq I(r-1)+1$. Then

$$\begin{aligned} e(I+1, (I+1)(r-1)+1+s) &= \sum_{k=0}^{r-1} e(I, (I+1)(r-1)+1+s-k) \\ &= \sum_{k=0}^{r-1} e(I, I(r-1)+1+s+(r-1-k)) \\ &= 0 \qquad \text{for all } s\geqq 0. \end{aligned}$$

That is, $e(I+1, j)=0$ for all $j\geqq(I+1)(r-1)+1$. Since the hypothesis is true for $I=0$, the lemma is proved by induction. □

LEMMA 10. *Let $f(i)$, $i\geqq 0$, be the sum of all entries at row i of the tree array:*

$$f(i)=\sum_{j=0}^{i(r-1)} e(i, j)=\sum_{j=-\infty}^{\infty} e(i, j).$$

Then $f(i)=r^i$.

Proof. Given $f(i)$ for some $i\geqq 0$, then

$$\begin{aligned} f(i+1) &= \sum_{j=-\infty}^{\infty} e(i+1, j) \\ &= \sum_{j=-\infty}^{\infty} \sum_{k=0}^{r-1} e(i, j-k) = \sum_{k=0}^{r-1} f(i) = r\cdot f(i). \end{aligned}$$

Since $f(0)=1$, we have $f(i)=r^i$ for $i\geqq 0$. □

THEOREM 3. *Let m be the smallest k such that $N_k\geqq n$ where N_k is defined by* (4). *Then*

$$\frac{\alpha m}{\alpha+\beta(r-1)}<\log_r(n)<m.$$

Therefore, $m=O(\log n)$ assuming α, β, and r are constants.

Proof. Applying Lemmas 9 and 10 to Fig. 6, we obtain Fig. 15. Clearly $r^x<n<r^m$, or equivalently $x<\log_r(n)<m$. Because of similar triangles, we have

$$\frac{m-x}{x(r-1)}=\frac{m}{h(m)}=\frac{\beta}{\alpha}$$

allowing for round-off error for integers. This implies

$$x=\frac{\alpha m}{\alpha+\beta(r-1)}.$$

□

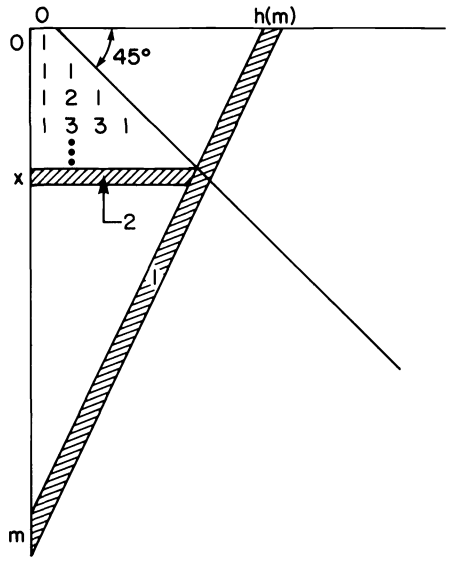


FIG. 15

The computation time, in terms of the order of magnitude, is listed below:

Step 1.	Compute entries	$\frac{m \cdot h(m)}{2}$
	Sum entries	$\frac{m \cdot h(m)}{2}$
Step 2.	$\sum_{j=0}^{h(m)-1} (S_j+1)$	$m+h(m)$
Step 3.		$h(m)$
Step 4.	Construct tree	$\frac{m \cdot h(m)}{2}+n.$

Since $h(m) \approx m\alpha/\beta$, then the total time is on the order of m^2+n assuming α and β are constants. Because $m = O(\log n)$, the total time is $O(n)$.

The storage requirement is roughly $\frac{1}{2} \cdot m \cdot h(m)$ for the array (although this can be reduced easily) and $\frac{1}{2} \cdot m \cdot h(m)+n$ for tree construction because of the pointer chains and storage of the tree. So the total storage is also $O(n)$.

If only the array of the optimal tree is needed (to get the properties of the optimal tree, for example), then both the computation time and storage will be $O(m^2) = O((\log n)^2)$.

8. Conclusions. In this paper, we consider the construction of optimal α-β leaf trees with degree r and edge weight $C_i = \alpha + (i-1)\beta$ for the edge to child i. This special form of edge weight enables us to map the trees to arrays and hence the resulting two-stage algorithm. In the first stage we select nodes from the array. In the second stage we construct the tree explicitly. The first stage (Steps 1–3 of the Algorithm) takes time $O((\log n)^2)$ and is independent of r. The second stage (Step 4) takes time $O(n)$, also independent of r.

In case the tree is not explicitly needed, e.g., when one is only interested in the cost of the optimal α-β leaf tree or other properties of the tree, the first stage, hence $O((\log n)^2)$ time, suffices.

Also, if *all* optimal trees are required, we can easily get them all from the array and generate them one by one.

Even though the algorithm is presented only for the case $\beta \geqq \alpha$, the same approach is equally applicable to the case $\beta < \alpha$. The only difference is in the definition of the z-line (Fig. 6). When $\beta \geqq \alpha$, we have a vertical z-line, which moves downwards (Fig. 6), whereas for $\beta < \alpha$, we have a horizontal z-line, which moves to the right (see [2, Fig. 5]). The algorithm and complexity are identical.

Finally, it should be pointed out that in [9], an algorithm for the general case, i.e. arbitrary C_i, is presented with running time $O(\min(r^2 n, rn \log n))$. The algorithm works in two phases: extension and mending. The extension phase alone produces an optimal tree if $C_1 + C_2 \leqq C_3$. It runs in time $O(\min(rn, n \log n))$. In fact, if the smallest elements of the r queues mentioned on the bottom of page 213 are kept in a heap, then the running time reduces to $O(n \log r)$.

For our problem, i.e. $C_i = \alpha + (i-1)\beta$, the condition $C_1 + C_2 \leqq C_3$ is equivalent to $\alpha \leqq \beta$. Hence the extension phase can produce an optimal tree in time $O(n \log r)$. However, for the other case $\alpha > \beta$ of our problem, the full two-phase algorithm is needed with running time $O(\min(r^2 n, rn \log n))$.

The savings in our algorithm are largely due to the *critical* fact that $C_i = \alpha + (i-1)\beta$, hence the possibility of separating the selection stage from the construction stage. In the selection stage, we can select a large number of nodes at a time from the array, which results in the more efficient running time $O((\log n)^2)$. On the other hand, the method in [9] integrates the selection and construction stages, and can only select one node at a time.

Acknowledgments. The authors are grateful to a referee for bringing reference [9] to their attention.

REFERENCES

[1] D. M. CHOY AND C. K. WONG, *Bounds for optimal α-β binary trees*, BIT, 17 (1977), pp. 1–15.

[2] ———, *Optimal α-β trees with capacity constraint*, Acta Inform., 10 (1978), pp. 273–296.

[3] L. E. STANFEL, *Optimal trees for a class of information retrieval problems*, Inform. Stor. Retr., 9 (1973), pp. 43–59.

[4] ———, *Optimal tree lists for information storage and retrieval*, Inform. Systems, 2 (1976), pp. 65–70.

[5] E. H. SUSSENGUTH, *Use of tree structures for processing files*, Comm. ACM, 6 (1963), pp. 272–279.

[6] Y. PATT, *Variable length tree structures having minimum average search time*, Comm. ACM, 12 (1969), pp. 72–76.

[7] R. KARP, *Minimum redundancy coding for the discrete, noiseless channel*, IRE Trans., IT-7 (1961), pp. 27–35.

[8] K. MEHLHORN, *An efficient algorithm for constructing nearly optimal prefix codes*, IEEE Trans. Inform. Theory, IT-26 (1980), pp. 513–517.

[9] Y. PERL, M. R. GAREY AND S. EVEN, *Efficient generation of optimal prefix code: Equiprobable words using unequal cost letters*, J. Assoc. Comput. Mach., 22 (1975), pp. 202–214.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0003 $01.25/0

OPTIMAL PLACEMENT FOR RIVER ROUTING*

CHARLES E. LEISERSON† AND RON Y. PINTER†‡

Abstract. Programs for integrated circuit layout typically have two phases: placement and routing. The router tries to produce as efficient a layout as possible, but of course the quality of the routing depends heavily on the quality of the placement. On the other hand, the placement procedure ideally should know the impact of its placement decisions on the quality of a routing. In this paper, we present a placement-and-routing problem for which there is perfect interaction between the two phases. The algorithms for this commonly arising problem are fast, simple and optimal.

River routing is the problem of connecting in order a set of terminals $a_1, \cdots, a_n$ on a line to another set $b_1, \cdots, b_n$ across a rectangular channel. The terminals are located on modules which must be placed relative to one another before routing. This placement-and-routing problem arises frequently in design systems like *bristle-blocks* where *stretch lines* through a module can effectively break it into several *chunks*, each of which may be placed separately. In this paper we give *concise* necessary and sufficient conditions for wirability which are applied to reduce the optimal placement problem to the graph-theoretic *single-source-longest-paths* problem. For rectilinear wiring, the special structure of graphs that arise allows an optimal solution to be determined quickly.

Key words. design automation, layout of integrated circuits, placement and routing, river routing, graph theory, longest paths, analysis of algorithms

1. Introduction.

River routing is a special routing problem which arises often in the design of integrated circuits, and it has been shown to be optimally solvable in polynomial-time for many wiring models (see in particular [1], [3], [10] and [11]). In this paper, we demonstrate that the placement problem for river routing is also polynomial-time solvable.

The general character of the placement problem for river routing is illustrated in Fig. 1. Two sets of terminals $a_1, \cdots, a_n$ and $b_1, \cdots, b_n$ are to be connected by

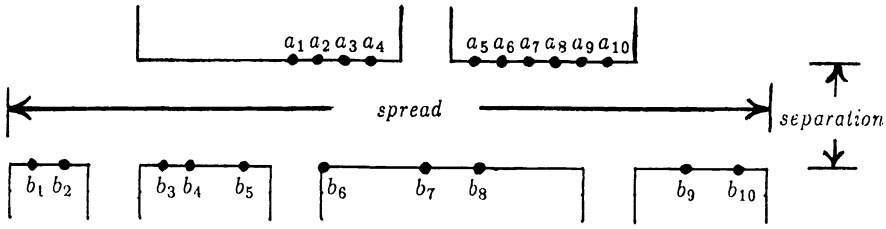


FIG. 1. *Two sets of chunks on either side of a rectangular channel. Terminal a_i must be connected to b_i for $i = 1, \cdots, 10$.*

wires across a rectangular channel so that wire i is routed from a_i to b_i. The terminals on each side of the channel are grouped into *chunks* which must be placed as a unit. The quality of a legal placement—one for which the channel can be routed—can be measured in terms of the dimensions of the channel. The *separation* is the vertical distance between the two lines of terminals, and the *spread* is the horizontal dimension of the channel.

* Received by the editors October 16, 1981 and in final revised form August 10, 1982. This research was supported in part by the Defense Advanced Research Projects Agency under contract N00014-80-C-0622.

† Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

‡ Present address: Bell Laboratories, Murray Hill, New Jersey 07974.

The *wiring model* gives the constraints that the routing must satisfy. Although our results can be generalized to include a variety of wiring models (see § 5), we concentrate on the (*one-layer*) *square-grid model.* Crossovers are disallowed in the square-grid model, and all wires must take disjoint paths through the grid.

The placement problem for river routing arises often during ordinary integrated circuit design. A common instance is when the terminals of one or more modules are to be connected to drivers. The various independent "chunks" are the modules, which lie on one side of the channel, and the drivers, which lie on the other.

A more interesting manifestation of the placement problem occurs in the context of design systems such as bristle-blocks [5] and DPL/Daedalus [2], [9]. These systems encourage a designer to build plug-together modules so that the difficulties associated with general routing can be avoided. A designer may specify *stretch lines* which run through a module and allow the module to be expanded perpendicular to the stretch line, as demonstrated in Fig. 2. When two independently designed modules are plugged together, stretch lines permit the terminals to be *pitch aligned*, that is, the distances between pairs of adjacent terminals are made to match the distances between their mates and routing is avoided because the separation of the channel is zero. Unfortunately, this approach may not succeed unless stretch lines are put between every pair

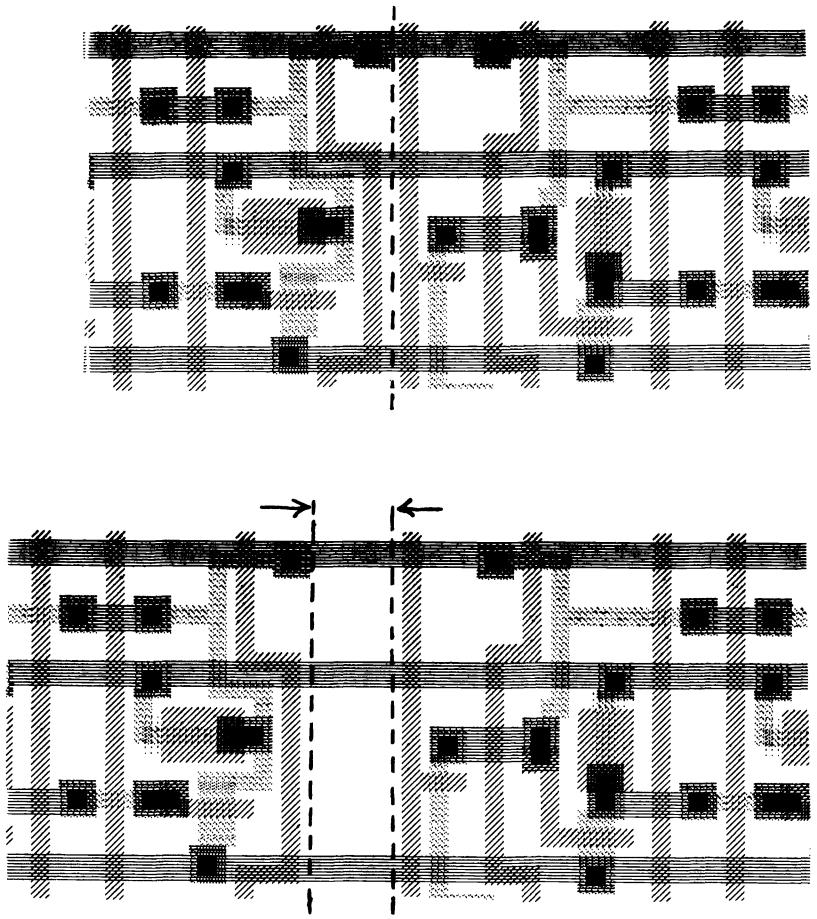


FIG. 2. *A module before and after stretching (courtesy of MIT Artificial Intelligence Laboratory).*

of adjacent terminals. The stretch lines may not only disrupt the internal structure of the modules, but the consequence may be an inordinate amount of stretching that leaves the channel with a large spread.

The other extreme is to forego stretching altogether and river route between the terminals. But the cost may still be large if a large separation is required in order to achieve a routing. A reasonable compromise is to place stretch lines where it is convenient and then do a little stretching and a little routing. Determining how much of each to do is exactly the placement problem for river routing.

The remainder of this paper demonstrates that optimal solutions to the placement problem can be achieved efficiently. Section 2 gives a concise necessary and sufficient condition for a channel to be routable in the square-grid model. Section 3 shows that the form of this condition allows the placement problem to be reduced to the graph-theoretic problem of finding the longest paths from a source vertex to all other vertices in a graph. Based on this problem reduction, a linear-time algorithm for optimal placement is given in § 4. Section 5 shows that the algorithm extends to wiring models other than the square-grid model, but its performance depends on the particular wirability conditions for the model. Section 6 discusses the application of our results to other routing situations and suggests further placement problems.

2. Necessary and sufficient conditions for wirability. To demonstrate the results of this paper, we adopt an extremely simple wiring model: the *(one-layer) square-grid model.* All wires are constrained to run on an underlying grid of integer lattice points, and no two wires may occupy the same grid point. In the real world, wires have width and minimum spacing between them. We adopt the convention that a grid point corresponds to the lower left portion of a wire.

Figure 3 shows a solution to the problem of Fig. 1 using this model. The terminals $a_1, \cdots, a_n$ and $b_1, \cdots, b_n$ occupy grid points on opposite sides of the channel. As can

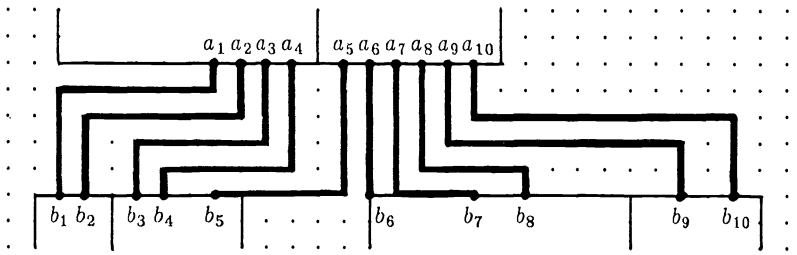


FIG. 3. *A possible solution to the problem in Fig.* 1 *for which the separation is* 5 *and the spread is* 28.

be seen in the figure, we obey the lower-left convention by routing wires on the bottom row of grid points in the channel but not on the top row. This convention also allows terminals to be located at the left corner of a chunk, such as b_6, but not at the right corner.

In order to establish constraints on wirability in this model, consider a straight line segment drawn from (x_1, y_1) to, but not including, (x_2, y_2). We ask the question, "How many wires can cross this line?" With a simple analysis we can show that the answer is $\max(|x_2-x_1|, |y_2-y_1|)$. Without loss of generality, assume the situation is as in Fig. 4, and look at the grid points immediately below the line, that is,

$$\left\{(x, y)\,\middle|\, x_1 \leqq x < x_2 \text{ and } y = \left\lfloor y_1 + \frac{y_2-y_1}{x_2-x_1}(x-x_1) \right\rfloor\right\}.$$

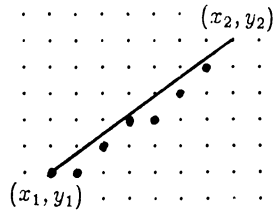


FIG. 4. *The number of wires crossing the half-open line segment is at most the number of grid points immediately below the line.*

Any wire crossing the line must perforce occupy one of these grid points, and therefore the number of such wires is bounded by the cardinality of this set.

Let us now turn to the river routing problem and examine how this constraint can be brought to bear. Let $a_1, \cdots, a_n$ denote both the names of the terminals at the top of the channel and their x-coordinates, and let the same convention hold for the terminals $b_1, \cdots, b_n$ at the bottom of the channel. Figure 5 shows a half-open line

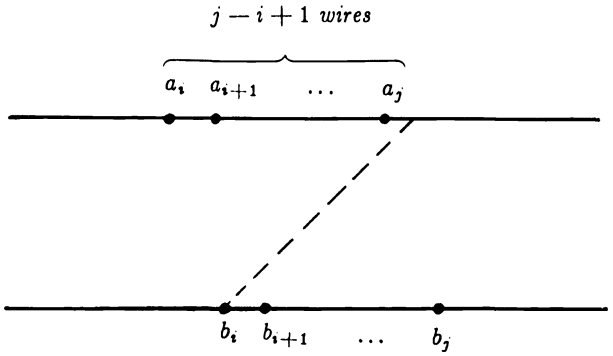


FIG. 5. *The $j-i+1$ wires from $a_i, \cdots, a_j$ must cross the dashed line between b_i and a_j+1.*

segment drawn from terminal b_i to the grid point immediately to the right of terminal a_j. The $j-i+1$ wires emanating from $a_i, \cdots, a_j$ must all cross this line. Similarly, the $j-i+1$ wires emanating from $b_i, \cdots, b_j$ must all cross a line drawn from a_i to b_j+1. In order for a channel with separation t to be routable, therefore, it must be the case that

$$\max(a_j - b_i + 1, t) \geqq j-i+1 \quad \text{and} \quad \max(b_j - a_i + 1, t) \geqq j-i+1 \tag{1}$$

for $1 \leqq i \leqq j \leqq n$.

Although condition (1) is a new condition for wirability, the analysis that leads to it is essentially the same as that in [3] and represents previous work in the field. One of the contributions of this paper is to provide a more compact condition which is equivalent:

$$a_{i+t} - b_i \geqq t \quad \text{and} \quad b_{i+t} - a_i \geqq t \tag{2}$$

for $1 \leqq i \leqq n-t$. The channel is always routable if $t \geqq n$.

Condition (1) implies condition (2) because condition (2) can be obtained by substituting $j = i+t$ in condition (1). For the opposite direction, suppose first that

$j-i+1<t$; then $\max(a_j-b_i+1, t) \geqq t > j-i+1$. If $j-i+1 \geqq t$, on the other hand, then

$$a_j - b_i + 1 = a_{i+t+(j-i-t)} - b_i + 1$$
$$\geqq a_{i+t} - b_i + 1 + (j-i-t) \geqq t + 1 + (j-i-t) = j-i+1$$

since $a_{k+1} \geqq a_k + 1$ for all $1 \leqq k < n$. Thus the two conditions are indeed equivalent.

Figure 6 shows a simple geometric interpretation of condition (2). The condition $a_{i+t} - b_i \geqq t$ means that a line with unit slope going up and to the right from b_i must intersect the top of the channel at or to the left of terminal a_{i+t}. And if the condition

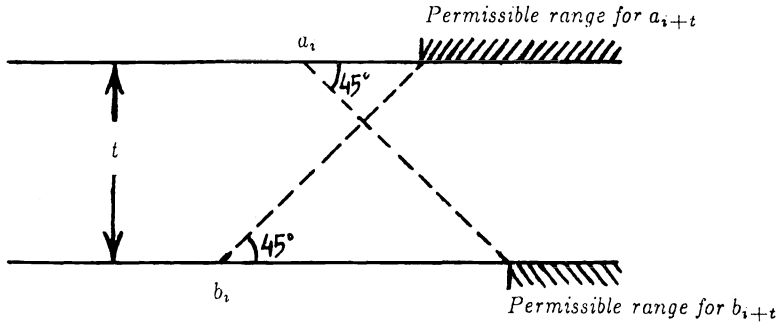


FIG. 6. *Geometric interpretation of* $a_{i+t} \geqq b_i + t$ *and* $b_{i+t} \geqq a_i + t$.

fails, terminal b_j must be to the right of a_j for $i \leqq j < i+t-1$, that is, each wire from an a_j goes down and to the right, which can be shown to follow from the fact that $a_{j+1} \geqq a_j + 1$. (For $b_{i+t} - a_i \geqq t$ the line with slope -1 going down and to the right from a_i must intersect the bottom of the channel at or to the left of terminal b_{i+t}.)

This geometric interpretation can be used to show that condition (2) is not only a necessary condition for routability of the channel, but a sufficient condition as well. In fact, a simple greedy algorithm will successfully route a routable channel. Processing terminals left to right, the greedy algorithm routes each wire across the channel until it hits a previously routed wire; then it follows the contour of the opposite side until it reaches its destination.

To see that this algorithm works given condition (2), we must be more precise about what paths are taken by the wires. Consider without loss of generality a block of consecutive wires that go down and to the right, that is, $a_i \leqq b_i$ for all wires in the block. For any horizontal position x such that $a_i - t < x \leqq b_i$, define

$$\eta_i(x) = \max\left(a_i - x, \max_{b_{i-r} \geqq x} r\right).$$

The path of wire i is then described by the locus of points $(x + \eta_i(x), \eta_i(x))$ for $a_i - t < x \leqq b_i$.

A geometric interpretation of this formulation uses the same intuition as was given in Fig. 6. The line with unit slope drawn from $(x, 0)$ where x is in the range $a_i - t < x \leqq b_i$ must cross wire i. The value $\eta_i(x)$ gives the y-coordinate of wire i where it crosses this line of unit slope. The two-part maximum in the definition of $\eta_i(x)$ corresponds to whether the wire is being routed straight across the channel or whether it is following the contour of the bottom. The value of $\eta_i(x)$ for the latter situation is the number of wires to the left of wire i which must cross the line of unit slope.

We must now show that the locus of points for a wire is a path, that the paths are disjoint, and that they never leave the channel. That the locus of points is indeed a path can be seen by observing that as x ranges from $a_i - t$ to b_i, the initial point is $(a_i, t-1)$, the final point is $(b_i, 0)$ and with a change of one in x the coordinates of the path change by a single grid unit in exactly one of the two dimensions. To show that the paths are disjoint, consider two adjacent wires i and $i+1$ and observe for $a_{i+1} - t < x \leqq b_i$ that $a_i - x < a_{i+1} - x$ and $\max_{b_{i-r} \geqq x} r < \max_{b_{i+1-r} \geqq x} r$ and therefore $\eta_i(x) < \eta_{i+1}(x)$.

To show a path of a wire never leaves the channel, we demonstrate that $\eta_i(x) < t$ for all i and x in the associated range. It is for this part of the proof that we need the assumption that condition (2) holds. If for a wire i, the two-part maximum in the definition of $\eta_i(x)$ is achieved by $a_i - x$, then $\eta_i(x)$ must be less than t because $x > a_i - t$. Suppose then, that the two-part maximum is achieved by the maximal r such that $b_{i-r} \geqq x$. To show that $r < t$, we assume the contrary and obtain a contradiction. But since $b_{i-t} \geqq b_{i-r} \geqq x > a_i - t$, the contradiction is immediate because $a_i - b_{i-t} \geqq t$ from condition (2).

3. The structure of the placement problem. The objective of a placement algorithm is to set up a routing problem that is solvable and minimizes some cost function. Many criteria can be adopted to measure the cost of a placement for river routing, whether in terms of area (total or channel) or some other function of spread and separation. A plot of minimal spread versus given separation reveals that the region of feasible placements may not be convex although the curve is guaranteed to be monotonically decreasing. (Fig. 7 shows the plot for the problem of Fig. 1.) Any measure of placement cost that is a function of spread and separation and which is monotonically increasing in each of spread and separation will therefore find a minimum on this curve.

Thus we content ourselves with producing points on this curve, that is, *determining a placement which achieves the minimum spread for a given separation t*, if indeed the channel is routable in t tracks. If minimum separation is the goal, for example, binary search can determine the optimum t in $O(\lg t)$ steps. Since the algorithm presented in the next section determines a placement for fixed t in $O(n)$ time, where n is the

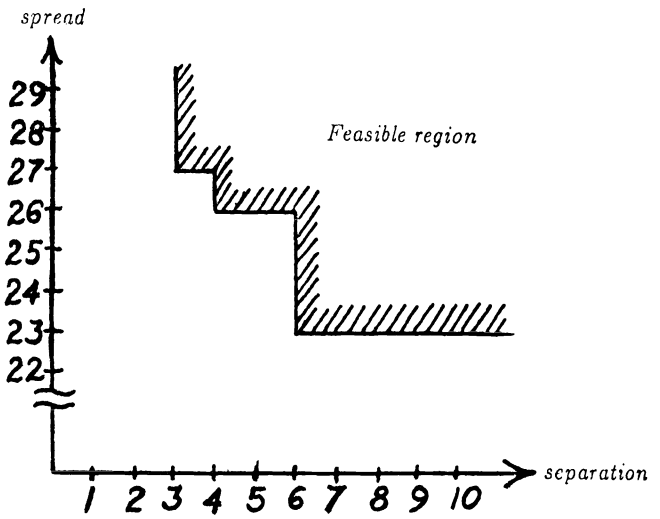
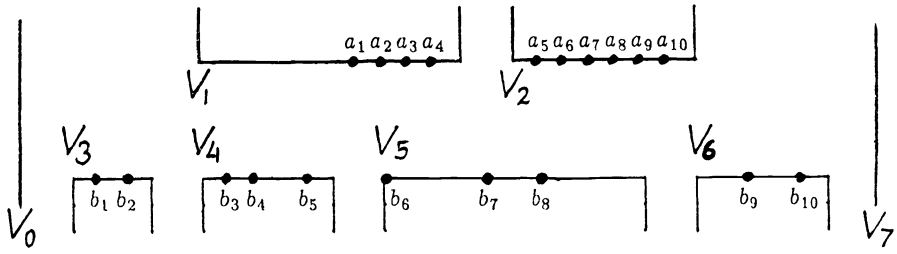


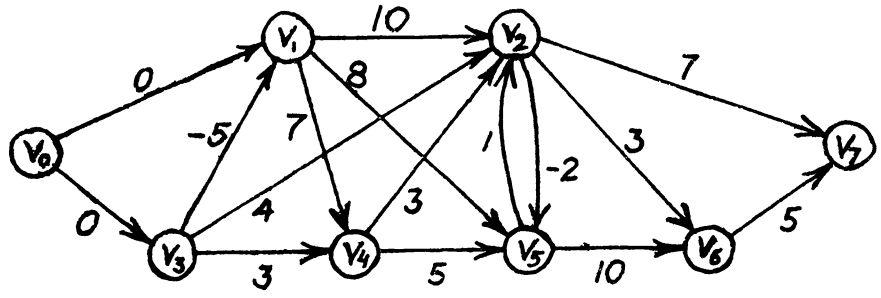
FIG. 7. *The curve of minimum spread versus separation for the example of Fig.* 1.

number of terminals, and since the separation need never be more than n, a minimum-separation placement can be achieved in $O(n \lg n)$ time. For more general objective functions such as area, the optimum value can be determined in $O(n^2)$ time.

We now examine the character of the placement problem for river routing when the separation t is given. The n terminals are located on m chunks which are partitioned into two sets that form the top and bottom of the channel. For convenience, we shall number the chunks from 1 to k on the top and $k+1$ to m on the bottom. The order of chunks on each side of channel is fixed, but they may be moved sideways so long as they do not overlap. For each chunk i, a variable v_i represents the horizontal position of its left edge. Any placement can therefore be specified by an assignment of values to these variables. We also add two variables v_0 and v_{m+1} to the set of variables, which represent the left and right boundaries of the channel. The spread is thus $v_{m+1} - v_0$. Figure 8(a) shows the eight variables for the example from Fig. 1.



(a) Assignment of variables to chunks and channel boundaries.



(b) The placement graph for separation 3.

FIG. 8. *Representing the placement constraints as a graph for the example of Fig.* 1.

Since the relative positions of terminals within a chunk are fixed, the wirability constraints of condition (2) can be reexpressed in terms of the chunks themselves to give placement constraints that any assignment of values to the v_i must satisfy. If terminal a_{i+t} lies on chunk h and terminal b_i lies on chunk j, the constraint $a_{i+t} - b_i \geqq t$ can be rewritten as $v_h - v_j \geqq r_{hj}$, where r_{hj} reflects t and the offsets of the terminals from the left edge of their respective chunks. The constraint between two chunks determined in this way will be the maximal constraint induced by pairs of terminals.

Additional constraints arise from the relative positions of chunks on either side of the channel. For each pair of adjacent chunks i and $i+1$, the constraint $v_{i+1} - v_i \geqq w_i$

must be added to the set of placement constraints, where w_i is the width of chunk i. Four more constraints are needed which involve the boundary variables v_0 and v_{m+1}. For chunks 1 and $k+1$ which are leftmost on the top and bottom, the constraints $v_1 - v_0 \geqq 0$ and $v_{k+1} - v_0 \geqq 0$ enforce that these chunks lie to the right of the left boundary of the channel. For chunks k and m which are rightmost on the top and bottom, the relations $v_{m+1} - v_k \geqq w_k$ and $v_{m+1} - v_m \geqq w_m$ constrain them to lie to the left of the right boundary.

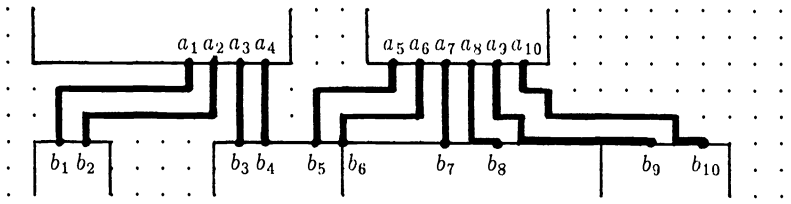
Figure 8(b) shows a *placement graph* which represents the constraints between chunks for the placement problem of Fig. 1 where the separation is 3 tracks. A directed edge with weight δ_{kl} goes from v_k to v_l if there is a constraint of the form $v_l - v_k \geqq \delta_{kl}$. For example, the weight of 1 on the *cross edge* going from v_5 to v_2 is the maximal constraint of $a_9 - b_6 \geqq 3$ and $a_{10} - b_7 \geqq 3$ which yield $v_2 - v_5 \geqq -2$ and $v_2 - v_5 \geqq 1$ since $a_9 = v_2 + 5$, $a_{10} = v_2 + 6$, $b_6 = v_5$ and $b_7 = v_5 + 4$. The *side edge* from v_4 to v_5 arises from the constraint that chunk 4, which is 5 units long, must not overlap chunk 5.

The goal of the placement problem is to find an assignment of values to the v_i which minimizes the spread $v_{m+1} - v_0$ subject to the set of constraints. This formulation is an instance of linear programming where both the constraints and the objective function involve only differences of variables. Not surprisingly, this problem can be solved more efficiently than by using general linear programming techniques. In fact, it reduces to a *single-source-longest-paths* problem in the placement graph. The length of a longest path from v_0 to v_{m+1} corresponds to the smallest spread of the channel that complies with all the constraints. The placement of each chunk i relative to the left end of the channel is the longest path from v_0 to v_i. If the placement graph has a cycle of positive weight, then no placement is possible for the given separation.

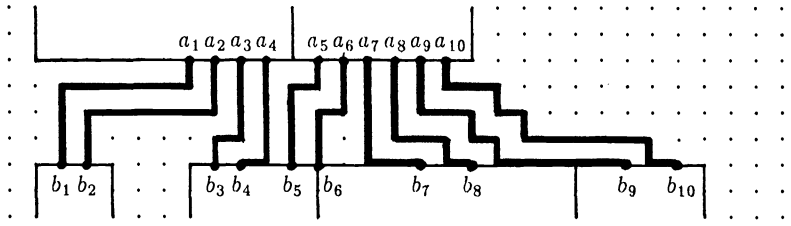
For the placement problem of Fig. 1 with a three-track separation, the longest path from v_0 to v_2 in the placement graph (Fig. 8) is $v_0 - v_1 - v_4 - v_5 - v_2$ with weight 13 which corresponds to the positioning of chunk 2 in the optimal placement shown in Fig. 9(a). Figures 9(b) through 9(d) show optimal solutions to the placement problem of Fig. 1 for separations $t = 4$ through $t = 6$. The constraints for $t = 2$ yield a cycle of positive weight in the placement graph and thus no placement is possible which achieves a separation of only two tracks. On the other hand, no more than 6 tracks are ever needed. The spread achieved for $t = 6$ is the sum of the widths of the chunks on one side of the channel, which is a lower bound for the spread.

4. A linear-time algorithm for the fixed-separation placement problem. The analysis of § 3 showed that the optimal placement problem for fixed-separation river routing was reducible to the single-source-longest-paths problem on a placement graph. For a general graph $G = (V, E)$ this problem can be solved in time $O(|V| \cdot |E|)$ by a *Bellman–Ford algorithm* [6]. Better performance is possible, however, due to the special structure of placement graphs. This section reviews the Bellman–Ford algorithm and shows how it can be adapted to give an $O(m)$-time algorithm for the longest-paths problem on a placement graph, where m is the number of chunks. Since the placement constraints can be generated in $O(n)$ time, where n is the number of terminal pairs, this algorithm leads to an optimal linear-time algorithm for the fixed-separation placement problem. The discovery of a linear-time algorithm represents joint research with James B. Saxe of Carnegie–Mellon University.

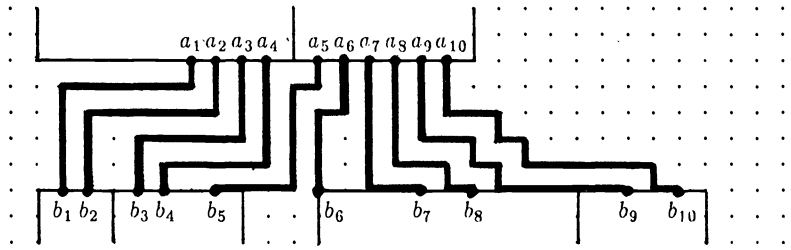
The linear-time algorithm is a refinement of the standard Bellman–Ford algorithm which for each vertex v_i, where $i = 1, \cdots, m+1$, iteratively updates the length $\lambda(v_i)$ of a tentative longest path from v_0 to v_i. The algorithm initializes $\lambda(v_0)$ to zero and all other $\lambda(v_i)$ to $-\infty$; then it sequences through a list $\mathscr{E}$ of edges, and for each edge



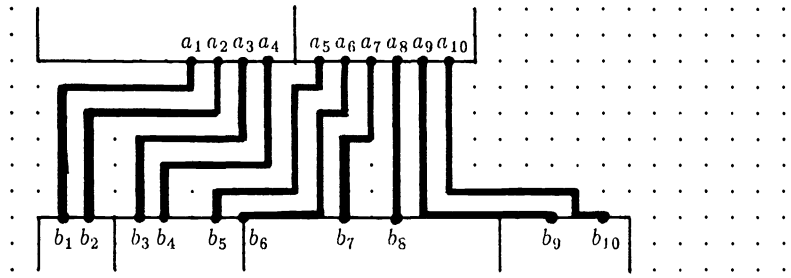
(a) Separation 3, spread 27.



(b) Separation 4, spread 26.



(c) Separation 5, spread 26.



(d) Separation 6, spread 23.

FIG. 9. *Optimal placements and routings for the problem of Fig.* 1 *with separations ranging from* $t = 3$ *to* $t = 6$.

(v_i, v_j) with weight δ_{ij} updates $\lambda(v_j)$ by

$$\lambda(v_j) \leftarrow \max(\lambda(v_j), \delta_{ij} + \lambda(v_i)).$$

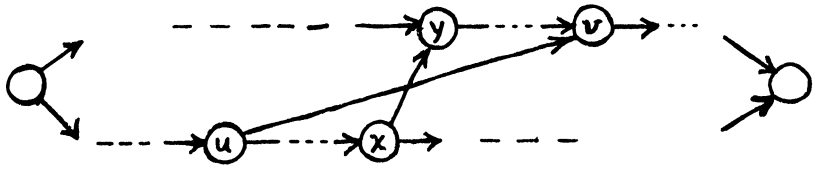
The list $\mathscr{E}$ of edges is the key to the correctness of the algorithm. *The length of a longest path from the source* v_0 *to a vertex* v_j *converges to the correct value if the edges of the path form a subsequence of the list* $\mathscr{E}$. (This can be proved by adapting the analysis of [12].) In the normal algorithm for a general graph $G = (V, E)$, the list $\mathscr{E}$

is $|V|-1$ repetitions of an arbitrary ordering of the edges in E, which ensures that every vertex-disjoint path in G beginning with v_0 is a subsequence of $\mathscr{E}$. If there are no cycles of positive weight in the graph G, then from v_0 to each other vertex in G, there is a longest path that is vertex-disjoint; hence the algorithm is guaranteed to succeed. The condition of positive-weight cycles can be tested at the end of the algorithm either by checking whether all constraints are satisfied or by simply running the algorithm through the edges in E one additional time and testing whether the values of any $\lambda(v_i)$ change.

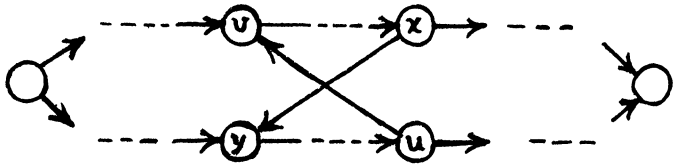
The list $\mathscr{E}$ is also the key to the performance of a Bellman–Ford algorithm. For the general algorithm on an arbitrary graph $G=(V,E)$, the length of the list is $(|V|-1)\cdot|E|$, and thus the algorithm runs in $O(|V|\cdot|E|)$ time. For a placement graph it is not difficult to show that both $|V|$ and $|E|$ are $O(m)$, and thus the longest-paths problem can be solved in $O(m^2)$ time by general algorithm. But a linear-time algorithm can be found by exploiting the special structure of a placement graph to construct a list $\mathscr{E}$ of length $O(m)$ that guarantees the correctness of the Bellman–Ford algorithm. We now look at the structure of placement graphs more closely.

The vertices of a placement graph $G=(V,E)$ corresponding to the chunks on the top of the channel have a natural linear order imposed by the left-to-right order of the chunks. We define the partial order $\prec$ as the union of this linear order with the similar linear order of bottom vertices. Thus $u \prec v$ for vertices u and v if their chunks lie on the same side of the channel and the chunk that corresponds to u lies to the left of the one which corresponds to v. The left-boundary vertex v_0 precedes all other vertices, and all vertices precede the right-boundary vertex v_{m+1}. The partial order $\preceq$ is the natural extension to $\prec$ that includes equality.

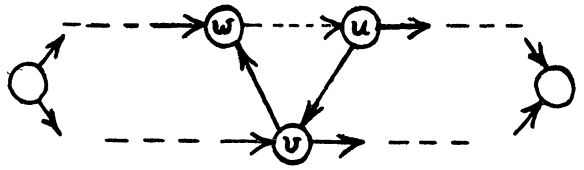
The next lemma describes some of the structural properties of placement graphs. Fig. 10 illustrates the impossible situations described in properties (i) and (ii) and



(a) The situation forbidden by property (i).



(b) The situation forbidden by property (ii).



(c) Every simple cycle contains at most one vertex from the top or at most one vertex from the bottom. The edges incident on the vertex are a consequence of property (iii).

FIG. 10. *The properties of the placement graph enumerated in Lemma* 1.

shows the only kind of simple cycle that can occur in a placement graph together with the two consecutive cross edges that satisfy property (iii).

LEMMA 1. *Any placement graph* $G = (V, E)$ *has the following properties*:

(i) *There do not exist cross edges* (u, v) *and* (x, y) *such that* $u < x$ *and* $y < v$.

(ii) *There do not exist cross edges* (u, v) *and* (x, y) *such that* $v < x$ *and* $y < u$.

(iii) *All cycles have two consecutive cross edges* (u, v) *and* (v, w) *such that* $w \leq u$.

Proof. Properties (i) and (ii) can be proved by considering which of the terminal constraints from condition (2) induce the edges in the placement graph. For each of these cases, suppose the edge (u, v) was caused by the terminals i in u and $i+t$ in v, and the edge (x, y) came from the terminals j in x and $j+t$ in y. For property (i) we have $u < x$ and $y < v$ and thus $i < j$ and $j+t < i+t$. Canceling t from this latter inequality obtains the contradiction. The assumption to be proved impossible in (ii) is that $v < x$ and $y < u$, which implies $i+t < j$ and $j+t < i$. Since t is nonnegative, we gain a contradiction.

To prove property (iii), we need only consider simple (vertex-disjoint) cycles. Since no cycle can consist solely of side edges, every simple cycle must have a cross edge (u, v) going from bottom to top. In order to complete the cycle, there must be a top-to-bottom edge (w, x) such that $v \leq w$ and $x \leq u$. If $v = w$ or $x = u$, then the pair of edges satisfies property (iii). But if $v \neq w$ and $x \neq u$, then the pair of edges violates property (ii). □

Each edge in the placement graph is either a top edge, a top-bottom edge, a bottom-top edge or a bottom edge. For each of these four sets of edges, there is a natural linear order of edges based on $\leq$, where (u, v) precedes (x, y) for two edges in the same set if $u \leq x$ and $v \leq y$. Property (i) guarantees that the linear order holds for two cross edges in the same set. Let TT, TB, BT and BB be the four lists of edges according to the natural linear order and include the two edges out of v_0 and the two edges into v_{m+1} in either TT or BB as appropriate.

The list $\mathscr{E}$ used by the Bellman–Ford algorithm is constructed by a merge of the four lists which we call MERGE. At each step of MERGE, a tournament is played among the first elements of each list. If (u, v) and (v, w) are the first elements of two lists, then (u, v) beats (v, w) if $w \not\leq u$. Since there may be more than one edge beaten by none of the other three, ties are broken arbitrarily. The winner is appended to $\mathscr{E}$ and removed from the head of its list. The tournament is then repeated until no edges remain in any of the four lists. The performance of the tournament can be improved by recognizing that only six of the twelve possible comparisons of edges need be tried and that $w \not\leq u$ is guaranteed for all but two. Figure 11 shows a possible ordering of edges in $\mathscr{E}$ for the placement graph in Fig. 8.

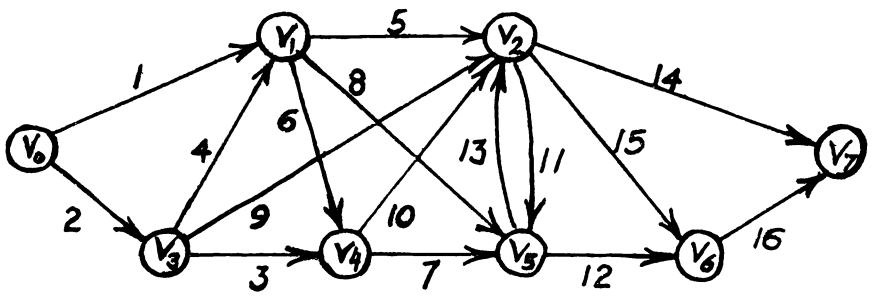


FIG. 11. *A possible ordering of edges in* $\mathscr{E}$ *for the placement graph in Fig.* 8.

In order for MERGE to be well defined, the tournament must always produce a winner, which is a consequence of the next lemma.

LEMMA 2. *The list $\mathscr{E}$ produced by* MERGE *is a topological sort of the edges of E according to the relation R, where $(u, v)R(v, w)$ if $w \not\preceq u$.*

Proof. First, we show that the relation R is acyclic so that the edges can indeed be topologically sorted. By definition of R, a cycle in R induces a cycle in the placement graph. According to property (iii), the cycle must have two consecutive cross edges (u, v) and (v, w) such that $w \preceq u$. But since $(u, v)R(v, w)$, we also have that $w \not\preceq u$, which is a contradiction.

The proof that MERGE topologically sorts the edges of E according to R makes use of the fact that if a vertex v is the tail of an arbitrary edge in any one of the four lists TT, TB, BT or BB then for every $u \preceq v$ there is an edge in the same list emanating from u. Suppose that MERGE does not topologically sort the edges of E according to R. Then there is a first edge (u, v) in $\mathscr{E}$ such that there exists an edge (v, w) earlier in $\mathscr{E}$ and $(u, v)R(v, w)$. Consider the edge (x, y) in the same list as (u, v) that competed with (v, w) when (v, w) was the winner of the tournament. For each of the possible combinations of lists for (u, v) and (v, w), it can always be deduced that there is an edge emanating from y which makes (x, y) an earlier violator of the topological sort than (u, v). □

Since each edge of E is included exactly once in the list $\mathscr{E}$ created by MERGE, the Bellman–Ford algorithm applied to $\mathscr{E}$ has a running time linear in the number of chunks. The correct values for longest paths are produced by the algorithm if for every vertex v, there is a subsequence of $\mathscr{E}$ that realizes a longest path from v_0 to v. Since for every longest path, there is a vertex-disjoint longest path, the following theorem proves the correctness of this linear-time Bellman–Ford algorithm.

THEOREM 3. *Let G be a placement graph with left-boundary vertex v_0. Then every vertex-disjoint path beginning with v_0 is a subsequence of the list $\mathscr{E}$ created by the procedure* MERGE.

Proof. We need only show that every pair of consecutive edges in a vertex-disjoint path from v_0 satisfies R because then Lemma 2 guarantees that the path is a subsequence of $\mathscr{E}$. Suppose (u, v) and (v, w) are two consecutive edges on a vertex-disjoint path from v_0 which violate R, that is, $w \preceq u$. If either (u, v) or (v, w) is a side edge, the pair must satisfy R, and thus both must be cross edges with the vertices u and w on the same side. Since if $w = u$, the path is not vertex-disjoint, we need only show that $w \prec u$ is impossible.

Assume, therefore, that $w \prec u$, and consider the initial portion of the path from v_0 to u. since $v_0 \prec v$ and $v_0 \prec w$, there must be an edge (x, y) on the path which goes from the set of vertices to the left of (v, w) to the right of (v, w) in order to get to u. But then either property (i) or property (ii) is violated depending on whether $x \prec v$ and $w \prec y$, or $y \prec v$ and $x \prec w$. □

5. Other wiring models. The reduction from the fixed-separation placement problem in the square-grid model to the single-source-longest-paths problem is possible because the wirability constraints can all be written in the form $v_i - v_j \geqq \delta_{ij}$. Thus for any wiring model where wiring constraints can be written in this form, the reduction will succeed. Also, it should be observed that in general, the performance of the single-source-longest-path algorithm will not be linear, but will be a function of the number of constraints times the number of variables. This section reviews other models and gives the necessary and sufficient wirability constraints for each. Some of these models are discussed in [1], [3], [10] and [11].

1. *One-layer, gridless rectilinear.* Wires in this model must run horizontally or vertically, and although they need not run on grid points, no two wires can come within one unit of each other. The wirability constraints for this model are the same as for the square grid model:

$$a_{i+t} - b_i \geqq t \quad \text{and} \quad b_{i+t} - a_i \geqq t$$

for $1 \leqq i \leqq n - t$. As with the square-grid model, the fixed-separation placement algorithm for this model can be made to run in linear time.

2. *One-layer, gridless, rectilinear and forty-five degree.* This model is the same as the gridless rectilinear, but in addition wires can run which have slope ± 1. The constraints in this case are

$$a_{i+r} - b_i \geqq r\sqrt{2} - t \quad \text{and} \quad b_{i+r} - a_i \geqq r\sqrt{2} - t$$

for $t/\sqrt{2} \leqq r \leqq t$ and $1 \leqq i \leqq n - r$. The placement algorithm for this model runs in $O(\min(tm^2 + tn, m^3 + tn, m^3 + n^2))$ time.

3. *One-layer, gridless.* Wires can travel any direction. The constraints are

$$a_{i+r} - b_i \geqq \sqrt{r^2 - t^2} \quad \text{and} \quad b_{i+r} - a_i \geqq \sqrt{r^2 - t^2}$$

for $t \leqq r \leqq n$ and $1 \leqq i \leqq n - r$. The placement algorithm runs in $O(m^3 + n^2)$ time.

4. *Multilayer models.* All the models presented until now have been one-layer models. It is natural to generalize to l-layer models in which wires may travel on different layers. Remarkably, optimal routability can always be achieved with no contact cuts [1], that is, a wire need never switch layers. The necessary and sufficient conditions for these multilayer models are a natural extension of the one-layer conditions. For example, in the one-layer, gridless, rectilinear model the conditions are modified for l layers to be

$$a_{i+lt} - b_i \geqq t \quad \text{and} \quad b_{i+lt} - a_i \geqq t$$

for $1 \leqq i \leqq n - lt$.

There are some wiring models, however, where upper and lower bounds for wirability do not meet. For these models a constraint graph which represents upper bounds will give the best possible placement for those bounds. A graph representing lower bounds will give lower bounds on the best possible placement. Together, bounds can be established for some of these models, and heuristic algorithms invoked to attempt routing within the feasible range of optimality.

6. Extensions and conclusions. A variety of related placement problems can be solved by the method described in this paper. Some entail extensions to the problem specifications, others employ different wiring models. In this section we shall mention a few extensions we can handle and suggest further research on more complicated problems. These and other extensions are explored in [8].

Nonriver routing. The placement algorithm gives optimal placements for river routing, but there are other routing configurations for which it works optimally as well. One example is the two-layer, any-to-any routing problem where two sets of terminals must be connected across a channel, but they may be connected in any order.

Range-terminals. In some routing situations terminals occupy not a single point, but rather a contiguous region along the edge of the channel. For example, the terminal might be a wire that runs along the edge of the chunk, and connection can be made to the wire anywhere. The additional flexibility of viewing a terminal as a contiguous

range of points can be exploited by both the greedy routing algorithm and the placement algorithm in any of the river-routing models we have discussed.

Each range-terminal is specified by an interval $[a_i^L, a_i^R]$ or $[b_i^L, b_i^R]$. The greedy routing algorithm operates as before with minor changes. If the range-terminals overlap, the wire is routed straight across. Otherwise, assume without loss of generality that $a_i^R < b_i^L$ and use the standard greedy algorithm to route a wire from a_i^R to b_i^L.

The wirability conditions for placement are accordingly adjusted. In the rectilinear case, for example, the condition $a_{i+t} - b_i \geqq t$ is rewritten as $a_{i+t}^R - b_i^L \geqq t$ and condition $b_{i+t} - a_i \geqq t$ becomes $b_{i+t}^R - a_i^L \geqq t$. The transformation to chunk variables is as before and the placement algorithm is unchanged.

Variable-width wires. In some applications the wires that must be routed do not all have the same width. Our scheme can be generalized to deal with this situation as long as each wire has uniform width. By computing an experimental cumulative distribution function of the wire widths both routability and placement for a fixed-separation problem can be determined in linear time [8].

Minimum jogging. The number of jogs (or turns) of wires produced by the greedy algorithm may be excessive. In some cases, $\Omega(n^2)$ jogs will be produced when $O(n)$ jogs suffice to route the same channel. An adaptation of the routability constraints developed in § 2 can be used to absolutely minimize the total number of jogs in the channel [7].

River-routing in a polygon. Instead of constraining terminals to lie on two parallel lines, we allow them to reside anywhere along the boundary of a simple polygon. The planarity of the interconnect as well as the wirability within the polygon's area can be tested in time $O(n+p)$, where p is the number of corners in the polygon. A routing can be produced in time $O(n^2+pn)$ using an extension of the greedy algorithm. These results are reported in [8].

Parallel channels. Multiple, parallel horizontal channels are easily handled within the same graph-theoretic framework as long as the width of each channel is given. Every row of chunks is represented by a chain of vertices from a common left boundary to a common right boundary. The wiring conditions in the channels are represented by edges linking adjacent chains. The optimal placement is achieved by solving the longest paths problem on this graph. The standard Bellman–Ford algorithm runs in time $O(n+m^2)$ since the number of edges in the graph is $O(m)$. We do not know whether improvements of the kind used in § 4 for the single channel problem can be obtained for the multiple channels case.

The problem is much harder, however, when the input is specified so that only the *sum* of the channel separations is given. Then the problem is to allocate the channel widths so as to minimize the spread subject to the additional constraint on the channels' total width. This problem is NP-complete as has been shown by Pinter and Sipser [8].

Two-dimensional river routing. The two-dimensional river-routing problem is illustrated in Fig. 12. In the figure, a line between two chunks indicates that wires must be river-routed between them. Unfortunately, in order to optimally solve this general problem, it appears that the constraints indicated by the lines must be convex in both dimensions, not just in one as is the case for the wiring models considered here. When the constraints are convex, however, convex programming can be used to optimize a cost function such as the area of the bounding box of the layout. One model which gives convex constraints for the general two-dimensional problem is the one in which all wires must be routed as straight line segments between terminals such that no minimum spacing rules are violated. This model is not particularly

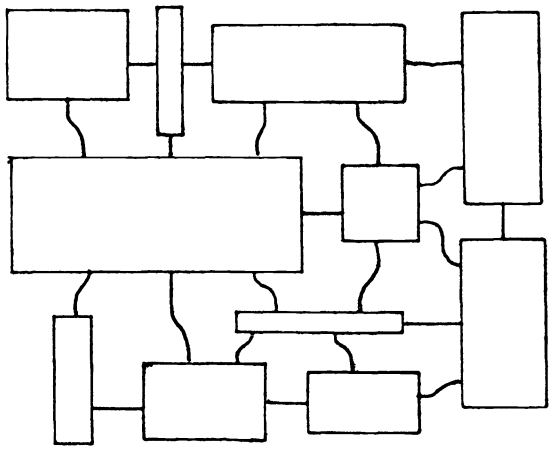


FIG. 12. *A two-dimensional extension to the river-routing problem. A solid line between two modules indicates routing occurs between them.*

interesting from a practical standpoint, however. Heuristics for solving the related two-dimensional *compaction* problem by repeatedly compacting in one dimension and then the other can be found in [4].

A major deficiency of many placement programs is that they lack knowledge about the wirability of the routing problems that they set up. We have shown for river routing that wirability conditions can be translated directly into placement constraints without the overhead of actually wiring the channel. For rectilinear river routing, the running time of the greedy wiring algorithm is $O(n^2)$, and any cost function for placement that is monotonic in spread and separation can be optimized in $O(n^2)$ time without the overhead of routing. Studying wirability in the general case may lead to the development of heuristics for wirability that do not involve routing. A program that uses this heuristic knowledge should be able to outperform the iterative place-route, place-route programs that dominate today.

Acknowledgments. We would like to thank Howie Shrobe of the MIT Artificial Intelligence Laboratory for posting the plots of the data paths from the Scheme81 chip which inspired out interest in this placement problem and for his valuable comments on the practicality of our work. We would also like to thank Alan Baratz and Ron Rivest from the MIT Laboratory for Computer Science for numerous helpful discussions, Shlomit Pinter (also from the Laboratory for Computer Science) for influencing the direction of our proof of Theorem 3 and Eli Messinger from the University of Washington for pointing out an error in condition (1) as it appeared in an earlier version of this paper. Finally, special thanks to Jim Saxe of Carnegie–Mellon University for his key contributions to the linear-time algorithm for longest paths.

REFERENCES

[1] A. E. BARATZ, *Algorithms for integrated circuit signal routing*, Ph.D. dissertation, Dept. Electrical Engineering/Computer Science, Massachusetts Institute of Technology, Cambridge, August 1981.

[2] I. BATALI, N. MAYLE, H. SHROBE, G. SUSSMAN AND D. WEISE, *The* DPL/*Daedalus design environment*, Proc. International Conference on VLSI, Univ. of Edinburgh, August 1981, pp. 183–192.

[3] D. DOLEV, K. KARPLUS, A. SIEGEL, A. STRONG AND J. D. ULLMAN, *Optimal wiring between rectangles*, Proc. Thirteenth Annual ACM Symposium on Theory of Computing, May 1981, pp. 312–317.

[4] M.-Y. HSUEH, *Symbolic layout and compaction of integrated circuits*, Memo UCB/ERL-M79/80, Ph.D. dissertation, Electronics Research Laboratory, Univ. California, Berkeley, December 1979.

[5] D. L. JOHANNSEN, *Silicon compilation*, Technical Report 4530, Ph.D. dissertation, Dept. Computer Science, California Institute of Technology, Pasadena, CA., 1981. An overview appears as *Bristle blocks: a silicon compiler*, in the Proc. of the Sixteenth Design Automation Conference, June 1979, pp. 310–313.

[6] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[7] R. Y. PINTER, *On routing two-point nets across a channel*, Proc. Nineteenth Design Automation Conference, June 1982, pp. 894–902.

[8] R. Y. PINTER, *The impact of layer assignment methods on layout algorithms for integrated circuits*, Ph.D. dissertation, Dept. Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, August 1982.

[9] H. E. SHROBE, *The data path generator*, Proc. Conference on Advanced Research in VLSI, Massachusetts Institute of Technology, Cambridge, January 1982, pp. 175–181.

[10] A. SIEGEL AND D. DOLEV, *The separation for general single-layer wiring barriers*, Proc. CMU Conference on VSLI Systems and Computations, October 1981, pp. 143–152.

[11] M. TOMPA, *An optimal solution to a wire-routing problem*, Proc. Twelfth Annual Symposium on Theory of Computing, April–May 1980, pp. 161–176.

[12] J. Y. YEN, *An algorithm for finding shortest routes from all source nodes to a given destination in general networks*, Quart. Appl. Math. 27 (1970), pp. 526–530.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0004 $01.25/0

OPTIMAL DYNAMIC EMBEDDING OF TREES INTO ARRAYS*

MICHAEL C. LOUI†

Abstract. An optimal method for dynamically embedding trees into arrays is presented. Every multihead tree machine of time complexity $t(n)$ can be simulated on-line by a multihead d-dimensional machine in time $O(t(n)^{1+1/d}/\log t(n))$. An information-theoretic argument gives the worst-case lower bound $\Omega(t(n)^{1+1/d}/\log t(n))$ on the time required.

Key words. tree, array, data structure, multidimensional Turing machine, simulation, embedding

1. Introduction. Several researchers [2], [6], [14], [15], [16] have investigated static embeddings among data structures. To model the physical implementation of a logical data structure, they fix a correspondence between locations in the guest (logical) structure and locations in the host (physical) structure and analyze the effect of various correspondences on access costs. Lynch's work on accessibility of values in algebras [9] gives lower bounds on these access costs. When few cells of the guest structure are used during a computation, these fixed embeddings waste space in the host structure.

In contrast, we have developed dynamic embeddings that designate representatives in the host only for cells of the guest that are actually used [7], [8]. To model storage and retrieval operations in data structures, we have used generalizations of Turing machines: machines with tree-structured worktapes model trees, and machines with multidimensional worktapes model arrays. The access heads of the machines correspond to access pointers into the data structures.

Continuing the study of dynamic embeddings, we present an optimal on-line simulation of a multihead tree machine of time complexity $t(n)$ by a multihead d-dimensional machine in time $O(t(n)^{1+1/d}/\log t(n))$. To establish the lower bound $\Omega(t(n)^{1+1/d}/\log t(n))$ on the time required, we employ the information-theoretic technique developed by Paul, Seiferas and Simon [11]. This technique extends the original counting arguments of Hennie [4].

Previously, Pippenger and Fischer [12] proved that every tree machine of time complexity $t(n)$ can be simulated on-line by a one-dimensional machine in time $O(t(n)^2/\log t(n))$, an optimal amount of time. For $d \geqq 2$, a theorem of Grigoriev [3] implies a simulation by a d-dimensional machine in time $O(t(n)^{1+1/(d-1)})$.

Conversely, Reischuk [13] devised on-line simulations of d-dimensional machines of time complexity $t(n)$ by tree machines in time $O(t(n)(5^d)^{\log^* t(n)})$. It is not known whether Reischuk's simulation is optimal.

2. Definitions. Cook and Aanderaa [1] introduced the *bounded activity machine*, a generalization of the Turing machine model. A bounded activity machine has a finite-state control, a read-only linear input tape (from which it reads input symbols), a write-only linear output tape (on which it writes output symbols) and a finite number of *storage media*, each of which has a finite number of access heads. A storage medium is an infinite set of *cells*, each of which can hold a symbol from a finite *storage alphabet*. One cell is designated the *origin* of the storage medium. Each access head is located at a cell of its storage medium. From every cell, a head can *shift* to one of a finite

* Received by the editors October 12, 1981, and in revised form August 20, 1982. This research was supported by the Joint Services Electronics Program (U.S. Army, U.S. Navy, U.S. Air Force) under contract N00014-79-C-0424.

† Coordinated Science Laboratory, University of Illinois, Urbana, Illinois 61801.

number of other cells. Cell X is *at distance* s from cell Y if s is the minimum number of shifts required for an access head on X to travel to Y.

A bounded activity machine operates in a sequence of *steps*. At each step the machine reads the symbols in the cells on which the input head and access heads are located. Depending on these symbols and its current control state, the machine

(i) writes symbols on the cells on which the access heads are located,
(ii) may write a symbol on its output tape,
(iii) changes its control state, and
(iv) shifts its heads.

We may assume that the machine can detect when two heads are located at the same cell and hence that the two heads do not attempt to write different symbols in this cell. Initially, all storage cells hold a particular *blank* symbol, and every access head is positioned on the origin of its storage medium.

The *time complexity* of the machine is a function $t(n)$ defined for each n to be the maximum over all input strings w of length n of the time (number of steps) that the machine spends on w.

A *tree worktape* is a storage medium whose cells are organized into a complete infinite rooted binary tree. The root has two children, and all other cells have two children and a parent. An access head can shift from a cell to its parent or to one of its children. The root of the tree worktape is its origin. A *tree machine* is a bounded activity machine whose storage media are all tree worktapes. Let W be a tree worktape. We fix a natural bijection between strings in $\{0, 1\}^*$, called *locations*, and cells of W; we write $W(b)$ for the cell at location b. Let λ denote the empty string. The root of W is $W(\lambda)$. In general, $W(b0)$ is the left child of $W(b)$, and $W(b1)$ is its right child. Let $W[b, r]$ denote the complete subtree of W of height r rooted at $W(b)$. The leaves of $W[b, r]$ are at distance r from $W(b)$.

A *d-dimensional worktape* is a storage medium whose cells correspond bijectively with d-tuples of integers, called the *coordinates*. An access head can shift from a cell to another whose coordinates differ in just one component by ± 1. The origin of the worktape is the cell whose coordinates are all zero. A *box* is a set of cells that form a d-dimensional cube. The *volume* of a box is the number of cells in it. The *base cell* of a box is the cell whose coordinates are the smallest. A *d-dimensional machine* is a bounded activity machine whose storage media are all d-dimensional worktapes.

3. Simulation. Fix integers $d \geqq 2$ and $h \geqq 1$. Let T be a tree machine of time complexity $t(n)$ with just one worktape W with h access heads. Let Δ be the storage alphabet of T. We may assume that whenever a head of T visits a storage cell, it writes a nonblank symbol in the cell. We shall devise an on-line simulation of T by a d-dimensional machine S in time $O(t(n)^{1+1/d}/\log t(n))$: There is a fixed constant k such that for every computation of T, if $s_1 \leqq s_2 \leqq \cdots$ are the steps at which T shifts its input head or output head, then there are corresponding steps $s_1' \leqq s_2' \leqq \cdots$ at which the input and output heads of S perform the same shifts, reading and writing the same symbols, and $s_i' \leqq k s_i^{1+1/d}/\log s_i$ for every i. Since every tree machine with a total of h access heads on several tree worktapes can be simulated in real time by a tree machine with h heads on one worktape with a larger storage alphabet, it will follow that every multitape tree machine can be simulated on-line by a d-dimensional machine in time $O(t(n)^{1+1/d}/\log t(n))$.

Let us outline the simulation before giving the details. The tree worktape of T is divided into subtrees called *blocks*. To simulate the contents of a block B, machine S uses a box P called a *page*. The page compactly represents the contents of the

corresponding block with a *path string* [12]. The length of the path string is proportional to the number of nonblank cells in B. Page P has an *Ancestor-pointer* that specifies the coordinates of the page that corresponds to the immediate ancestor of B. For every immediate descendant B' of B that has nonblank cells, P has a *Descendant-pointer* that specifies the coordinates of the page that corresponds to B'. To simulate one step of T, machine S determines the symbols read by the access heads of T by examining the path strings in the pages that correspond to the blocks in which the heads are located. Whenever a head exits a block B and enters another block B', machine S uses the pointers in the page that corresponds to B to find the page that corresponds to B'.

As the number of nonblank cells in B increases, the volume of the page that corresponds to B must increase. Thus, S allocates a sequence of pages $P_1, P_2, \cdots$ of increasing volume to represent the contents of B. To avoid changing all the Ancestor-pointers and Descendant-pointers, S uses an indirect addressing scheme to locate the current page that corresponds to B. Each P_i has a *First-pointer* and a *Current-pointer*. The First-pointer of every P_i specifies the coordinates of P_1, the first page that was allocated for B. The Current-pointer of P_1 specifies the coordinates of the current page that corresponds to B. Whenever a new larger page P_{i+1} is allocated for B, the Current-pointer of P_1 is modified to specify P_{i+1}.

The remainder of this section describes the simulation.

On a particular input string of length n, suppose T runs for $N' \leqq t(n)$ steps. Because S knows neither n nor N' a priori, S repeats the simulation described in this section for $N = 1, 2, 4, 8, \cdots$, using N for the length of the computation of T, until $N \geqq N'$. Machine S has an auxiliary worktape to record the input symbols that it has read. When $N > 1$, machine S first repeats the simulation of the first $N/2$ steps of T, obtaining the first $N/2$ input symbols from the auxiliary worktape. Then S simulates the next $N/2$ steps of T, recording on the auxiliary worktape the new input symbols read during steps $N/2+1$ through N. Furthermore, on the output tape, S writes only the symbols generated during steps $N/2+1$ through N, avoiding repetitious outputs. Thus, the input and output heads of S move only when it simulates steps $N/2+1$ through N. During its computation, S marks all cells that its worktape heads visit. Before embarking on the simulation with the next value of N, it erases its worktapes via a depth-first traversal of the marked cells. We shall show that for each N, the simulation with that value of N takes time $t_0(N) = O(N^{1+1/d}/\log N)$. It follows that S simulates T on-line in time

$$t_0(1) + t_0(2) + \cdots + t_0(2^{\lceil \log N' \rceil}) = O\left(\frac{t(n)^{1+1/d}}{\log t(n)}\right).$$

Assume that $N \geqq 28(h+1)4^d$ and $N \geqq 4^{5d}$. (Smaller cases are trivial.) Choose r so that

$$r4^r \leqq N^{1/d} \leqq (r+1)4^{r+1}.$$

The storage alphabet of S is so large that the coordinates of every cell of S within distance dN of the origin can be encoded by a string of at most r symbols. Call the encoded coordinates of a cell the cell's *position*. If cell X is at distance $s \leqq dN$ from the origin of its worktape, then the encoding permits S to use the position of X to send an access head from the origin to X in at most time proportional to s. The *position* of a box is the position of its base cell.

Cover W with overlapping trees $W[b, 2r+1]$ such that the length of b is a multiple of $r+1$. Call these trees *blocks*, and define $W_b = W[b, 2r+1]$. Every block has $2^{2r+2} - 1 \leqq 4^{r+1}$ cells. Call $W[b, r]$ the *upper half* of block W_b and the remainder of the block its *lower half*. By definition, if a cell X is at distance at least $r+1$ from the root, then X belongs to exactly two blocks: X is in the upper half of one block and in the lower half of another. The *immediate ancestor block* of W_b is the block W_a such that $W(a)$ is the ancestor of $W(b)$ at distance $r+1$ from $W(b)$; equivalently, the location a is the initial segment of b for which $|b| = |a| + r + 1$. An *immediate descendant block* of W_b is a block W_{bc} such that $W(bc)$ is a descendant of $W(b)$ at distance $r+1$ from $W(b)$; equivalently, $|c| = r+1$.

Define the function τ by

$$\tau(z) = 2^{\lceil \log z \rceil};$$

$\tau(z)$ is the smallest integral power of 2 such that $\tau(z) \geqq z$. Define u and the function π by

$$u = \tau((28(h+1)2^d N)^{1/d}), \qquad \pi(m) = \tau((7m)^{1/d}).$$

Observe that the volume of a box of side $\pi(m)$ is at least $7m$, and that $u = O(N^{1/d})$.

To maintain the simulated contents of blocks, S has a box of side u called the *mass store*; the base cell of the mass store is the origin. By definition, since $N \geqq 28(h+1)4^d$ and $d \geqq 2$, $u \leqq N$. Consequently, every cell in the mass store is within distance dN of the origin, and its position can be written with at most r symbols. The mass store contains *pages*. A page P is a box whose side is a power of 2 and whose contents are organized into a *path string*, an *Ancestor-pointer*, a *First-pointer* and a *Current-pointer*.

We describe how the path string of P represents the contents of a block $B = W_b$. The path string has symbols from three disjoint alphabets: Δ (the storage alphabet of T); three *shift symbols* for the shifts on the tree worktape; and *position symbols* to specify positions on the d-dimensional worktape. A symbol occurring in the path string *visits* cell X of W if a head that starts at $W(b)$ and shifts according to the shift symbols preceding that occurrence arrives at X. The path string *represents* the contents of the lower half of B if the following two conditions hold. First, for every nonblank cell X in the *lower half* of B, the path string has exactly one occurrence of a symbol in Δ that visits X, and X contains this symbol. Second, for every binary string x, the path string has at most one nonnull substring of contiguous position symbols that visit $W(bx)$; this substring has length at most r, and P has a *Descendant-pointer* whose value Descendant (P, x) is this string of position symbols. Each Descendant-pointer is part of the path string. A path string might have no Descendant-pointers.

The Ancestor-pointer, First-pointer, and Current-pointer specify positions of pages in the mass store. Write Ancestor (P), First (P), and Current (P) for the values of these pointers. For convenience, we identify a page with its position. For instance, Current (First (P)) is the value of the Current-pointer of the page at position First (P). As outlined at the beginning of § 3, the Current-pointers and First-pointers implement an indirect addressing scheme that enables S to locate the current page that corresponds to B. We assume a standard format for pages such that a page of volume $L+3r$ can accommodate a path string of length L and the Ancestor-pointer, First-pointer and Current-pointer, which together occupy at most $3r$ cells.

For configurations of S we define a *correspondence* between pages and blocks.

(i) There is exactly one page marked with a special symbol. This page corresponds to W_λ.

(ii) Let page P correspond to W_b. If Descendant (P, c) is nonnull, then Current (First (Descendant (P, c))) corresponds to W_{bc}.

Throughout the simulation, the following will be true for a page P that corresponds to a block $B = W_b$.

Invariant 1. The path string of P represents the contents of the lower half of B.

Invariant 2. If the path string of P visits v distinct cells of B, then the path string has at most $2v$ shift symbols and at most v Δ symbols. The cells of B visited by the path string are nonblank.

Invariant 3. The page Current (First (Ancestor (P))) corresponds to the immediate ancestor block of B. If $B' = W_{bc}$ is an immediate descendant block of B and the lower half of B' is nonblank, then Current (First (Descendant (P, c))) corresponds to B'.

Invariant 4. If $B = W_\lambda$, then for every nonblank cell X in B (not just in the lower half), the path string of P has exactly one occurrence of a symbol in Δ that visits X, and X contains this symbol.

Let $H_1, \cdots, H_h$ be the h access heads of T. To maintain the simulated access head locations, S has several *head location tapes* named L_{i1}, L_{i2} for $i = 1, \cdots, h$; they are used as linear tapes. On tape L_{i1} the location of H_i is written in consecutive contiguous substrings called *segments*; all segments preceding the last have length $r+1$, and the last has length between 0 and r. Because the head locations are broken into segments, S can avoid reading the entire location when it simulates one step. Tape L_{i2} is used as a unary counter with values 0 to $2r+1$.

Suppose H_i is located at cell $X = W(bx)$ in a block B rooted at $W(b)$. Let page P correspond to B. The value of the contents of L_{i2} indicates whether H_i is in the upper half or the lower half of B. Suppose S wishes to determine whether a symbol in the path string of P visits X. Depending on the value of L_{i2}, S copies the last one or two segments from L_{i1} to a spare worktape. While one head scans along the path string, S uses the information on this spare worktape to decide in a routine fashion which symbols of the path string visit X. If X is in the lower half of B, then S can retrieve the symbol that X contains and also the value Descendant (B, x).

To record the nonblank symbol δ that H_i writes on X, machine S first discovers whether a Δ symbol in the path string visits X, and if so, then S changes this symbol to δ. If no Δ symbol in the path string visits X, then S determines the ancestor Y closest to X that the path string visits in B. Let Y be at distance $s \geqq 1$ from X. Then S produces a new path string that differs from the old one only by the insertion of a string of $2s+1$ consecutive symbols: s shift symbols for the shifts from Y to X, the symbol δ (which visits X) and s shift symbols for the shifts from X to Y. If the original path string visited v distinct cells of B and had at most $2v$ shift symbols, then the new path string visits $v+s$ cells, including X, and has at most $2v+2s = 2(v+s)$ shift symbols. Throughout the computation of T, the nonblank cells of W form a connected set. Thus, if X and Y are nonblank, then all the $s-1$ cells between Y and X are also nonblank. Consequently, if the old path string visited only nonblank cells of B, then so does the new path string. We conclude that S can maintain Invariant 2.

The simulator S has heads $G_{u1}, \cdots, G_{uh}$, $G_{l1}, \cdots, G_{lh}$ on its mass store. The simulation begins with all these heads on a page P_0 of side $\pi(r)$ that corresponds to W_λ. Initially, Ancestor (P_0) = First (P_0) = Current $(P_0) = P_0$, and the path string of P_0 is empty.

In general, to simulate head H_i on cell X_i in block $B_i = W_{b_i}$, head G_{ui} is in page P_i and head G_{li} in page Q_i such that Q_i corresponds to B_i and $P_i =$ Current (First (Ancestor (Q_i))) corresponds to the immediate ancestor block A_i of B_i. If X_i is in the lower half of B_i, then S uses the path string of Q_i read by G_{li} to retrieve

the contents of X_i. If the path string of Q_i does not visit X_i, then X_i holds a blank symbol. If X_i is in the upper half of B_i and $b_i \neq \lambda$, then X_i is in the lower half of A_i, and S retrieves the contents of X_i from the path string of P_i read by G_{ui}. (If $b_i = \lambda$, then according to Invariant 4, the path string of Q_i has the contents of X_i even when X_i is in the top half of B_i.) To simulate the effect of one step of T, machine S records the new contents of each X_i in the appropriate path string (of P_i or of Q_i). If T shifts its input head or writes an ouput symbol at the end of this step, then S does the same. Finally, S updates the head location tapes. When S completes the simulation of this step, Invariant 1 holds: the path string of P_i represents the contents of the lower half of A_i, and the path string of Q_i represents the contents of the lower half of B_i.

Suppose H_i shifts from $W(b_i)$ to its parent. Let M = Current (First (Ancestor (P_i))). Machine S sends G_{li} from Q_i to P_i and sends G_{ui} from P_i to M, which corresponds to the immediate ancestor block of A_i, in whose lower half the parent of $W(b_i)$ is located. Call this adjustment of access heads an *upward reorientation* of G_{ui} and G_{li}.

If H_i shifts to a child of a leaf $W(b_icx)$ of B_i, where $|c| = r+1$, then S performs a *downward reorientation* of G_{ui} and G_{li} by sending G_{ui} from P_i to Q_i and G_{li} from Q_i to R = Current (First (Descendant (Q_i, c))), provided that Q_i has a Descendant (Q_i, c) pointer. If R is not defined, then the bottom half of W_{b_ic} is completely blank (by Invariant 3), although r cells in its top half (on a path from $W(b_ic)$ to $W(b_icx)$) must be nonblank. The storage allocation procedure ALLOCATE (described in § 5) produces a new completely blank page R of side $\pi(r)$ in the mass store. Next, on this new page S sets First (R) $\leftarrow R$, Current (R) $\leftarrow R$, Ancestor (R) $\leftarrow Q$. Also, S sets Descendant (Q_i, c) $\leftarrow R$ and sends G_{li} to R. After this initialization, R corresponds to W_{b_ic}, and Invariant 3 holds.

Now suppose that when S adds further symbols to the path string of a page P to record the contents of a cell in the corresponding block B, P is not large enough to contain the updated path string. With a call to ALLOCATE, S finds a new unused box P' in the mass store whose side is a power of 2 such that P' is just large enough to hold the new path string (as well as the Ancestor-pointer, First-pointer and Current-pointer). Then S writes the updated path string into P' and sets First (P') $\leftarrow$ First (P), Current (First (P)) $\leftarrow P'$ and Ancestor (P') $\leftarrow$ Ancestor (P). Page P' now corresponds to B, and Invariant 3 remains valid.

4. Analysis of the simulation. First, we establish upper bounds on the volumes of pages. Throughout this section we may assume that at least r cells of W_λ are nonblank.

LEMMA 1. *In a configuration of S during the simulation, let page P correspond to simulated block $B = W_b$ with m nonblank cells. Then the length of the path string of P is at most $4m$, and the side of P is at most $\pi(m)$.*

Proof. Invariant 2 guarantees that the path string of P has at most $2m$ shift symbols and at most m symbols in Δ. If P has a Descendant (P, c) pointer, then by Invariant 3, the lower half of block W_{bc} is nonblank; since the nonblank cells of W always form a connected set, at least r cells of $W[bc, r]$ in the lower half of B are nonblank. Consequently, the path string has at most m/r Descendant-pointers, each of length r. We have deduced that the length of the path string is at most $2m + m + (m/r)r = 4m$.

The Ancestor-pointer, First-pointer and Current-pointer together occupy at most $3r$ cells. Because a page is allocated only when the corresponding block has at least r nonblank cells, $m \geqq r$. Therefore, the volume of P is at most $4m + 3r \leqq 7m$, and the side of P is at most $\pi(m)$. □

LEMMA 2. *Throughout the simulation, the total volume of pages in the mass store is at most* $28h2^dN$.

Proof. At an arbitrary configuration of the simulation, let $P_1, P_2, \cdots$ be the pages that correspond to blocks on W. Let P_j correspond to block B_j, and let B_j have m_j nonblank cells. Since every cell of W belongs to at most two blocks and since the access heads of T can visit at most hN cells of W,

$$\sum_j m_j \leqq 2hN.$$

The mass store holds smaller pages that correspond to B_j in previous configurations of S. The volumes of these smaller pages are distinct powers of 2. It follows that the total volume of pages that have ever corresponded to B_j is at most twice the volume of P_j. Ergo, by Lemma 1, the total volume of all pages in the mass store is

$$\sum_j 2(\pi(m_j))^d \leqq 2 \cdot 2^d \sum_j (7m_j) \leqq 28h2^dN. \qquad \square$$

The time used by S to simulate one step of T is proportional to the length of the h path strings that it handles. Lemma 1 implies that every path string has length at most $O(4^r)$. Thus, to simulate N individual steps, S spends time $O(N4^r) = O(N^{1+1/d}/\log N)$ updating path strings and head location tapes.

After an upward or a downward reorientation of G_{ui} and G_{li}, the simulated head H_i is at distance r or $r+1$ from both the root and the leaves of a block. Consequently, this head can induce at most N/r reorientations. Each upward reorientation takes time $O(r) = O(\log N)$ to retrieve the position of another page and time $O(u) = O(N^{1/d})$ to move the heads across the mass store. For a downward reorientation, S may spend, in addition, time $O((\log N)^2)$ for a call to ALLOCATE (as we show in § 5). Therefore, the total time for reorientations is at most

$$\left(\frac{N}{r}\right) O((\log N)^2 + N^{1/d}) = O\left(\frac{N^{1+1/d}}{\log N}\right).$$

When S copies the contents of a page P to a larger page P', it spends time $O(u) = O(N^{1/d})$ to move the heads across the mass store, time $O((\log N)^2)$ for a call to ALLOCATE, and time $O(4^r + r)$ to copy the path string and pointer values. Since every page has volume at least r, Lemma 2 implies that S makes at most $O(N/r)$ allocations of pages. Thus, S spends time

$$O\left(\left(\frac{N}{r}\right)(N^{1/d} + 4^r + (\log N)^2)\right) = O\left(\frac{N^{1+1/d}}{\log N}\right)$$

finding and initializing new pages.

In summary, the simulator uses time $O(N^{1+1/d}/\log N)$ to simulate N individual steps, time $O(N^{1+1/d}/\log N)$ to reorient heads G_{ui} and G_{li}, and time $O(N^{1+1/d}/\log N)$ to prepare new larger pages in the mass store.

THEOREM 1. *Every multihead tree machine of time complexity* $t(n)$ *can be simulated on-line by a multihead d-dimensional machine in time* $O(t(n)^{1+1/d}/\log t(n))$.

Consider the logarithmic cost random access machines endowed only with addition and subtraction. Since each of these machines can be simulated on-line in linear time by a multihead tree machine [10], we obtain an immediate corollary of Theorem 1.

THEOREM 2. *Every logarithmic cost random access machine with addition and subtraction of time complexity* $t(n)$ *can be simulated on-line by a multihead d-dimensional machine in time* $O(t(n)^{1+1/d}/\log t(n))$.

5. Storage allocation. Machine S has a *free storage list*, a list of positions of blank boxes in the mass store. Initially, the free storage list holds the position of the mass store itself, a single box of side u. Throughout the computation of S, after the first call to ALLOCATE, for $q = 1, 2, \cdots, u/2$, this list has positions of at most $2^d - 1$ boxes of side q.

Procedure ALLOCATE.
Input: p, a power of 2.
Output: The position of a blank box of side p in the mass store.
Method: A buddy system [5] is used.
Step 1. If the free storage list has a box of side p, then skip to Step 2. Otherwise, let q^* be the smallest power of 2 for which the free storage list has a box of side q^*. (We shall show that when ALLOCATE is called during the simulation, q^* must exist.) For $q = q^*, q^*/2, \cdots, 4p, 2p$ in order, select the position of a box Q_q of side q and delete this position from the list; add to the list the positions of the 2^d disjoint boxes of side $q/2$ whose union is Q_q.
Step 2. Let y be the position of a box of side p on the free storage list; delete y from the list, and return the value y.

The time taken by ALLOCATE is $O(r \log u) = O((\log N)^2)$ because at most $(2^d - 1) \log u$ positions of length at most r are handled.

Let $q_1 \leqq q_2 \leqq \cdots q_s$ be the sides of boxes whose positions are on the free storage list when ALLOCATE is called to produce a blank box of side $\pi(m)$, where $r \leqq m \leqq 4^{r+1}$. Since $N \geqq 4^{5d}$ implies $r \geqq 4$, it follows that

$$m \leqq r4^r \leqq N^{1/d}. \tag{1}$$

Furthermore, since $m \geqq r \geqq 4$,

$$28m^d \geqq 2^d(7m). \tag{2}$$

Lemma 2 and the definition of u imply that

$$q_s^d + \cdots + q_1^d \geqq u^d - 28h2^dN \geqq 2^d(28N). \tag{3}$$

Since the free storage list has at most $2^d - 1$ boxes of each distinct side,

$$q_s^d + \cdots + q_1^d \leqq (2^d - 1)q_s^d + (2^d - 1)\left(\frac{q_s}{s}\right)^d + \cdots + (2^d - 1)(1) < (2q_s)^d. \tag{4}$$

Combining inequalities (1), (2), (3) and (4) yields

$$q_s \geqq (28N)^{1/d} \geqq (28)^{1/d}m \geqq 2(7m)^{1/d} \geqq \pi(m),$$

and ALLOCATE can find a box of side $\pi(m)$ in the mass store.

6. Lower bound. We define a tree machine T' and demonstrate that every d-dimensional machine that simulates T' on-line requires time $\Omega(N^{1+1/d}/\log N)$.

Machine T' has just one access head on one tree worktape and operates in real time. Its input alphabet is a set of *commands* of the form $\langle e, \sigma \rangle$, where $e \in \{0, 1, ?\}$ and σ is a shift for a tree worktape. Suppose T' is in a configuration in which the cell X at which the access head is located contains e'. On input $\langle e, \sigma \rangle$, machine T' writes e' on its output tape, and the access head writes e on X if $e \in \{0, 1\}$, but writes e' on X (its current contents) if $e = ?$. At the end of the step the access head executes the shift σ.

Let d-dimensional machine S' simulate T' on-line. To establish the lower bound $\Omega(N^{1+1/d}/\log N)$ on the time required by S', we formalize the following volumetric argument. The worktape head of T' can access each of a set of N cells within $\log N$ steps, whereas the heads of S' require $\Omega(N^{1/d})$ steps to access one of a set of N cells in the worst case. If the contents of a set of N cells of the worktape of T' are sufficiently random (in a sense made precise below), then there is a sequence of $\Omega(N/\log N)$ "hard questions" about the contents of these cells, each question having length $\log N$, such that S' requires time $\Omega(N^{1/d})$ to answer each question.

Let $\#$ be a new symbol. For strings x, y in $\{0, 1, \#\}^*$, let $K(x|y)$ be the *Kolmogoroff complexity of x given y* with respect to a fixed universal function U. Formally, $K(x|y)$ is the length of the shortest *binary* string b such that $U(b \# y) = x$. Intuitively, b is a binary description of x, given y. Write $K(x)$ for $K(x|\lambda)$, where λ is the empty string. The following elementary properties of K are well known: There is a fixed constant c such that for all x and y,

$$K(x) \leqq 2|x| + c, \qquad K(x) \leqq K(x|y) + K(y) + c.$$

Call a binary string x for which $K(x) \geqq |x|$ *incompressible*. For every n, since there are 2^n binary strings of length n but only $2^n - 1$ possible shorter binary descriptions, there exists at least one incompressible binary string of length n.

LEMMA 3. *Let $h \geqq 1$ and let x be an incompressible string of length $N > 8(c+h)$. For every set of h strings $\{y_1, \cdots, y_h\}$ of length at most $N/4h$ each, $K(x|y_1 \# \cdots \# y_h) > N/4$.*

Proof. If not then

$$K(x) \leqq K(x|y_1 \# \cdots \# y_h) + K(y_1 \# \cdots \# y_h) + c \leqq \frac{N}{4} + (2h)\left(\frac{N}{4h} + 1\right) + 2c < N.$$

This is a contradiction. □

THEOREM 3. *Let d-dimensional machine S' with head-to-head jumps on one worktape simulate T' on-line in time $t'(N)$. Then $t'(N) = \Omega(N^{1+1/d}/\log N)$.*

Proof. For every N that is a sufficiently large power of 2, we shall construct a string of N input commands on which S' requires time $\Omega(N^{1+1/d}/\log N)$. The input string will have a filling part Q_0 of length $N/2$ followed by a query part of length $N/2$.

Let W be the worktape of T', and let $W_\lambda = W[\lambda, \log_2 (N/8)]$. The filling part compels the head of T' to write on the $(N/4) - 1$ cells of W_λ such that a depth-first traversal of the contents of W_λ gives an incompressible string x of length $(N/4) - 1$.

A *question* is a string of $2 \log_2 N$ commands of the form $\langle ?, \sigma \rangle$ that drives the head of T' from the root $W(\lambda)$ to a cell of W_λ and back to the root. Note that the contents of W_λ remain unchanged when T' processes a question. The query part will be a sequence of $N/(4 \log_2 N)$ questions $Q_1, Q_2, \cdots$. We shall choose the questions so that S' spends time $\Omega(N^{1/d})$ to process each Q_j.

Let S' have h access heads. For $j \geqq 0$, consider the configuration of S' after it has processed Q_j. Let $B_1, \cdots, B_n$ be the boxes of side $(N/(32c'h))^{1/d}$ centered at the heads in this configuration, where the constant c' depends on S' and is specified later. These boxes hold all the cells accessed by S' during the next $(N/(32c'h))^{1/d}/2$ steps. We now show that there is some question Q_{j+1} that forces some head of S' to exit $B_1 \cup \cdots \cup B_h$ when S' processes Q_{j+1}. Otherwise, let y_i be a binary encoding of the contents of B_i and z_i be a binary encoding of the relative position of access head i in B_i. Evidently, if c' is sufficiently large, then both $|y_i| \leqq c'|B_i|$ and $|z_i| \leqq c'|B_i|$ for every i. From the string $y_1 \# \cdots \# y_h \# z_1 \# \cdots \# z_h$, only a small constant amount of additional information (essentially a binary description of this discussion) is necessary

to generate x because S' can process every question Q_{j+1} with the heads remaining in $B_1 \cup \cdots \cup B_h$. We deduce that $K(x|y_1 \# \cdots \# y_h \# z_1 \# \cdots \# z_h) = O(1)$, contravening Lemma 4.

Therefore, since some head spends time $(N/(32c'h))^{1/d}/2$ to exit $B_1 \cup \cdots \cup B_h$ when S' processes question Q_{j+1}, the time spent by S' on the query part alone is at least

$$\left(\frac{N}{4 \log N}\right)\left(\frac{N}{32c'h}\right)^{1/d}\Big/2 = \Omega\left(\frac{N^{1+1/d}}{\log N}\right). \qquad \square$$

Acknowledgment. A referee simplified the constants in the definitions of π and u.

REFERENCES

[1] S. A. COOK AND S. O. AANDERAA, *On the minimum computation time of functions*, Trans. Amer. Math. Soc., 142 (1969), pp. 291–314.

[2] R. A. DE MILLO, S. C. EISENSTAT AND R. J. LIPTON, *Space-time tradeoffs in structured programming: An improved combinatorial embedding theorem*, J. Assoc. Comput. Mach., 27 (1980), 123–127.

[3] D. JU. GRIGORIEV, *Imbedding theorems for Turing machines of different dimensions and Kolmogorov's algorithms*, Soviet Math. Dokl., 18 (1977), pp. 588–592.

[4] F. C. HENNIE, *On-line Turing machine computations*, IEEE Trans. Elec. Comp., EC-15 (1966), pp. 35–44.

[5] D. E. KNUTH, *The Art of Computer Programming, Vol.* 1: *Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.

[6] R. J. LIPTON, S. C. EISENSTAT AND R. A. DE MILLO, *Space and time hierarchies for classes of control structures and data structures*, J. Assoc. Comput. Mach., 23 (1976), pp. 720–732.

[7] M. C. LOUI, *Simulations among multidimensional Turing machines*, Theoret. Comput. Sci., 21 (1982), pp. 145–161.

[8] ———, *Minimizing access pointers into trees and arrays*, Rep. R-910, ACT-27, Coordinated Sci. Lab., Univ. of Illinois, Urbana, June 1981.

[9] N. A. LYNCH, *Accessibility of values as a determinant of relative complexity in algebras*, J. Comput. System Sci., 24 (1982), pp. 101–113.

[10] W. J. PAUL AND R. REISCHUK, *On time versus space* II. J. Comput. System Sci., 22 (1981), pp. 312–327.

[11] W. J. PAUL, J. I. SEIFERAS AND J. SIMON, *An information-theoretic approach to time bounds for on-line computation*, J. Comput. System Sci., 23 (1981), pp. 108–126.

[12] N. PIPPENGER AND M. J. FISCHER, *Relations among complexity measures*, J. Assoc. Comput. Mach., 26 (1979), pp. 361–381.

[13] K. R. REISCHUK, *A fast implementation of a multidimensional storage into a tree storage*, Theoret. Comput. Sci., 19 (1982), pp. 253–266.

[14] A. L. ROSENBERG, *Data encodings and their costs*, Acta Informatica, 9 (1978), pp. 273–292.

[15] ———, *Encoding data structures in trees*, J. Assoc. Comput. Mach., 26 (1979), pp. 668–689.

[16] A. L. ROSENBERG AND L. SNYDER, *Bounds on the costs of data encodings*, Math. Systems Theory, 12 (1978), pp. 9–39.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0005 $01.25/0

A GENERALIZED CLASS OF POLYNOMIALS THAT ARE HARD TO FACTOR*

ERICH KALTOFEN,† DAVID R. MUSSER,‡ AND B. DAVID SAUNDERS†

Abstract. A class of univariate polynomials is defined which make the Berlekamp–Hensel factorization algorithm take an exponential amount of time. This class contains as subclasses the Swinnerton-Dyer polynomials discussed by Berlekamp and a subset of the cyclotomic polynomials. Aside from shedding light on the complexity of polynomial factorization this class is also useful in testing implementations of the Berlekamp–Hensel and related algorithms.

Key words. polynomial factorization, exponential time, Berlekamp–Hensel algorithm, Galois groups, computer algebra

1. Introduction and summary of results. This paper generalizes a class of univariate polynomials with integral coefficients attributed to H. P. F. Swinnerton-Dyer by E. R. Berlekamp [BERL70, p. 733]. We use Galois theoretical methods to prove their properties of interest.

These polynomials are of particular interest for the Berlekamp–Hensel factorization algorithm [KNUD81, p. 433], which determines factors modulo p and lifts them to find the integral factors of a polynomial. Because the polynomials in the class we will define are irreducible over the integers but have a large number of factors modulo p for every prime p, the Berlekamp–Hensel algorithm behaves badly on them: In determining their irreducibility it needs a number of operations that is exponential in the degree and coefficient lengths of the polynomials. This was also true for the Swinnerton-Dyer polynomials, but the class we define is much larger and contains subclasses that, unlike the Swinnerton–Dyer polynomials could make a hypothetical algorithm (see § 4) remain super-polynomial in its computing time.

Let n be a positive integer and let r be an integer $\geqq 2$. By ζ_r we denote $\exp(2\pi i/r)$, the first primitive rth root of unity. Let $p_1, \cdots, p_n$ be n distinct positive prime numbers. By $f_{r;p_1,\cdots,p_n}(x)$ we denote the monic univariate polynomial in x whose roots are $\zeta_r^{i_1}(p_1)^{1/r} + \cdots + \zeta_r^{i_n}(p_n)^{1/r}$ with $1 \leqq i_1, \cdots, i_n \leqq r$.

All $f_{r;p_1,\cdots,p_n}$ have integral coefficients and are irreducible polynomials of degree r^n over the integers. If r is a prime number the following will be shown: If the coefficients of $f_{r;p_1,\cdots,p_n}$ are projected into a field of residues modulo any prime number q, henceforth denoted by Z_q, the image polynomials $f_{r;p_1,\cdots,p_n} \pmod q$ factor into irreducible polynomials over Z_q which have degree at most r.

If $r = 2$ this construction gives a slightly simpler version of the Swinnerton-Dyer polynomials which treat $\sqrt{-1}$ as an additional prime number. But our Galois theoretical proofs can be easily extended to yield this special case.

The condition of r being a prime number is not crucial for the unpleasant running time behavior for the factorization of these polynomials. For composite r the degrees of the irreducible factors in the modular domain are then bounded by r^2 (we will actually prove a somewhat better bound).

A modified version of these polynomials is also presented because of its closely related properties: By $f^*_{r;p_1,\cdots,p_n}$ we denote the polynomial whose roots are $\zeta_r^{i_0} +$

* Received by the editors March 8, 1981.

† Rensselaer Polytechnic Institute, Troy, New York, 12181. The research of these authors was supported by the National Science Foundation under grant MCS-7909158.

‡ Computer Science Branch, General Electric Research & Development Center, Schenectady, New York.

$\zeta_r^{i_1}(p_1)^{1/r}+\cdots+\zeta_r^{i_n}(p_n)^{1/r}$ where $1\leqq i_0, i_1, \cdots, i_n \leqq r$ and $(i_0, r)=1$, meaning that the greatest common divisor of i_0 and r is 1.

Again all $f^*_{r;p_1,\cdots,p_n}$ are integer polynomials which factor modulo any prime q into polynomials whose degrees are bounded as for $f_{r;p_1,\cdots,p_n}$. If r is 2, 4, 6 or an odd integer, $f^*_{r;p_1,\cdots,p_n}$ is also irreducible over the integers. Otherwise these polynomials may be reducible but we can guarantee that all factors over the integers are of degree at least $2r^n$.

If $n=0$, $f^*_{r;\varnothing}$ are the cyclotomic polynomials $\Psi_r(x)$. We will show that for certain composite r the maximum degree of factors in any residue field implies a superpolynomial running time for the Berlekamp–Hensel factorization algorithm. This fact is discussed in [MUSD75, p. 302]. D. Knuth [KNUD81, p. 437] uses Berlekamp's algorithm to prove the modular factorization property for Ψ_8. Because these polynomials can be factored modulo any prime number, P. Weinberger's and L. Rothschild's algorithm [WERO76] for factoring integral polynomials over these polynomials' splitting fields becomes exponential complex. L. Kronecker's procedure [VdWA53, pp. 136–137], which depends on factoring an integral polynomial constructed from the input polynomial and the minimal polynomial of a primitive element for the algebraic number field, over which the factorization is to be computed, also becomes very inefficient, if the algebraic number field is the splitting field of our polynomials.

In § 2 we introduce the number theory and Galois theory required to prove our main theorems in § 3. In § 4 we give examples of these polynomials and additionally analyze the computing time required to factor them. We conclude in § 5 with a recap of the Galois theoretical considerations underlying our results.

Notation. By Z we denote the integers and by Q the rational numbers.

2. Number theoretical and Galois theoretical background. Let r be an integer $\geqq 2$. By U_r we denote the set of residues modulo r which are relatively prime to r. This set forms a group under multiplication modulo r and there exists a minimal nonnegative integer $\lambda(r)$ such that for each $s\in U_r$: $s^{\lambda(r)}\equiv 1 \pmod r$. $\lambda(r)$ is called the minimum universal exponent modulo r. It is known (see [SIEW64, p. 246]) that

$$\lambda(2)=1, \quad \lambda(4)=2, \quad \lambda(2^\alpha)=2^{\alpha-2} \quad \text{for } \alpha\geqq 3,$$

$$\lambda(2^{\alpha_0}q_1^{\alpha_1}\cdots q_n^{\alpha_n})=\operatorname{lcm}(\lambda(2^{\alpha_0}),\phi(q_1^{\alpha_1}),\cdots,\phi(q_n^{\alpha_n}))$$

where the q_i are distinct odd prime numbers, ϕ is Euler's totient function and lcm means the least common multiple. Let p_i be the ith consecutive prime number. As a consequence of Chebyshev's theorem $p_i<2^i$ for all $i>1$ [SIEW64, p. 138]. This enables us to prove the following:

LEMMA 1. *Let j be an integer $\geqq 2$. Then there are infinitely many positive integers m (namely the product of the first k odd prime numbers with k sufficiently large) such that*

$$\frac{\phi(m)}{\lambda(m)}>(\log_2(\phi(m)))^j.$$

Proof. Let $m=p_2\cdots p_k$. Then $\phi(m)=(p_2-1)\cdots(p_k-1)<2^{k(k+1)/2-1}$ by the above estimate for p_i. Therefore $(\log_2(\phi(m)))^j<k^{3j}$. Also $\lambda(m)=\operatorname{lcm}(p_2-1,\cdots,p_k-1)<2(p_2-1)/2\cdots(p_k-1)/2=2^{2-k}\phi(m)$. Hence $\phi(m)/\lambda(m)>2^{k-2}>k^{3j}$ for k chosen large enough. Therefore for all sufficiently large k: $\phi(m)/\lambda(m)>(\log_2(\phi(m)))^j$. □

In the proof of Theorem 2 below we will make use of the fact that for every prime number r and for all $s \in U_r - \{1\}$: $(s^{r-1}-1)/(s-1)$ is a multiple of r. This follows from the Fermat theorem $(a^{\phi(b)} \equiv 1 \pmod b)$ for $(a, b) = 1$ and the fact that r is a prime number. In order to treat composite r we generalize this matter:

LEMMA 2. *Let r be a positive composite integer. By $\eta(r)$ we denote the minimum exponent such that for each $s \in U_r - \{1\}$: $(s^{\eta(r)}-1)/(s-1)$ is divisible by r. Then $\eta(r) \leqq r\lambda(r)$. In fact, $\eta(r) \leqq d\lambda(r)$ where $d = \operatorname{lcm}(\{(s-1, r): s \in U_r - \{1\}\})$.*

Proof. Since for any s, $(s-1, r)$ divides r so must d and therefore $d \leqq r$. We claim that $(s^{d\lambda(r)}-1)/(s-1)$ is a multiple of r: To prove this we first factor $s^{d\lambda(r)}-1$ as

$$(s^{\lambda(r)}-1)(s^{(d-1)\lambda(r)} + s^{(d-2)\lambda(r)} + \cdots + 1).$$

Now the left factor is a multiple of r. It is therefore sufficient to show that the right factor is a multiple of d since that means it can absorb any factor of r in $s-1$ (by definition of d). But $s^{k\lambda(r)} \equiv (s^{\lambda(r)})^k \equiv 1 \pmod d$ for $0 \leqq k \leqq d-1$ since d divides r and thus $(s^{(d-1)\lambda(r)} + \cdots + 1) \equiv d \cdot 1 \equiv 0 \pmod d$, as required. Therefore $\eta(r) \leqq d\lambda(r) \leqq r\lambda(r)$. □

At this point the question arises what the actual value of $\eta(r)$ is. We have not found an explicit formula. In Theorem 2 we will use the fact that $\lambda(d) \leqq \lambda(r)$ for all divisors d of r. This particular property remains true for η as we can see by the following argument, which we owe to A. Odlyzko. Let d be a divisor of r and let t be an integer with $t \neq 1$ and $(t, d) = 1$. We need to show that d divides

$$(t^{\eta(r)}-1)/(t-1), \quad \text{i.e.} \quad t^{\eta(r)-1} + \cdots + 1 \equiv 0 \pmod d.$$

We can find integers m and e such that $r = d^m e$ and $(d, e) = 1$. The last condition guarantees the existence of a residue b modulo e such that $bd \equiv 1 - t \pmod e$. Let $s = t + bd$. Thus $s \equiv t \pmod d$ and $s \equiv 1 \pmod e$ which means that $s \neq 1$ and $(s, r) = 1$. Hence $s^{\eta(r)-1} + \cdots + 1 \equiv 0 \pmod r$, by the definition of η, and taking this equation modulo d yields our claim. Table 1 indicates for some r the values $\eta(r)$ and its bounds from Lemma 2:

TABLE 1

r	4	8	9	15	22	27	35	38	49	$3 \cdot 7^2$	$3^2 \cdot 7^2$
$\eta(r)$	2	8	18	60	10	54	420	18	294	294	882
$d\lambda(r)$	4	8	18	60	20	162	420	36	294	6,174	18,522
$r\lambda(r)$	8	16	54	60	220	486	420	684	2,058	6,174	18,522

We will need some well-known properties of the cyclotomic polynomials later and shall mention them now: Let r be an integer $\geqq 2$ and let ζ_r be a primitive rth root of unity. There always exist $\phi(r)$ distinct primitive rth roots of unity in an extension field of Q or Z_q provided that q is a prime number not dividing r. They are the powers of ζ_r whose exponents are relatively prime to r. Then

$$\Psi_r(x) = \prod_{\substack{i=1 \\ (i,r)=1}}^{r} (x - \zeta_r^i) = \prod_{d|r} (x^d - 1)^{\mu(r/d)}$$

denotes the rth cyclotomic polynomial which has all integer coefficients (or their residues modulo q if the ground field is Z_q). By $d|r$ we mean that d is a divisor of r and μ denotes the Moebius function: $\mu(n) = (-1)^m$ if n is squarefree and has m distinct prime divisors, $\mu(1) = 1$, and otherwise $\mu(n) = 0$. (See [VdWA53, p. 112].)

If $\zeta_r = \exp(2\pi i/r)$ (i.e., the ground field is Q) then Ψ_r is irreducible over Z [VdWA53, p. 162].

LEMMA 3. *Let q be a prime number and let m and r be positive integers such that r is relatively prime to q. Then*

$$\Psi_{r_{q^m}}(x) \equiv \Psi_r(x)^{\phi(q^m)} \pmod{q}.$$

Proof. First we notice the fact that for any integral polynomial f and any integer $i \geqq 0$: $f(x^{q^i}) \equiv f(x)^{q^i} \pmod{q}$. Then by using the formulas for the cyclotomic polynomials and the Moebius function given above the stated congruence can be easily shown. □

By the Galois group of a polynomial we mean the automorphism group of its splitting field over the field of its coefficients. Then the Galois group of Ψ_r over Q is isomorphic to U_r under multiplication modulo r [VdWA53, p. 162].

Let f and g be two monic polynomials whose coefficients lie in some integral domain R. Let α_i, $1 \leqq i \leqq \deg(f)$ and β_j, $1 \leqq j \leqq \deg(g)$ denote their roots respectively. Since the polynomial

$$\prod_{i=1}^{\deg(f)} \prod_{j=1}^{\deg(g)} (x - \alpha_i - \beta_j)$$

is symmetric in both the α_i and the β_j it follows from the fundamental theorem of symmetric functions [VdWA53, p. 79] that its coefficients also lie in R. There is a resultant method which makes it feasible to actually compute this polynomial:

LEMMA 4. *Let R be an integral domain and let f and g be monic polynomials in $R[x]$. Then*

$$(-1)^{\deg(f)\deg(g)} \operatorname{res}_y(f(x-y), g(y))$$

is a monic polynomial in $R[x]$ of degree $\deg(f) \cdot \deg(g)$ whose roots are $\alpha_i + \beta_j$ where α_i $(1 \leqq i \leqq \deg(f))$ are the roots of f and β_j $(1 \leqq j \leqq \deg(g))$ are the roots of g. By res_y we mean the resultant with respect to the indeterminate y (see [VdWA53, p. 84]).

Proof. One may apply a theorem given in [VdWA53, p. 86] for representing the resultant of two polynomials as a symmetric function in their roots. A complete proof is given in [LOOS82]. □

The next two lemmas will help explain why our polynomials split into so many factors modulo any prime number. First we show what happens to the Galois group when an integral polynomial is projected to a polynomial over a residue field.

LEMMA 5. *Let f be a monic separable polynomial in $Z[x]$ and let $\bar{f} \in Z_q[x]$ be its natural projection modulo a prime number q. If $\bar{f}$ is separable (over Z_q) the Galois group of $\bar{f}$ over Z_q is a subgroup (as a permutation group on the suitably arranged roots) of the Galois group of f over Q.*

Proof. See [VdWA53, p. 190]. □

The above lemma has been generalized by [ZASH79] for the case in which $\bar{f}$ is not separable. However, this is only the case if q divides $\operatorname{res}_x(f(x), df(x)/dx)$ over Z, the discriminant of f, and one must avoid those primes in order to be able to perform the Hensel factor lifting algorithm.

LEMMA 6. *Let $\bar{f} \in Z_q[x]$, q being prime. Assume that all elements of the Galois group $\bar{f}$ (as permutations on the distinct roots of $\bar{f}$) are written as products of disjoint cycles. Then $\bar{f}$ does not contain an irreducible factor of degree higher than the length of the longest cycle.*

Proof. The statement follows immediately from the statement made about the generating element of the Galois group of $\bar{f}$ in [VdWA53, p. 191] or likewise from [GAAL73, Thm. 13, p. 230]. □

Now we mention a slight generalization of Eisenstein's irreducibility criterion, which can be used to show the irreducibility of some but not all of our polynomials.

LEMMA 7. *Let* $f(x)=a_0+a_1x+\cdots+a_nx^n\in Z[x]$. *If there exists a prime number* p *and an exponent* i *relatively prime to* n *such that*

$$p^i|a_0, p^i|a_1, \cdots, p^i|a_{n-1}, p\nmid a_n, p^{i+1}\nmid a_0$$

then f *is irreducible over* Z.

Proof. The statement is a special case of Dumas' theorem [VdWA53, p. 76] (with $\alpha=i$ and $\beta=n$). It was first discovered by L. Königsberger [KOEN1895]. □

Notice that the condition $(i, n)=1$ in the above lemma is also necessary, because $x^4+4x^3+8x^2+8x+4=(x^2+2x+2)^2$ yields a counterexample if this is not the case.

The next two lemmas constitute the key for our irreducibility proofs. By $[K:F]$ we denote the degree of a field K over a subfield F and by $F(\theta_1, \cdots, \theta_n)$ we denote the field F extended by the elements $\theta_1, \cdots, \theta_n$.

LEMMA 8. *Let* r *be an integer* $\geqq 2$, ζ_r *a primitive* r*th root of unity, and let* $p_1, \cdots, p_n$ *be distinct positive primes*:

a) $[Q((p_1)^{1/r}, \cdots, (p_n)^{1/r}): Q]=r^n$.

b) *If* $r\geqq 3$ *then* $2r^n\leqq[Q(\zeta_r, (p_1)^{1/r}, \cdots, (p_n)^{1/r}): Q]\leqq\phi(r)r^n$.

c) *If* r *is odd or* 2, 4, *or* 6 *then* $[Q(\zeta_r, (p_1)^{1/r}, \cdots, (p_n)^{1/r}): Q]=\phi(r)r^n$.

Proof. Part a) is proven in [BESI40] well as [RICI74]. Part b) follows immediately from part a) and the fact that for $r\geqq 3$ every ζ_r is a nonreal number of algebraic degree $\phi(r)$ over Q. Part c) is proven for odd r in [RICI74] which is also a special case of [CAVB68, Thm. 10, p. 50]. If $r=2$ part c) is actually the same as part a) because $\zeta_2=-1$. For $r=4$ or 6 we combine part b) and the fact that both $\phi(4)$ and $\phi(6)$ are 2. □

In the above lemma, the primes $p_1, \cdots, p_n$ may be replaced by a more general sequence of values. For example A. S. Besicovitch [BESI40] shows that part a) holds for numbers $p_1, \cdots, p_n$ such that there exist primes $q_1, \cdots, q_n$ satisfying $q_i|p_i$, $q_i^2\nmid p_i$ and $q_i\nmid p_j$, for $i\neq j$. For a still more general condition on the p_i see [NARW74, Lemma 7.11, p. 354].

Also notice that part c) may not hold for even $r\geqq 8$ depending on what primes $p_1, \cdots, p_n$ are chosen. Counterexamples may be constructed using the fact that $\sqrt{2}\in Q(\zeta_8)$ or $\sqrt{5}\in Q(\zeta_{10})$, etc.

LEMMA 9. *Let* r *be an integer* $\geqq 2$, ζ_r *a primitive* r*th root of unity, and let* $p_1, \cdots, p_n$ *be distinct prime numbers. Then* $(p_n)^{1/r}$ *is not an element of the field* $Q(\zeta_r, (p_1)^{1/r}, \cdots, (p_{n-1})^{1/r})$.

Proof. If r is 2 the fact follows from part a) of Lemma 8. By F_k we denote the field $Q(\zeta_r, (p_1)^{1/r}, \cdots, (p_k)^{1/r})$ with $1\leqq k\leqq n$. Now assume that r is $\geqq 3$ and $(p_n)^{1/r}\in F_{n-1}$ which implies that $F_n=F_{n-1}$. Applying part b) of Lemma 8 we get $2r^n\leqq[F_n: Q]=[F_{n-1}: Q]\leqq\phi(r)r^{n-1}$, which is impossible. □

We now summarize some properties of Galois fields which enable us to give an alternate proof of Theorem 2 in § 3: Let $GF(q^n)$ be the splitting field of $x^{q^n}-x$ as a polynomial in x with coefficients in Z_q, q being a prime number. Then $GF(q^n)$ is a finite field with q^n elements of characteristic q whose multiplicative group is cyclic. All fields with q^n elements are isomorphic to $GF(q^n)$ and hence it is called the Galois field with q^n elements. The degree $[GF(q^n): Z_q]$ is n and $GF(q^n)$ has exactly one subfield $GF(q^m)$ provided that m divides n. The automorphism group on $GF(q^n)$ is isomorphic to Z_n under addition, and one of its generators maps each element α of $GF(q^n)$ into α^q (the so-called "Frobenius automorphism") [VdWA53, p. 115].

Let f be an irreducible polynomial of degree n with coefficients in Z_q, q being prime, and let α be a root of f. Since $Z_q(\alpha)$ is isomorphic to $Z_q[x]/(f(x))$, the residues modulo f, $Z_q(\alpha)$ contains q^n elements and thus $\alpha \in GF(q^n)$. The remaining roots of f are $\alpha^q, \cdots, \alpha^{q^{n-1}}$ because of the structure of the Galois group mentioned above.

LEMMA 10. *Let $m \in Z_q$, q being prime, and let r be an integer greater than* 1.

a) *A necessary and sufficient condition for the existence of an rth root of m in Z_q (i.e., a residue b such that $b^r \equiv m \pmod q$) is that r is either relatively prime to $q-1$ or $m^{(q-1)/d} \equiv 1 \pmod q$ where $d = (q-1, r)$ (cf.* [VdWA53, *exer.* 2, *p.* 118]).

b) *The polynomial $x^r - m$ in $Z_q[x]$ has at least one root α such that $\alpha \in GF(q^d)$ and d divides r.*

Proof. a) Let g be a primitive root of Z_q, i.e., a generating element of $Z_q - \{0\}$ with multiplication. Then $g, g^2, \cdots, g^{q-1} = 1$ are distinct residues modulo q. If $(r, q-1) = 1$ then $g^r, g^{2r}, \cdots, g^{(q-1)r} = 1$ are also distinct residues and therefore exactly one element is equal to m. If b is an rth root of m then by Lagrange's theorem $b^{(q-1)r/d} \equiv m^{(q-1)/d} \equiv 1 \pmod q$ where $d = (q-1, r)$. We finally prove that if $m^{(q-1)/d} \equiv 1 \pmod q$ then $m \equiv g^{jr} \pmod q$ for some integer $j \geqq 1$. Assume $m \equiv g^i \pmod q$ and d does not divide i. Then $q-1$ does not divide $i(q-1)/d$ and therefore $m^{(q-1)/d} \equiv g^{i(q-1)/d} \not\equiv 1 \pmod q$. Hence $m \equiv g^{kd} \pmod q$ with $k \geqq 1$. Since $(r/d, q-1) = 1$ there exists a $j \equiv k(r/d)^{-1} \pmod{q-1}$ and therefore $m \equiv g^{kd} \equiv g^{jr} \pmod q$. (Notice that this proof works also if we replace Z_q by $GF(q^n)$.)

b) We use induction on r: For $r = 1$ the statement is trivial. Assume that $r > 1$. We now distinguish two cases:

Case 1. There is a factor $r_1 > 1$ of r such that an r_1th root m_1 of m exists in Z_q. By part a) we already know that this is always true if $r \nmid q-1$. Let $r_2 = r/r_1$. Then $x^r - m \equiv x^{r_2 r_1} - m_1^{r_1} \equiv (x^{r_2} - m_1)(x^{(r_1-1)r_2} + x^{(r_1-2)r_2} m_1 + \cdots + m_1^{r_1-1}) \pmod q$. Applying the induction hypothesis to $x^{r_2} - m_1$ yields the statement for r.

Case 2. r divides $q-1$. Let α be a root of $x^r - m$ and let ζ_r be a primitive rth root of unity both of which lie in some Galois field. Let h be the minimal polynomial of α over Z_q whose constant coefficient be denoted by h_0. Since $h(x)$ divides $x^r - m \equiv (x-\alpha)(x-\zeta_r\alpha)\cdots(x-\zeta_r^{r-1}\alpha)$, it follows that $h_0 = \zeta_r^t \alpha^s$ with $s = \deg(h)$ and t some positive integer. Therefore $h_0^r = m^s$. If $d = (s, r)$ we can find suitable integers u, v such that $us + vr = d$. Then $m^d = m^{us} m^{vr} = h_0^{ur} m^{vr} = (h_0^u m^v)^r$ which implies that m^d possesses an rth root in Z_q. From part a) we conclude that $m^{d(q-1)/r} \equiv 1 \pmod q$ and further that there exists an (r/d)th root of m. If $s < r$ then $r/d > 1$ and we can apply Case 1. Otherwise $x^r - m$ is already irreducible. □

Case 2 of the above proof has an interesting side effect: Let q be a prime and r an integer dividing $q-1$. Then $x^r - m$ is irreducible over Z_q if $m^{(q-1)/d} \not\equiv 1 \pmod q$ for all divisors d of r. As we showed in part a) of the above lemma this is true for all m of the form g^i where g is a primitive root of Z_q and $(i, r) = 1$. Hence by picking a random residue m the probability that $x^r - m$ does not factor in $Z_q[x]$ is $\phi(r)/r$. Choosing an rth degree polynomial randomly only yields a probability of $1/r$ [RABM80]. Moreover it follows from Theorem 328 in [HAWR79, p. 267] that $\phi(r)/r > 0.56/\log\log r$ for almost all r. Therefore in searching for irreducible polynomials in $Z_q[x]$ of degree r, r a divisor of $q-1$, we will succeed considerably sooner by choosing the above polynomials than entirely random ones.

3. Main results.

THEOREM 1. *Let r be an integer $\geqq 2$ and let $p_1, \cdots, p_n$ be distinct prime numbers. Then $f_{r;p_1,\cdots,p_n}$ and $f^*_{r;p_1,\cdots,p_n}$ have integer coefficients and the following irreducibility conditions hold:*

a) *$f_{r;p_1,\cdots,p_n}$ is irreducible over the integers and each irreducible factor of $f^*_{r;p_1,\cdots,p_n}$ over the integers with $r \geqq 3$ has degree at least $2r^n$.*

b) *If $r = 2, 4, 6$ or odd then $f^*_{r;p_1,\cdots,p_n}$ is irreducible.*

Proof. Using Lemma 4 inductively we see that the coefficients of $f_{r;p_1,\cdots,p_n}$ and $f^*_{r;p_1,\cdots,p_n}$ are integers and that their degrees are r^n and $\phi(r)r^n$ respectively. (Notice that Ψ_r has integer coefficients as mentioned in § 1.) First we prove by induction that $(p_1)^{1/r}+\cdots+(p_n)^{1/r}$ is a primitive element of $Q((p_1)^{1/r}, \cdots, (p_n)^{1/r})$. We make use of the construction of a primitive element given in [VdWA53, p. 126]: Let $\alpha_1 = (p_1)^{1/r}+\cdots+(p_{n-1})^{1/r}$ and $\alpha_2, \cdots, \alpha_{r^{n-1}}$ be the remaining roots of $f_{f;p_1,\cdots,p_{n-1}}$. By the induction hypothesis $Q(\alpha_1) = Q((p_1)^{1/r}, \cdots, (p_{n-1})^{1/r})$. The minimal polynomial of α_1 is of degree $[Q(\alpha_1): Q]$ which is r^{n-1} by Lemma 8. Therefore $f_{r;p_1,\cdots,p_{n-1}}$ is this minimal polynomial. Let $\beta_1 = (p_n)^{1/r}, \beta_2, \cdots, \beta_r$ be the roots of $x^r - p_n$ which is irreducible by Eisenstein's criterion (Lemma 7). Then $\alpha_1+\beta_1$ is a primitive element of $Q(\alpha_1, \beta_1) = Q((p_1)^{1/r}, \cdots, (p_n)^{1/r})$ provided that $\alpha_1+\beta_1 \neq \alpha_i+\beta_j$ for $1 \leqq i \leqq r^{n-1}$ and $1 < j \leqq r$. For the sake of contradiction assume that this condition cannot be achieved, namely there exist an i and a $j > 1$ such that $\alpha_1-\alpha_i = \beta_1-\beta_j$. Since $\beta_j = \zeta_r^k (p_n)^{1/r}$ for some $k \geqq 1$ it follows that $\alpha_1-\alpha_i = (p_n)^{1/r}(1-\zeta_r^k)$ and therefore $(p_n)^{1/r} = (\alpha_1-\alpha_i)/(1-\zeta_r^k)$ which is an element of $Q(\zeta_r, (p_1)^{1/r}, \cdots, (p_{n-1})^{1/r})$, in contradiction to Lemma 9.

Noticing that Ψ_r is irreducible we can prove in exactly the same way that $\zeta_r+(p_1)^{1/r}+\cdots+(p_n)^{1/r}$ is a primitive element of $Q(\zeta_r, (p_1)^{1/r}, \cdots, (p_n)^{1/r})$. (However, the α_i will be the roots of an irreducible factor of $f^*_{r;p_1,\cdots,p_{n-1}}$.)

We now conclude that the minimal polynomials of these primitive elements are of the same degree as the field extensions obtained by adjoining them to the rationals which we know by Lemma 8, parts a) and c). Therefore $f_{r;p_1,\cdots,p_n}$ and, in the case that $r = 2, 4, 6$ or an odd integer, $f^*_{r;p_1,\cdots,p_n}$ are these minimal polynomials and hence must be irreducible.

All irreducible factors of $f^*_{r;p_1,\cdots,p_n}$ have degree at least $2r^n$ because all roots are primitive elements by the argument above and the lower bound of the corresponding field extension is known from Lemma 8b). □

THEOREM 2. *Let r be an integer $\geqq 2$ and let $p_1, \cdots, p_n$ be prime numbers. For any prime number q the following factorization properties hold for the projected polynomials $f_{r;p_1,\cdots,p_n} \pmod q$ and $f^*_{r;p_1,\cdots,p_n} \pmod q$:*

a) *The maximum degree of any irreducible factor of both polynomials over the residue field modulo q is at most $r\lambda(r)$. Special case: If r is a prime number the maximum degree is r.*

b) *If $n = 0$ then the maximum degree of an irreducible factor of $f^*_{r;\varnothing} \pmod q = \Psi_r \pmod q$ is $\lambda(r)$.*

Proof. (a) We first show that the length of the longest cycle in any permutation of the Galois group of $f_{r;p_1,\cdots,p_n}$ or $f^*_{r;p_1,\cdots,p_n}$ is at most $\max(r, \eta(r))$, where $\eta(r)$ is as defined in Lemma 2. Let σ be an automorphism on $Q(\zeta_r, (p_1)^{1/r}, \cdots, (p_n)^{1/r})$. As such it has to map the roots of the polynomials Ψ_r and $x^r - p_i$ into roots of the same polynomials. In particular $\sigma(\zeta_r) = \zeta_r^{s_\sigma}$ where ζ_r is a primitive rth root of unity and s_σ is relatively prime to r. Also $\sigma(p_i)^{1/r} = \zeta_r^{m_i}(p_i)^{1/r}$, where the m_i depend also on σ $(1 \leqq i \leqq n)$. (Notice that ζ_r generates all distinct rth roots of unity.) We now distinguish two cases:

Case 1. $s_\sigma = 1$. Applying σ r times we get $\sigma^{(r)}((p_i)^{1/r}) = (p_i)^{1/r}$ for all $1 \leqq i \leqq n$ and therefore $\sigma^{(r)}$ maps each root of $f_{r;p_1,\cdots,p_n}$ and $f^*_{r;p_1,\cdots,p_n}$ onto itself which is to say that the permutation corresponding to σ has cycles of length at most r.

Case 2. $s_\sigma > 1$. By Lemma 2 we know that both $s_\sigma^{\eta(r)} \equiv 1 \pmod r$ and $(s_\sigma^{\eta(r)} - 1)/(s_\sigma - 1) \equiv 0 \pmod r$. A short computation shows that then $\sigma^{(\eta(r))}(\zeta_r) = \zeta_r$ and

$\sigma^{(\eta(r))}((p_i)^{1/r}) = (p_i)^{1/r}$ for all $1 \leq i \leq n$. Therefore the cycle lengths of the permutation corresponding to σ are at most $\eta(r)$. Cases 1 and 2 together prove the statement made initially. If the image polynomials are separable we are finished by virtue of Lemmas 2, 5 and 6.

But we can repeat the above arguments for automorphisms on the splitting field of $f_{r;p_1,\cdots,p_n} \pmod q$ itself because as we mentioned before the properties of rth roots of unity carry over for ground fields of characteristic q, provided that q does not divide r.

Finally let q^m be the highest power of q dividing r. By using the identity introduced in the proof of Lemma 3 and by using Lemma 3 itself we can determine the multiplicities of the roots of $\Psi_r \pmod q$ and $x^r - p_i \pmod q$ (which lie in some Galois field). Therefore $f_{r;p_1,\cdots,p_n} \equiv (f_{r/q^m;p_1,\cdots,p_n})^{q^{mn}} \pmod q$ and $f^*_{r;p_1,\cdots,p_n} \equiv (f^*_{r/q^m;p_1,\cdots,p_n})^{\phi(q^m)q^{mn}} \pmod q$. It follows from the formula for λ given at the beginning of § 2 that $\lambda(r/q^m)$ divides $\lambda(r)$. Then by Lemma 2 and the already proven theorem for the case that q does not divide r we conclude that the maximum degree in this case is $r/q^m\lambda(r/q^m) < r\lambda(r)$. If r is a prime number the above proof together with the remark made above Lemma 2 actually gives the degree bound r.

b) If $\Psi_r \pmod q$ is separable we know its Galois group to be a subgroup of U_r under multiplication modulo r. (This by Lemma 5 but one may verify it directly.) The definition of λ and Lemma 6 then lead to the statement. If $\Psi_r \pmod q$ is inseparable q necessarily divides r. Again putting together the above, Lemma 3 and the fact that $\lambda(r/q^m)$ divides $\lambda(r)$ proves the theorem for this case. □

In special cases the bound $r\lambda(r)$ is actually too pessimistic. Using the remark below Lemma 2, we can actually show that a valid bound is $\max(r, \eta(r))$, which may be considerably smaller than $r\lambda(r)$ (see Table 1 below Lemma 2). By Lemma 10a) we also know that each p_i possesses an rth root in Z_q if r is relatively prime to $q-1$. Then the maximum degree over Z_q can be bounded by $\lambda(r)$ instead. (One uses the same arguments as above but notes that $\sigma((p_i)^{1/r}) = (p_i)^{1/r}$ where $(p_i)^{1/r}$ denotes an rth root of p_i in Z_q.) The second case of Lemma 10a) applies as well.

We conclude this section with a second proof of Theorem 2 expanding ideas from [BERL70, p. 734] with the help of Lemma 10b). However, this method does not introduce the function η and therefore in view of the preceding remarks is somewhat weaker.

Alternate Proof of Theorem 2. If q divides r we must apply the same reduction as in the last part of the previous proof. Now assume that $q \nmid r$. We will show part b) first:

b) Let α be a root of an irreducible factor g of $\psi_r \pmod q$. Then g is separable and the remaining roots are $\alpha^q, \alpha^{q^2}, \cdots, \alpha^{q^{\deg(g)-1}}$. However $q^{\lambda(r)} \equiv 1 \pmod r$ and also $\alpha^r = 1$ which implies $\alpha^{q^{\lambda(r)}} = \alpha$. Therefore $\deg(g) \leqq \lambda(r)$.

a) By Lemma 10b) and the observation about the subfields of a Galois field we know that at least one root of each $x^r - p_i$, $1 \leqq i \leqq n$ lies in $GF(q^r)$. From part b) above we conclude that all rth roots of unity are in a $GF(q^s)$ with $s \leqq \lambda(r)$. Therefore all roots of $x^r - p_i$ and Ψ_r lie in $GF(q^{rs})$ and hence any sum of them does also. If $f_{r;p_1,\cdots,p_n} \pmod q$ or $f^*_{r;p_1,\cdots,p_n} \pmod q$ had an irreducible factor g of degree $> rs$ then one of its roots would generate $GF(q^{\deg(g)})$. But we know that this root lies in $GF(q^{rs})$. Therefore $\deg(g) \leqq rs \leqq r\lambda(r)$. □

4. Computational considerations. One may use Lemma 4 in connection with a method to compute cyclotomic polynomials [KNUD81, p. 440] to actually generate sample polynomials. In producing some of the following examples we used the computer algebra systems MACSYMA and SAC2.

TABLE 2

$n=0$

(1) $f^*_{8;\varnothing}(x)=\Psi_8(x)=x^4+1$, $\lambda(x)=2$[KNUD81, p. 437].
(2) $f^*_{12;\varnothing}(x)=\Psi_{12}(x)=x^4+x^2+1$, $\lambda(12)=2$ [VdWA53, p. 115].
(3) $f^*_{15;\varnothing}(x)=\Psi_{15}(x)=x^8-x^7+x^5-x^4+x^3-x+1$, $\lambda(15)=4$.

$n=1$

(4) $f^*_{3;2}(x)=x^6+3x^5+6x^4+3x^3+9x+9$.
(5) $f^*_{8;2}(x)=(x^{16}+4x^{12}-16x^{11}+80x^9+2x^8+160x^7+128x^6-160x^5+28x^4$
$-48x^3+128x^2-16x+1)(x^{16}+4x^{12}+16x^{11}-80x^9+2x^8-160x^7$
$+128x^6+160x^5+28x^4+48x^3+128x^2+16x+1)$.

$n=2$

(6) $f_{2;2,3}(x)=x^4-10x^2+1$ [GAAL73, p. 233].
(7) $f_{3;2,3}(x)=x^9-15x^6-87x^3-125$.
(8) $f_{4;2,3}(x)=x^{16}-20x^{12}+666x^8-3860x^4+1$.
(9) $f_{5;2,3}(x)=x^{25}-25x^{20}-3500x^{15}-57500x^{10}+21875x^5-3125$.
(10) $f^*_{3;2,3}(x)=x^{18}+9x^{17}+45x^{16}+126x^{15}+189x^{14}+27x^{13}-540x^{12}-1215x^{11}$
$+1377x^{10}+15444x^9+46899x^8+90153x^7+133893x^6+125388x^5$
$+29160x^4-32076x^3+26244x^2-8748x+2916$.

$n=3$

(11) $f_{2;2,3,5}(x)=x^8-40x^6+352x^4-960x^2+576$.
(12) $f_{2;-1,2,3}(x)=x^8-16x^6+88x^4+192x^2+144$ [KNUD81, p. 625].

Table 2 illustrates very well our results: All but polynomial (5) are irreducible over the integers. Since $\sqrt{2}\in Q(\zeta_8)$ we also know that (5) must be composite. Notice here that Lemma 7 cannot be used to show the irreducibility of (4), (9), (11) and (12). All the polynomials (1)–(12) factor in any modular field into polynomials of smaller degrees and make excellent test cases for implementations of the Berlekamp–Hensel factorization algorithm. That is, polynomial (10) factors

mod 7: $(x^3+x^2+4x+3)(x^3+2x^2+5x+5)(x^3+2x^2+4x+2)$
$(x^3+x^2+3x+5)(x^3+2x^2+2x+3)(x^3+x^2+x+2)$,

mod 17: $(x^2+12x+16)(x^2+16x+7)(x^2+9x+13)(x^2+9x+9)$
$(x^2+16x+16)(x^2+12x+9)(x^2+5x+7)(x^2+16x+1)(x+8)^2$,

mod 103: $(x^3+9x^2+27x+25)(x^3+62x^2+11x+28)(x^3+73x^2+94x+28)$
$(x^3+39x^2+95x+32)(x^3+92x^2+6x+25)(x^3+43x^2+67x+32)$,

mod 1979: $(x^2+1823x+1632)(x^2+85x+6)(x^2+828x+749)$
$(x^2+1069x+6)(x^2+1069x+749)(x^2+1069x+1632)$
$(x^2+1069x+878)(x^2+85x+1744)(x^2+828x+1744)$.

The variation of the maximum degree bound for different primes can be explained by the remark following the first proof of Theorem 2.

The Berlekamp–Hensel factorization algorithm contains the following "bottleneck" [KNUD81, p. 434]: If f is a polynomial of degree k and splits in a chosen residue field into j irreducible factors then one must perform at least $2^{j-1}-1$ trial divisions to prove its irreducibility over the integers. In the case of $f_{r;p_1,\cdots,p_n}k=.r^n$ and $j\geqq r^{n-1}/\lambda(r)$ and hence at least $2^{r^{n-1}/\lambda(r)-1}-1$ steps are executed. Fixing r gives an $O(2^k)$ lower timing bound for these inputs. We will show below that the lengths of

the coefficients are bounded by $O(k \log \log (k))$ and thus the worst case time complexity of the Berlekamp–Hensel algorithm is indeed an exponential function of the degree and coefficient lengths of its inputs. Since the degrees of all irreducible factors of $f_{r;p_1,\cdots,p_n}$ are independent of n the modifications of this algorithm suggested in [KNUD81, p. 434] do not eliminate the exponential running time behavior.

However, one additional possibility has arisen: The probabilistic version of Berlekamp's factorization algorithm over finite fields runs sufficiently fast for large modular fields [RABM80]. By avoiding the Hensel factor lifting step we now can choose a prime dividing the discriminant of the polynomial to be factored causing multiple factors to appear in the factorization of the image polynomial. Then it may happen (though this is quite unlikely) that there occur only a few distinct factors with high multiplicities. (Compare this observation with the proof of Eisenstein's criterion in [KNUD81, p. 626].)

Unfortunately, this is certainly not possible for the cyclotomic polynomials Ψ_m with m chosen as in Lemma 1: If $m = p_2 \cdots p_k$ (p_i being the ith consecutive prime) then $\Psi_m \pmod q$ is inseparable only if $q|m$. Therefore $q = p_i$ for some index i in $\{2, \cdots, k\}$. Generally such a prime q is too small to justify the omission of the Hensel lifting procedure. But even if we ignore this fact the number of trials still exceeds any polynomial time function of the input sizes: By Lemma 3, $\Psi_m \equiv (\Psi_{m/q})^q \pmod q$ and we know that $\Psi_{m/q} \pmod q$ is separable and hence has at least $\phi(m/q)/\lambda(m/q)$ distinct factors. Writing n for $\phi(m)$, Lemma 1 implies that almost all of the integers m yield $n/\lambda(m) > (\log n)^4$. Using the prime number distribution law (or Theorem 8 in [HAWR79, p. 10]), we know that $\phi(q) = q - 1$ is of order $O(k(\log k))$. From $\log_2 (n) > k - 2$ we conclude that $\phi(q)$ is of order $O((\log n)^2)$. Since $\phi(m/q)/\lambda(m/q) > (\log n)^4/\phi(q)$ we have at least $O((\log n)^2)$ distinct factors and thus need at least $O((q+1)^{(\log n)^2})$ trial divisions to prove Ψ_m irreducible. Of course this function is superpolynomial in $n = \phi(m) = \deg(\Psi_m)$.

Finally we establish certain bounds for the coefficients of our polynomials when the primes p_i are as small as possible. For a polynomial f, let norm (f) denote the sum of the absolute values of the coefficients of f.

THEOREM 3. *Let r be a fixed integer $\geqq 2$ and let $p_1, \cdots, p_n$ be the first n primes.*

a) $\log (\text{norm } (f)) = O(\deg (f) \log \log (\deg (f)))$ *for* $f = f_{r;p_1,\cdots,p_n}$ *and for* $f = f^*_{r;p_1,\cdots,p_n}$.

b) $\log_2 (\text{norm } (\Psi_m)) \leqq \phi(m)$ *for* $m \geqq 1$.

Proof. Given $f(x) = a_0 + a_1 x + \cdots + a_{k-1} x^{k-1} + x^k \in Z[x]$, let B denote the maximum of the absolute values of the roots of f. Then, since the coefficients are the elementary symmetric functions of the roots, it follows that $|a_i| \leqq \binom{k}{i} B^{k-i}$ for $0 \leqq i \leqq k$. Therefore norm $(f) \leqq (B+1)^k$.

a) For $f = f_{r;p_1,\cdots,p_n}$ the maximum absolute value of the roots is $B = (p_1)^{1/r} + \cdots + (p_n)^{1/r}$ and for $f^* = f^*_{r;p_1,\cdots,p_n}$ it is $B^* = 1 + B$. As noted above p_i is of order $O(i \log (i))$, so that B and B^* are of order $O(n^2)$. Since r is fixed n is of order $O(\log \deg (f))$ and $O(\log \deg (f^*))$. Taking the logarithm of the previous inequality for the norm immediately establishes part a).

b) Every root of Ψ_m has absolute value 1 and hence norm $(\Psi_m) \leqq (1+1)^{\phi(m)}$. □

5. Remarks. By way of summary, we attempt to abstract from our arguments about our class of polynomials an "intuitive" explanation of their unusual factorization behavior. Their irreducibility over the integers seems quite plausible because a set of roots of powers of primes is linearly independent over the field of rational numbers. However, their Galois groups have very short cycle lengths and hence a modular projection will preserve very little structure of these groups. Another approach exploits

an interesting property of the pure equation $x^r - m$ but we feel that these methods are dual to each other. Therefore it seems that the extreme "compression" of the Galois groups by taking the remainders is to be blamed for the abundance of factors we then get.

The average behavior of the Berlekamp-Hensel algorithm is quite well understood [COLG79]. The density theorems used for its analysis can be also applied to our polynomials and based on them, P. Weinberger has devised an algorithm that, provided the generalized Riemann hypothesis holds, establishes irreducibility in polynomial time (cf. [KNUD81, p. 632]). More recently, A Lenstra, H. Lenstra and L. Lovász [LELL82] have replaced the exponential part of the Berlekamp-Hensel algorithm by a polynomial time algorithm for computing a short vector in an integral lattice. We hope that our polynomials make an interesting test case for this new algorithm.

REFERENCES

[BERL70] E. R. BERLEKAMP, *Factoring polynomials over large finite fields*, Math. Comput., 24 (1970), pp. 713–735.

[BESI40] A. S. BESICOVITCH, *On the linear independence of fractional powers of integers*, J. London Math. Soc., 15 (1940), pp. 1–3.

[CAVB68] B. F. CAVINESS, *On canonical forms and simplification*, Ph.D. thesis, Carnegie-Mellon Univ., Pittsburgh, 1968.

[COLG79] G. E. COLLINS, *Factoring univariate integral polynomials in polynomial average time*, Proc. EUROSAM '79, Springer-Verlag, New York, 1979.

[GAAL73] L. GAAL, *Classical Galois Theory with Examples*, Chelsea, New York, 1973.

[HAWR79] G. H. HARDY AND E. M. WRIGHT, *An Introduction to the Theory of Numbers*, Fifth ed., Oxford Univ. Press, Oxford, 1979.

[KNUD81] D. E. KNUTH, *The Art of Computer Programming*, Vol. 2, 2nd ed., Addison-Wesley, Reading, MA, 1981.

[KOEN1895] L. KÖNIGSBERGER, *Grundzüge einer arithmetischen Theorie der algebraischen Grőszen*, J. Reine Angew. Math., 92 (1895), pp. 53–78.

[LELL82] A. K. LENSTRA, H. W. LENSTRA AND L. LOVAŚZ, *Factoring polynomials with rational coefficients*, Report 82-05, Mathematisch Instituut, Amsterdam, 1982.

[LOOS82] R. LOOS, *Computing in algebraic extensions*, Computing, Supplement 4 (1982).

[MUSD75] D. R. MUSSER. *Multivariate polynomial factorization*, J. Assoc. Comput. Mach., 22 (1975), pp. 291–308.

[NARW74] N. NARKIEWICZ, *Elementary and Analytic Theory of Algebraic Numbers*, Polish Sci. Publ., Warszawa 1974. (In Polish.)

[RABM80] M. O. RABIN, *Probabilistic algorithms in finite fields*, this Journal, 9 (1980), pp. 273–280.

[RICI74] I. RICHARDS, *An application of Galois theory to elementary arithmetic*, Adv. in Math., 13 (1974), pp. 268–273.

[SIEW64] W. SIERPINSKI, *Elementary Theory of Numbers*, Polish Sci. Publ., Warszawa, 1964. (In Polish).

[VdWA53] B. L. VAN DER WAERDEN, *Modern Algebra*, Vol. 1, Ungar, New York, 1953.

[WERO76] P. J. WEINBERGER AND L. P. ROTHSCHILD, *Factoring polynomials over algebraic number fields*, ACM Trans. Math. Software, 2 (1976), pp. 335–350.

[ZASH79] H. ZASSENHAUS, *On the Van der Waerden criterion for the group of an equation*, Proc. EUROSAM '79, Springer-Verlag, New York, 1979.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0006 $01.25/0

MULTIDIMENSIONAL SORTING*

JACOB E. GOODMAN† AND RICHARD POLLACK‡

Abstract. We introduce a process called *geometric sorting*, which can be applied to an arbitrary configuration of points in d-dimensional space, and which encodes in compact form the order properties of the configuration, just as the arrangement of a set of numbers in size place encodes its order properties. We give an algorithm for carrying out this sorting procedure in time $O(n^d \log n)$, which generalizes the optimum sorting time of $O(n \log n)$ for the linear case: In addition, we give an efficient algorithm for determining whether two randomly numbered configurations in $\mathbb{R}^d$ have the same order type, using a distinguished family of orderings of each. Finally, we indicate how this new concept of sorting can be applied to problems in pattern recognition, stereochemistry, and cluster analysis.

Key words. geometric sorting, computational geometry, efficient algorithms, planar configuration of points, multidimensional configuration, orientation, convex hull

Introduction. Finding the vertices of the convex hull of a given set of points in the plane, a problem that has been written about extensively in the last ten years [14], [19], [20], generalizes in a natural way the problem of determining the maximum and minimum of a given set of numbers on the real line. But there is much more to the "order type" of an indexed set $\{x_1, \cdots, x_n\}$ of numbers than just their maximum and minimum; this is why finding their extremes takes just $O(n)$ comparisons, while sorting them—by an optimal worst-case method such as a heapsort [18]—takes $O(n \log n)$ comparisons. In the same way, there is much more to a set of points in the plane—or in d-space, for that matter—than their convex hull. It is this more general problem of determing the "order type" of a given configuration of points in d-dimensional space, and determining it by a fast, efficient algorithm, to which this paper addresses itself.

If n numbers, $x_1, \cdots, x_n$, are given, there is just one kind of order pattern that they can have: that of the integers $1, 2, \cdots, r$ for some $r \leqq n$. The problem of sorting them comes down to finding a mapping π of the set $\{1, \cdots, n\}$ onto a set $\{1, \cdots, r\}$ that is "order-preserving", in the sense that $x_i \leqq x_j$ if and only if $\pi(i) \leqq \pi(j)$. But in the plane, or in higher dimensions, as we shall see, a configuration of n points can have many "patterns" or "order types". (It is a difficult problem, in fact, to find just how many, but we shall give some bounds below.) To sort such a configuration, we must therefore do *two* things: 1) determine which of the standard order patterns our configuration $\mathscr{C}$ fits, and 2) find a mapping from $\mathscr{C}$ into that standard pattern which exhibits the order isomorphism. We shall describe a means of carrying out each of these steps below.

First, what do we mean by the "order type" of a configuration of points? If n distinct points, $P_1, \cdots, P_n$, are given on a line, sorting them can be thought of in two ways: finding

(a) the set $\Lambda(i)$ of points on the positive side (i.e., to the right) of each P_i, since then the fundamental atoms of the ordering of $P_1, \cdots, P_n$, i.e. the ordered pairs i, j for which $P_i < P_j$, are immediately accessible, or

* Received by the editors September 30, 1981, and in revised form June 30, 1982.

† Department of Mathematics, City College of City University of New York, New York 10031. The work of this author was supported in part by the National Science Foundation under grant MCS 82-01831.

‡ Courant Institute of Mathematical Sciences, New York University, New York 10012. The work of this author was supported in part by the National Science Foundation under grant MCS 82-01342.

(b) the *number* $\lambda(i)$ of points on the positive side (i.e., to the right) of each P_i, since then the position of each point P_i in the ranking $P_{i_1} < \cdots < P_{i_n}$ is immediately deducible.

Furthermore, it is clear that from (b) we can read off (a), and of course (a) immediately gives (b). In the plane, indeed in d-dimensional space, each of these has a counterpart: the fundamental atoms of the "order pattern" made up of a planar configuration of points, what Dreiding and Wirth [6] call *chirons*, are the ordered triples i, j, k for which P_i, P_j, P_k are positively (i.e. "counterclockwise") oriented; hence to know these it would, as above, suffice to find

(a) the set $\Lambda(i, j)$ of points P_k lying on the positive side (i.e., to the left) of each directed line P_iP_j.

Similarly, the counterpart of (b) above would be to find

(b) the *number* $\lambda(i, j)$ of points P_k lying on the positive side (i.e., to the left) of each directed line P_iP_j.

It is a remarkable fact that—just as on the line—(a) and (b) are equivalent, in the sense that knowledge of one implies knowledge of the other. This result, which is not immediately obvious even in the planar case, was first shown, in that case, in [12], by the use of the method of "allowable sequences of permutations". Here we show by a direct geometric argument that it holds in every dimension, and even in the case where we allow degeneracies in our configurations: repeated points, collinear triples, coplanar quadruples, etc.; this is the substance of Theorem 1.8 below, which we call the "basic theorem of geometric sorting".

In §§ 2 and 3 we present a fast algorithm for sorting, i.e. determining the λ-function of, a configuration $\mathscr{C}$ of n points in $\mathbb{R}^d$; the d-dimensional algorithm in § 3 is based on the 2-dimensional one in § 2.

In §§ 4 and 5 we address the comparison problem: How do we determine whether two configurations of n points can be renumbered so that they are "λ-equivalent", i.e. have the same order type? Surprisingly enough, it turns out that we do not need to check all $n!$ renumberings, but at most n, in dimension 2, and $6n$-12, in dimension 3. In higher dimensions this number grows, but it is always bounded by $cn^{\lfloor d/2 \rfloor}$. In § 4 we introduce a distinguished family of orderings of the points of a configuration $\mathscr{C}$ in $\mathbb{R}^d$, which we call the "canonical orderings of $\mathscr{C}$", and which we believe to be of interest in their own right, in order to generate all the necessary renumberings. We present algorithms to handle various cases which may arise; in § 5 we present an algorithm for carrying out the entire comparison process.

Finally, in § 6 we discuss the optimality of the sorting algorithm, and we touch on several potential applications of geometric sorting: to pattern recognition, to stereochemistry, and to cluster analysis. We end with some open problems.

It should perhaps be pointed out that the notion of what we call the "order type" of a configuration $\mathscr{C}$ of points can be seen to be equivalent to what is known as the oriented matroid structure determined by $\mathscr{C}$ (see [2], [4], [5], [8]). However, its dual expression, in terms of Λ and λ, which forms the basis of this paper, is new here, as are the applications that follow to efficient methods of computing and encoding the order type of a configuration, and to the comparison problem for configurations.

1. The basic theorem of geometric sorting.

DEFINITION 1.1. If $(P_0, \cdots, P_d)$ is a sequence of $d+1$ points in $\mathbb{R}^d$, with $P_i = (x_{i1}, \cdots, x_{id})$ for each i, we say they have *positive orientation*, written $P_0 \cdots P_d > 0$, if

$$\det(x_{ij}) > 0,$$

where $x_{i0} = 1$ for each i. $P_0 \cdots P_d < 0$ and $P_0 \cdots P_d = 0$ are similarly defined.

Remark 1.2. If $d=1$, this reduces to: $P_0P_1>0$ whenever P_1 is to the "right" of P_0. If $d=2$, it says that $P_0P_1P_2>0$ whenever $P_0P_1P_2$ is oriented "counterclockwise", i.e., when P_2 is to the left of the directed line P_0P_1. And if $d=3$, it says that $P_0P_1P_2P_3>0$ whenever a "right-handed screw" turning in the direction $P_0P_1P_2$ moves toward P_3 (assuming the x, y, z-axes form a right-handed system!).

DEFINITION 1.3. If $\mathscr{C}=\{P_1,\cdots,P_n\}$ is a numbered configuration in $\mathbb{R}^d$ (with possible repetitions among the points P_i), and if $P_{i_0},\cdots,P_{i_{d-1}}$ affinely span a hyperplane of $\mathbb{R}^d$, let

$$\Lambda(i_0,\cdots,i_{d-1})=\{j\,|P_{i_0}\cdots P_{i_{d-1}}P_j>0\},$$

and let

$$\lambda(i_0,\cdots,i_{d-1})=|\Lambda(i_0,\cdots,i_{d-1})|;$$

if $P_{i_0},\cdots,P_{i_{d-1}}$ are affinely dependent, we write

$$\Lambda(i_0,\cdots,i_{d-1})=\Omega$$

and

$$\lambda(i_0,\cdots,i_{d-1})=\omega.$$

We can now define the "order type" of a numbered configuration $\mathscr{C}$ in $\mathbb{R}^d$ in two different ways, according to either the *number* $\lambda(i_0,\cdots,i_{d-1})$ of points on the positive side of each hyperplane $\langle P_{i_0},\cdots,P_{i_{d-1}}\rangle$, or the *set* $\Lambda(i_0,\cdots,i_{d-1})$ of these points. It is clear that if two configurations, $\mathscr{C}$ and $\mathscr{C}'$, are Λ-equivalent, in this sense, then they are λ-equivalent; Theorem 1.8 below shows that—just as on the line—the converse is true, i.e., knowing how *many* points lie on the positive side of each hyperplane $\langle P_{i_0},\cdots,P_{i_{d-1}}\rangle$ determines *which* ones do. First we need a few lemmas.

LEMMA 1.4. *If $P_1,\cdots,P_n$ affinely span $\mathbb{R}^d$, then $P_{i_0}=P_{i_1}$ if and only if $\lambda(i_0,i_1,j_2,\cdots,j_{d-1})=\omega$ for every choice of $j_2,\cdots,j_{d-1}$.*

Proof. The necessity of the condition is clear, and the sufficiency follows from the fact that the set consisting of any two distinct points can be completed to an affine spanning set of $d+1$ points.

LEMMA 1.5. *If $P_0,\cdots,P_d$ affinely span $\mathbb{R}^d$ and $\lambda(0,\cdots,d-1)=0$, then $\lambda(0,\cdots,j-1,d,j+1,\cdots,d-1)>0$ for each j, $0\leqq j\leqq d-1$.*

Proof. The hypothesis implies that $P_0\cdots P_d<0$. Hence $P_0\cdots P_{j-1}P_dP_{j+1}\cdots P_{d-1}P_j>0$, so that

$$j\in\Lambda(0,\cdots,j-1,d,j+1,\cdots,d-1).$$

LEMMA 1.6. *If $P_0\cdots P_d<0$ and if P' is any point lying in the hyperplane $\langle P_0,\cdots,P_{d-1}\rangle$, then $P_0,\cdots P_{j-1}P'P_{j+1}\cdots P_d<0$ for some j, $0\leqq j\leqq d-1$.*

Proof. Let $P_k=(x_{k1},\cdots,x_{kd})$ for each k. When we write

$$\overrightarrow{OP'}=\sum_{0\leqq k\leqq d-1}c_k\overrightarrow{OP_k},\qquad\text{with}\sum_k c_k=1,$$

we must have $c_j>0$ for some j, $0\leq j\leq d-1$. The result then follows by splitting the

determinant

$$\begin{vmatrix} 1 & x_{01} & \cdots & x_{0d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{j-1,1} & \cdots & x_{j-1,d} \\ \sum c_k & \sum c_k x_{k1} & \cdots & \sum c_k x_{kd} \\ 1 & x_{j+1,1} & \cdots & x_{j+1,d} \\ \vdots & \vdots & & \vdots \\ 1 & x_{d1} & \cdots & x_{dd} \end{vmatrix}$$

into a sum of determinants along its jth row by multilinearity, and noting that all the summands give 0 except for the jth, which gives

$$c_j \begin{vmatrix} 1 & x_{01} & \cdots & x_{0d} \\ & & \cdots & \\ 1 & x_{d1} & \cdots & x_{dd} \end{vmatrix},$$

and this is negative.

LEMMA 1.7. *Let H be a hyperplane in $\mathbb{R}^d$, let P_0 be any point not in H, and let H_0 be the hyperplane parallel to H and passing through P_0. Finally, let U be the halfspace of $\mathbb{R}^d$ that is on the same side of H_0 as H, i.e., the component of $\mathbb{R}^d \backslash H_0$ containing H, and let $\pi : U \to H$ denote projection from P_0. Then one of the two orientations of H as a $(d-1)$-dimensional affine space in its own right will agree with the standard orientation of $\mathbb{R}^d$ in the following sense: if $P_1, P_2, \cdots, P_d$ are points of U, then $P_0P_1 \cdots P_d$ will be positively oriented, negatively oriented, or affinely dependent in $\mathbb{R}^d$ if and only if $\pi(P_1)\pi(P_2) \cdots \pi(P_d)$ has the corresponding property in H.*

Proof. Suppose $P_0P_1 \cdots P_d > 0$ for some choice of $P_1, \cdots, P_d$ in U. H has only two orientations. Let us fix one in which $\pi(P_1) \cdots \pi(P_d) > 0$ also. Then if $Q_1, \cdots, Q_d$ are any other points in U for which $P_0Q_1 \cdots Q_d > 0$, we can move the simplex $P_0, P_1, \cdots, P_d$ continuously to $P_0, Q_1, \cdots, Q_d$ (with P_0 remaining fixed and the other vertices remaining in U) so that each intermediate configuration also has positive orientation. Since $\pi(R_1), \cdots, \pi(R_d)$ are affinely dependent in H if and only if $P_0, R_1, \cdots, R_d$ are affinely dependent in $\mathbb{R}^d$, it follows by the intermediate value theorem that we must have $\pi(Q_1) \cdots \pi(Q_d) > 0$ as well, from which the assertion follows immediately.

THEOREM 1.8. *If, for a configuration $\mathscr{C} = \{P_1, \cdots, P_n\}$ in $\mathbb{R}^d$, the function λ is given, then the function Λ is uniquely determined.*

Proof. The proof is by double induction: by induction on d and, for a given d, by induction on n. Let us first note that the statement is clear for $d = 1$, as is immediate from Remark 1.2. Suppose, then, that $d > 1$. We may assume $\lambda(i_0, \cdots, i_{d-1}) > 0$ for some $i_0, \cdots, i_{d-1}$; otherwise the statement is trivially true. Now let us fix $i_0, \cdots, i_{d-1}$ such that $\lambda(i_0, \cdots, i_{d-1}) = 0$; such a d-tuple must exist, since the convex hull Δ of $\{P_1, \cdots, P_n\}$ has faces that span hyperplanes, and—if necessary, by interchanging two of the indices—we may select a spanning set for such a hyperplane that has no points of $\mathscr{C}$ on its "positive" side. Let F be the face of Δ on which the points $P_{i_0}, \cdots, P_{i_{d-1}}$ lie. If k is any index such that $\lambda(i_0, \cdots, i_{j-1}, k, i_{j+1}, \cdots, i_{d-1}) = 0$ for some j, $0 \leqq j \leqq d-1$, then the points $P_{i_0}, \cdots, P_{i_{j-1}}, P_k, P_{i_{j+1}}, \cdots, P_{i_{d-1}}$ also lie on a face F' of Δ, and we must have $F' = F$: otherwise the result of Lemma 1.5 would be violated; hence $P_k \in F$. Conversely, if P_k is any point on F we must have $\lambda(i_0, \cdots, i_{j-1}, k, i_{j+1}, \cdots, i_{d-1}) = 0$ for some j; this follows from Lemma 1.6 since the points of $\mathscr{C}$ span $\mathbb{R}^d$. Thus λ determines all of the points lying on each face of Δ. Now

by Lemma 1.4, λ determines the equivalence classes of the relation

$$i \sim j \Leftrightarrow P_i = P_j;$$

we can therefore coalesce each "cluster" of repeated points into a single point, and get a configuration $\bar{\mathscr{C}}$ consisting of distinct points $\bar{P}_1, \cdots, \bar{P}_r$, with $r \leqq n$. In particular, we know the faces of the convex hull $\bar{\Delta}$ of $\bar{\mathscr{C}}$—they are just the images of the faces of Δ. Hence, since an extreme point of $\bar{\Delta}$ is an intersection of faces that has cardinality 1, we can identify the extreme points of $\bar{\Delta}$, and so also the extreme points of Δ. Say P_n is such an extreme point, and say $\{P_{m+1}, \cdots, P_n\}$ is the cluster of repeated points to which P_n belongs. Let H_0 be a supporting hyperplane of Δ at P_n, and let U be the open halfspace determined by H_0 which contains the points $P_1, \cdots, P_m$. Let H be a hyperplane parallel to H_0 and lying in U, with the orientation given by Lemma 1.7; as in that lemma, let $\pi : U \to H$ be projection from P_n. Then by that lemma, if we let Λn (resp. λ_n) be the Λ- (resp. λ-) function for the configuration $\mathscr{C}_n = \{\pi(P_1), \cdots, \pi(P_m)\}$, we have

$$\lambda_n(\pi(P_{i_1}), \cdots, \pi(P_{i_{d-1}})) = \lambda(P_n, P_{i_1}, \cdots, P_{i_{d-1}}),$$

for any choice of indices. This shows that the function λ_n is determined, so—by induction on the dimension—Λ_n is also. But this means, again by Lemma 1.7, that we can determine the orientation of every $d+1$-tuple of $\mathscr{C}$ involving P_n, i.e.,

$$\text{(1.1)} \qquad \begin{aligned} &\textit{we can identify those } d\textit{-tuples } P_{i_0}, \cdots, P_{i_{d-1}} \in \mathscr{C} \textit{ for which} \\ &P_{i_0} \cdots P_{i_{d-1}} P_n > 0. \end{aligned}$$

Let $\tilde{\mathscr{C}}$ be the configuration $\{P_1, \cdots, P_m\}$ that remains when the entire cluster of repeated points to which P_n belongs has been deleted; then we have just shown that the λ-function $\tilde{\lambda}$ of $\tilde{\mathscr{C}}$ is determined: it is given by

$$\tilde{\lambda}(i_0, \cdots, i_{d-1}) = \lambda(i_0, \cdots, i_{d-1}) \qquad (\text{resp. } \lambda(i_0, \cdots, i_{d-1}) - (n-m))$$
$$\text{if } P_{i_0} \cdots P_{i_{d-1}} P_n \leqq 0 \qquad (\text{resp.} > 0).$$

Hence by induction on the number of points of the configuration, the corresponding function $\tilde{\Lambda}$ is also determined, and therefore—again with the help of (1.1)—so is Λ, which proves the theorem.

COROLLARY 1.9. *If two configurations have the same λ-function, they have the same order type, i.e., the same orientation of $(d+1)$-tuples $P_{i_0}, \cdots, P_{i_d}$ for every $i_0, \cdots, i_d$.*

We can therefore formalize the notion of order type by stating

DEFINITION 1.10. If $\mathscr{C} = \{P_1, \cdots, P_n\}$ and $\mathscr{C}' = \{P'_1, \cdots, P'_n\}$ are numbered configurations in $\mathbb{R}^d$ for which $\lambda_{\mathscr{C}}(i_0, \cdots, i_{d-1}) = \lambda_{\mathscr{C}'}(i_0, \cdots, i_{d-1})$ for every $i_0, \cdots, i_{d-1}$, we say $\mathscr{C}$ and $\mathscr{C}'$ have the same order type, or $\mathscr{C}$ is equivalent to $\mathscr{C}'$ ($\mathscr{C} \sim \mathscr{C}'$).

By analogy with the interpretation [18, p. 4] of "sorting" a set of numbers as meaning determining the position of each when they are arranged in increasing order, we then have:

DEFINITION 1.11. If points $P_1, \cdots, P_n$ in $\mathbb{R}^d$ are given, to *sort* them *geometrically* will mean to determine the associated λ-function.

Remark 1.12. If $P_1, \cdots, P_n$ affinely span $\mathbb{R}^d$, the sorting function λ determines not only the "d-dimensional order type" of any subset, but also the linear order of a collinear subset, the "2-dimensional order type" of a coplanar subset, and so on—just fill out the remaining places with points chosen in general position, and use the result of Theorem 1.8. If $P_1, \cdots, P_n$ do *not* span $\mathbb{R}^d$, however, then sorting them in the sense above yields very little information on their actual (lower-dimensional) order

type, since the value of λ is either 0 or ω for each d-tuple. But in this case, by throwing in the points $(1, 0, \cdots, 0), (0, 1, 0, \cdots, 0), \cdots, (0, \cdots, 0, 1)$ and *then* sorting, we can get all the information about the order type of the original configuration. From now on, whenever we speak of sorting a configuration $\mathscr{C}$ in $\mathbb{R}^d$, we shall therefore assume that $\mathbb{R}^d$ is spanned by the points of $\mathscr{C}$.

2. An algorithm for sorting in the plane. Here is a fast algorithm for determining the λ-matrix of a planar configuration. The idea is quite simple: consider each point P_i, in turn, as an "origin", reflect each $P_j \neq P_i$ in P_i, obtaining new points "P_{n+j}", and sort the points and their reflections circularly about P_i, the order within each ray about P_i being immaterial. In order to obtain $\lambda(i, j)$, we then have merely to count the number of original points from the ray P_iP_j around to the ray P_iP_{n+j}.

ALGORITHM 2.1.

Input: $(P_i = (x_i, y_i))$, $1 \leqq i \leqq n$.

Output: $(\lambda(i, j))$, $1 \leqq i \leqq n$, $1 \leqq j \leqq n$.

1. (Starting with $i = 1$) pick the next i (ending with $i = n$).
2. For every $j = 1, \cdots, n$, let $u_i = x_j - x_i$, $v_j = y_j - y_i$.
3. For every $j = 1, \cdots, n$, if $(u_j, v_j) = (0, 0)$, let $\lambda(i, j) = \omega$; otherwise, call j "good".
4. For every good $j = 1, \cdots, n$, let $u_{n+j} = -u_j$, $v_{n+j} = -v_j$, and let $m_j = m_{n+j} = v_j/u_j$ if $u_j \neq 0$.
5. Sort the indices $\{j \mid j \text{ is good}\} \cup \{n + j \mid j \text{ is good}\}$ into subsets, as follows:
   (a) first those for which $u_j > 0$, using m_j as key;
   (b) next those for which $u_j = 0$ and $v_j > 0$;
   (c) next those for which $u_j < 0$, using m_j as key;
   (d) finally those for which $u_j = 0$ and $v_j < 0$.

(In (a) and (c) we get a list of subsets; the order within each subset is irrelevant.) Say the result is

$$J_{11}, \cdots, J_{1s_1}, \cdots, J_{r1}, \cdots, J_{rs_r},$$

where the points with indices $J_{k1}, \cdots, J_{ks_k}$ constitute an entire subset, and there are r subsets all together.

6. For each $k = 1, \cdots, r$, let

$$n_k = |\{m \mid 1 \leqq m \leqq s_k, J_{km} \leqq n\}|.$$

7. For each good $j = 1, \cdots, n$, let

$$\lambda(i, j) = \begin{cases} \displaystyle\sum_{k=k(j)+1}^{k(j+n)-1} n_k & \text{if } k(j+n) > k(j), \\ \displaystyle\sum_{k=k(j)+1}^{r} n_k + \sum_{k=1}^{k(j+n)-1} n_k & \text{if } k(j+n) < k(j). \end{cases}$$

(Note: $k(j)$ means, of course, the number of the subset within which j lies.)

8. Return to step 1.

Analysis. Step 2 translates each point P_i, in turn, to the origin O. Step 3 calls λ "undefined" if $P_j = P_i$, and designates P_j as "good" otherwise. Step 4 reflects each good P_j in P_i, and calculates the slope of P_iP_j. Step 5 sorts the good points and their reflections into rays starting at O, listed in counterclockwise order, the order along each ray being immaterial (see Fig. 1). Step 6 counts the *original* points in each ray. Finally, step 7 counts the number of original points going counterclockwise from the ray containing P_j to the ray containing its reflection: this gives $\lambda(i, j)$.

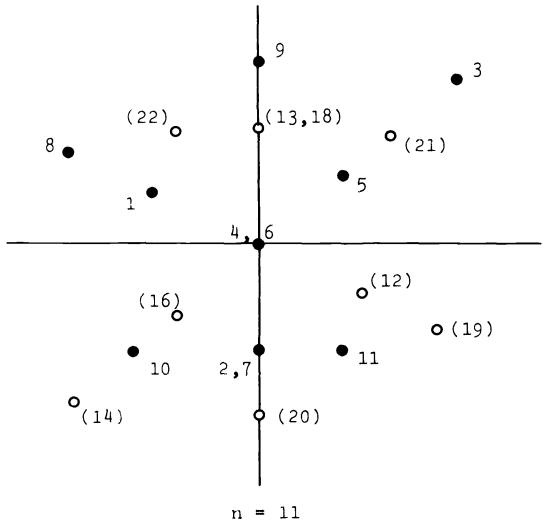


FIG. 1. $n = 11; J_{11}, \cdots, J_{83} = 11$; 12, 19; 3, 5, 21; 9, 13, 18; 22; 1, 8; 10, 14, 16; 2, 7, 20.

For each $i = 1, \cdots, n$, steps 2, 3, 4, and 6 take time $O(n)$, step 7 also takes time $O(n)$ if the sum for each j is computed by correcting the sum for the previous j, and the sorting in step 5 takes $O(n \log n)$ if done by an optimal worst-case method, such as heapsort [18]; hence for each i the time is $O(n \log n)$. It follows that the total sorting time is $O(n^2 \log n)$.

3. An algorithm for sorting in higher dimensions. The process of geometric sorting is not quite so simple in higher dimensions as in the plane, if one wants to obtain an efficient algorithm. For example, a straightforward approach would be to project the configuration $\mathcal{C}$ parallel to each hyperplane $\langle P_{1_i}, \cdots, P_{i_d} \rangle$ onto a coordinate axis transversal to that hyperplane, and simply count the number of points on each side of the projection of the hyperplane itself. This, however, would take time $O(n^{d+1})$ for fixed d, and this is precisely what we are trying to avoid, since the λ-function for a configuration in $\mathbb{R}^d$ occupies only $O(n^d)$ space.

It turns out that a combination of this projection method (but onto a *plane* instead of a line), together with the sorting-by-slope procedure of Algorithm 2.1, reduces the time to $O(n^d \log n)$. Here is the basic idea (the details, which are a bit complicated, are presented in the Analysis section immediately following the algorithm): For each choice of $d-1$ points in general position, we choose a coordinate plane Π transversal to their span Σ, and project the entire configuration parallel to Σ onto Π. We then sort the projected points and their reflections circularly around the projection of Σ, precisely as in Algorithm 2.1, and the same device used there, of counting around from a ray through a point to the ray through its reflection, then gives the value of λ. A few "tricks" are used in the algorithm to shorten the execution time. These include (a) a dilatation of the projected configuration around the "origin", which comes from suppressing the denominators in the application of Cramer's rule used to find the projected configuration, and which has no effect on the order type of the projection, and (b) finding $\lambda(i_1, \cdots, i_d)$ only for $i_1 < \cdots < i_{d-2} < \{i_{d-1}, i_d\}$ and then applying even permutations to the indices to find the remaining values of λ; these are explained in more detail in the Analysis section below.

ALGORITHM 3.1′.

Input: $(P_i = (x_{i1}, \cdots, x_{id}))$, $1 \leqq i \leqq n$.

Output: $(\lambda(i_1, \cdots, i_d))$, $1 \leqq i_1, \cdots, i_d \leqq n$.

1. Let $\lambda(i_1, \cdots, i_d) = \omega$ for every $i_1, \cdots, i_d$, $1 \leqq i_j \leqq n$.

2. (Starting with $i_1, \cdots, i_{d-1} = 1, \cdots, d-1$) pick the next $i_1, \cdots, i_{d-1}$ with $i_1 < \cdots < i_{d-1}$ (ending with $n-d+2, \cdots, n$).

3. If

$$\det \begin{pmatrix} 1 & x_{i_1 1} & \cdots & x_{i_1 j-1} & x_{i_1 j+1} & \cdots & x_{i_1 k-1} & x_{i_1 k+1} & \cdots & x_{i_1 d} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{i_{d-1} 1} & \cdots & x_{i_{d-1} j-1} & x_{i_{d-1} j+1} & \cdots & x_{i_{d-1} k-1} & x_{i_{d-1} k+1} & \cdots & x_{i_{d-1} d} \end{pmatrix}$$

vanishes for every j, k with $1 \leqq j < k \leqq d$, go to step 2.
Otherwise, let (α, β) be the first (j, k) for which this determinant $\neq 0$, and let $\sigma = \pm 1$ be the sign of this determinant.

4. For each $j = 1, \cdots, n$, let

$$y_\alpha(j) = \det \begin{pmatrix} 1 & x_{i_1 1} & \cdots & x_{i_1, \beta-1} & x_{i_1, \beta+1} & \cdots & x_{i_1 d} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{i_{d-1} 1} & \cdots & x_{i_{d-1}, \beta-1} & x_{i_{d-1}, \beta+1} & \cdots & x_{i_{d-1} d} \\ 0 & x_{j1} & \cdots & x_{j, \beta-1} & x_{j, \beta+1} & \cdots & x_{jd} \end{pmatrix},$$

$$y_\beta(j) = \det \begin{pmatrix} 1 & x_{i_1 1} & \cdots & x_{i_1, \alpha-1} & x_{i_1, \alpha+1} & \cdots & x_{i_1 d} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{i_{d-1} 1} & \cdots & x_{i_{d-1}, \alpha-1} & x_{i_{d-1}, \alpha+1} & \cdots & x_{i_{d-1} d} \\ 0 & x_{j1} & \cdots & x_{j, \alpha-1} & x_{j, \alpha+1} & \cdots & x_{jd} \end{pmatrix}.$$

5. If $\sigma = +1$, let $u'(j) = y_\alpha(j)$ and $v'(j) = y_\beta(j)$ for each $j = 1, \cdots, n$; if $\sigma = -1$, let $u'(j) = y_\beta(j)$ and $v'(j) = y_\alpha(j)$ for each $j = 1, \cdots, n$.

6. For each $j = 1, \cdots, n$, let $u_j = u'(j) - u'(i_1)$ and $v_j = v'(j) - v'(i_1)$.

Now perform steps 3 through 7 of Algorithm 2.1, modified as follows:

7. For each $j = 1, \cdots, n$, if $(u_j, v_j) \neq (0, 0)$, call j "good".

8. Same as step 4, but define u_{n+j}, v_{n+j}, and m_{n+j} only for good $j = i_{d-2}+1, \cdots, n$; define m_j for *all* good $j = 1, \cdots, n$.

9. Same as step 5, but only for the indices $\{j \mid j \text{ is good}\} \cup \{n+j \mid j \text{ is good and } j \geqq i_{d-2}+1\}$.

10. Same as step 6.

11. Same as step 7, but only for good $j = i_{d-2}+1, \cdots, n$; call the result $\lambda(i_1, \cdots, i_{d-1}, j)$.

12. For each good $j = i_{d-1}+1, \cdots, n$, apply every even permutation π, in turn, to $i_1, \cdots, i_{d-1}, j$; let

$$\lambda(\pi(i_1), \cdots, \pi(i_{d-1}), \pi(j)) = \lambda(i_1, \cdots, i_{d-1}, j).$$

13. Return to step 2.

Analysis. Algorithm 3.1 computes $\lambda(i_1, \cdots, i_d)$ whenever $i_1 < \cdots < i_{d-2} < \{i_{d-1}, i_d\}$, provided the points $P_{i_1}, \cdots, P_{i_{d-1}}$ are in general position, and then applies even permutations to these ordered d-tuples of indices and records the same values for λ. In this way *all* ordered d-tuples for which λ is defined have λ computed for them. In all other cases, where λ is undefined, we write $\lambda = \omega$ to represent this. It is therefore easier to write $\lambda = \omega$ everywhere to begin with, and write over this whenever the need arises; hence step 1. The determinant (call it Δ) computed in step 3 serves

three purposes: (a) it determines whether the points $P_{i_1}, \cdots, P_{i_{d-1}}$ span a $(d-2)$-flat, (b) it identifies a coordinate plane (the x_α, x_β-plane) transversal to that $(d-2)$-flat, and (c) its sign reflects the orientation of the ordered $(d-1)$-tuple $P_{i_1}, \cdots, P_{i_{d-1}}$. Step 4 comes from an application of Cramer's rule, used to find the projection $f(P_j)$ of each point P_j parallel to the $(d-2)$-flat $\langle P_{i_1}, \cdots, P_{i_{d-1}}\rangle$ into the x_α, x_β-plane. We have suppressed the denominators that Cramer's rule gives; doing so amounts to a dilatation around the origin, and possibly to a reflection through the origin as well, if the suppressed denominator is negative. This simplifies the calculation, and neither the dilatation nor the reflection affects the collinearity or the orientation of points in the x_α, x_β-plane. Step 5 compensates for the orientation of the points $P_{i_1}, \cdots, P_{i_{d-1}}$; the result of steps 4 and 5 is that the points $P_{i_1}, \cdots, P_{i_{d-1}}, P_j, P_k$ have positive orientation in $\mathbb{R}^d$ if and only if $f(P_{i_1}), f(P_j), f(P_k)$ have positive orientation in the u, v-plane (with the coordinates in their natural order in both cases). This is most easily seen as follows: Let us first check it in the special case of the points $P_0(0, \cdots, 0)$, $P_1(1, 0, \cdots, 0), \cdots, P_d(0, \cdots, 0, 1)$. Choose any α, β with $1 \leqq \alpha < \beta \leqq d$, and let π be any permutation of $\{0, \cdots, d\}$ such that $\pi(d-1) = \alpha$, $\pi(d) = \beta$. Let

$$\bar{\Delta} = \det \begin{pmatrix} 1 & x_{\pi(0),1} & \cdots & x_{\pi(0),d} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & x_{\pi(d),1} & \cdots & x_{\pi(d),d} \end{pmatrix};$$

we have $\bar{\Delta} = \operatorname{sgn}(\pi)$, of course. On the other hand, the $(d-2)$-flat along which we are going to project to the x_α, x_β-plane is $\langle P_{\pi(0)}, \cdots, P_{\pi(d-2)}\rangle$, so it is precisely the αth and βth columns we must leave out to get a nonzero determinant; hence

$$\Delta = \det \begin{pmatrix} 1 & x_{\pi(0)1} & \cdots & x_{\pi(0),\alpha-1} & x_{\pi(0),\alpha+1} & \cdots & x_{\pi(0),\beta-1} & x_{\pi(0),\beta+1} & \cdots & x_{\pi(0)d} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{\pi(d-2)1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & x_{\pi(d-2)d} \end{pmatrix}.$$

Expanding $\bar{\Delta}$ by its αth and βth columns, we see that $\Delta = (-1)^{\alpha+\beta+1}\bar{\Delta}$. Now

$$y_\alpha(\alpha) = \det \begin{pmatrix} 1 & x_{\pi(0)1} & \cdots & x_{\pi(0),\beta-1} & x_{\pi(0),\beta+1} & \cdots & x_{\pi(0)d} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{\pi(d-2),1} & \cdots & x_{\pi(d-2),\beta-1} & x_{\pi(d-2),\beta+1} & \cdots & x_{\pi(d-2)d} \\ 0 & 0 \cdots 0 & 1 & 0 \cdots & \cdots & \cdots & 0 \\ & & (\alpha) & & & & \end{pmatrix}$$

and

$$y_\beta(\alpha) = \det \begin{pmatrix} 1 & x_{\pi(0)1} & \cdots & x_{\pi(0),\alpha-1} & x_{\pi(0),\alpha+1} & \cdots & x_{\pi(0)d} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_{\pi(d-2)1} & \cdots & x_{\pi(d-2),\alpha-1} & x_{\pi(d-2),\alpha+1} & \cdots & x_{\pi(d-2)d} \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \end{pmatrix};$$

hence expanding by the last row we get

$$y_\alpha(\alpha) = (-1)^{d+\alpha+1}\Delta, \qquad y_\beta(\alpha) = 0.$$

Similarly, we get

$$y_\alpha(\beta) = 0, \qquad y_\beta(\beta) = (-1)^{d+\beta}\Delta.$$

It follows that the orientation of the points $(0, 0)$, $(y_\alpha(\alpha), y_\beta(\alpha))$, $(y_\alpha(\beta), y_\beta(\beta))$ in the x_α, x_β-plane, with the coordinates taken in that order, is

$$\det \begin{pmatrix} 1 & 0 & 0 \\ 1 & (-1)^{d+\alpha+1}\Delta & 0 \\ 1 & 0 & (-1)^{d+\beta}\Delta \end{pmatrix} = (-1)^{\alpha+\beta+1} = \Delta\bar{\Delta}.$$

But since we are relabeling (x_α, x_β) either (u, v) or (v, u)—depending on whether Δ is $+1$ or -1 (resp.)—it follows that in the u, v-plane the orientation of the points O, $f(P_\alpha), f(P_\beta)$ is precisely $\bar{\Delta}$, i.e., matches that of the original $(d+1)$-tuple $P_0, \cdots, P_d$. A simple continuity argument then shows that the same conclusion must hold for *any* choice of $P_0, \cdots, P_d$ in general position. (The point is that our formulas for (y_α, y_β) give the correct projection to within a dilatation and/or a reflection in one or both coordinate axes, so they preserve collinearity and either preserve or reverse orientation; the above shows that they do, in fact, preserve it.)

Step 6 translates $f(P_{i_1})$ to the origin, and step 7 designates j as "good" if $f(P_j) \neq f(P_{i_1})$, i.e., if P_j is not in the flat $\langle P_{i_1}, \cdots, P_{i_{d-1}}\rangle$. In step 8, we compute the reflection in the origin of only the good points P_j with indices $j > i_{d-2}$, since these are all for which we have to compute $\lambda(i_1, \cdots, i_{d-2}, j)$; on the other hand we must take into account *all* the points P_k, even those with lower indices, when we count those on the positive side of $\langle P_{i_1}, \cdots, P_{i_{d-1}}, P_j\rangle$, hence we need *all* the slopes. Step 9 does the sorting for the good indices $j > i_{d-2}$, and steps 10 and 11 compute λ. In step 12, finally, we extend the λ-function to all ordered d-tuples; notice that every ordered d-tuple $(j_1, \cdots, j_d)$ with the j_k distinct and with $1 \leqq j_k \leqq n$ for every k is an even permutation of an ordered d-tuple $(i_1, \cdots, i_d)$, in fact of a unique one, with $i_1 < \cdots < i_{d-2} < \{i_{d-1}, i_d\}$—hence we never compute λ more than once for the same set of distinct indices. (If desired for the sake of compactness, in fact, the λ-array can be computed only for d-tuples with $i_1 < \cdots < i_{d-2} < \{i_{d-1}, i_d\}$; however, for comparison purposes this will not be enough, since when we renumber the points the indices will no longer form an increasing sequence—see § 5 below.)

The time required, for each $i_1, \cdots, i_{d-1}$ in step 3, is $O(d^2)$; in step 4 it is $O(d^2 n)$ (or faster), in steps 5, 6, 7 and 8 it is $O(n)$, in step 9 $O(n \log n)$, in steps 10 and 11 $O(n)$, and in step 12 $O(d!n)$. Since there are $O(n^{d-1}/(d-1)!)$ increasing $(d-1)$-tuples $i_1, \cdots, i_{d-1}$, the total time for Algorithm 3.1 is therefore

$$O((n^{d-1}/(d-1)!)(n(d!+\log n))) = O(n^d(d!+\log n)/(d-1)!).$$

In particular, if we think of d as fixed and n large, the time is $O(n^d \log n)$, which generalizes the time of $O(n^2 \log n)$ for Algorithm 2.1.

4. The canonical orderings of a configuration in $\mathbb{R}^d$. If a configuration $\mathscr{C}$ of points is given on a directed line, there is precisely one canonical way to order them; two if the line has no preferred orientation. But in the plane, or in higher dimensions, this is no longer true. Of course if our space is coordinatized, we can order lexicographically, say, by the coordinates; although such an ordering is useful for certain purposes, it is not intrinsic to the set, and would change if we performed an orientation-preserving affine transformation on the set; thus sets of the same "shape" or order type would have a great many such orderings. On the other hand, we could order the points of $\mathscr{C}$ by letting a hyperplane sweep across them; this would give, in general, $n(n-1)$ distinct orderings for a planar configuration, but two configurations of the same order type might have different sets of these orderings, even totally disjoint ones! (see [12, Thm. 1.7 (iv)$\Rightarrow$(vi)] for such an example). It is, however, possible to distinguish $k \leqq n$ of the $n!$ possible orderings of a planar configuration $\mathscr{C}$, where k is the number of vertices of the convex hull of $\mathscr{C}$, that turn out to be invariant under order-type-preserving transformations, and that we shall find useful in § 5 when we want to compare randomly numbered configurations. These we shall call the *canonical orderings* associated to $\mathscr{C}$. They are defined as follows:

If $\mathscr{C} = \{P_1(x_1, y_1), \cdots, P_n(x_n, y_n)\}$ is a configuration in the plane, and if P_i is any vertex of the convex hull conv $(\mathscr{C})$, consider a ray with origin P_i that begins by pointing

away from conv ($\mathscr{C}$), and that rotates counterclockwise. We list P_i first, then the remaining points $P_j (j \neq i)$ in the order in which the ray encounters them; if it meets $P_{j_1}, \cdots, P_{j_r}$ simultaneously, we list these in order of increasing distance from P_i. (If there are repeated points, we list these in any order at all.) This is illustrated in Fig. 2, with $i = 5$.

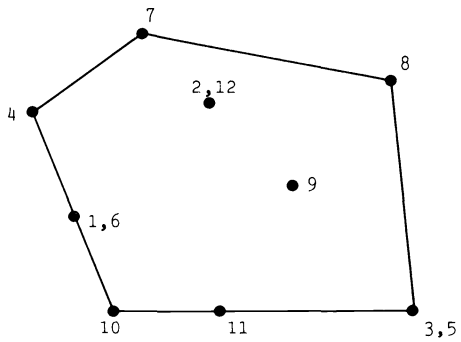


FIG. 2. 3, 5, 8, 9, 12, 2, 7, 4, 1, 6, 11, 10.

It is important to realize that if another configuration $\mathscr{C}' = \{P_1'(x_1', y_1'), \cdots, P_n'(x_n', y_n')\}$ is equivalent to $\mathscr{C}$, for example if $\mathscr{C}'$ is the image of $\mathscr{C}$ under an orientation-preserving affine transformation, then

(1) P_i will be an extreme point of $\mathscr{C}$ if and only if P_i' is an extreme point of $\mathscr{C}'$, and

(2) the canonical ordering of $\{1, \cdots, n\}$ associated to P_i will agree with that associated to P_i' (or will agree up to a permutation of the points within each class of repeated points if these happen to exist).

We can do the same thing in higher dimensions, as soon as we have the notion of a *face-flag*. If Π is a d-dimensional polytope lying in $\mathbb{R}^d$, we call a sequence $(\sigma_0, \cdots, \sigma_{d-2})$ of faces of Π a *face-flag*, if each σ_j is a face of σ_{j+1}, and if each σ_j is a face of Π of dimension j. (Note that if $\mathscr{C} = \{P_1, \cdots, P_n\}$ is a configuration in $\mathbb{R}^d$, and if $\Sigma = (\sigma_0, \cdots, \sigma_{d-2})$ is a face-flag of conv ($\mathscr{C}$), there are points $P_{i_0}, \cdots, P_{i_{d-2}} \in \mathscr{C}$ such that $\sigma_j = \langle P_{i_0}, \cdots, P_{i_j} \rangle$ for each $j = 0, \cdots, d-2$.) Now imagine a hyperplane H which "rotates around" σ_{d-2} in the positive direction (as determined by the coordinates in the order $x_1, \cdots, x_d$), starting in a supporting position. It meets the points of $\mathscr{C}$, perhaps several at a time, in some order $\mathscr{C}_1, \cdots, \mathscr{C}_r$, where all the points of σ_{d-2} are thought of as included in $\mathscr{C}_1$. Each subset $\mathscr{C}_i$ is a configuration lying in a $(d-1)$-dimensional space, hence the face-flag Σ determines an ordering of $\mathscr{C}_i$ into subsets $\mathscr{C}_{i1}, \cdots, \mathscr{C}_{is_i}$ by "rotating" a $(d-2)$-flat "around" σ_{d-3}, beginning with σ_{d-2} and moving through $\mathscr{C}_i$. We continue inductively, lowering the dimension by 1 each time. For the configuration illustrated in Fig. 3, for example, the ordering 3, 8, 5, 7, 2, 4, 1, 6 is induced by the face-flag $\langle 3 \rangle$, $\langle 3, 5 \rangle$.

There is another way to think of this canonical ordering, which we shall use in Algorithm 4.1 below. If $u_1, \cdots, u_d$ are Cartesian coordinates in $\mathbb{R}^d$, with origin O, we can define a spherical coordinate system $(\phi_1, \cdots, \phi_{d-1}, \rho)$ by letting, for each point $P \in \mathbb{R}^d$, $\phi_1(P)$ = the angle between $\overrightarrow{OP}$ and the positive u_1-axis, $\phi_2(P)$ = the angle between $\overrightarrow{OP}^{(2)}$ and the positive u_2-axis, where $P^{(2)}$ is the projection of P in the $u_2, \cdots, u_d$-hyperplane, $\cdots$, $\phi_{d-1}(P)$ = the angle between $\overrightarrow{OP}^{(d-1)}$ and the positive u_{d-1}-axis, where $P^{(d-1)}$ is the projection of P in the u_{d-1}, u_d-plane, and $\rho = |OP|$; in

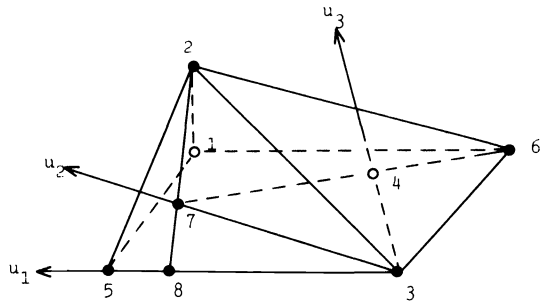


FIG. 3,

each case, if $\overrightarrow{OP}^{(i)} = 0$, call the corresponding angle 0 also. (This generalizes the usual spherical coordinate system in $\mathbb{R}^3$.) If $\mathscr{V} = \{\vec{v}_1, \cdots, \vec{v}_d\}$ is any real basis of the vector space $\mathbb{R}^d$, we may interpret the $\mathscr{V}$-coordinates $u_1, \cdots, u_d$ of any point P arising from the representation $\overrightarrow{OP} = \sum u_i \vec{v}_i$ as the Cartesian coordinates of a corresponding point P' in another copy of $\mathbb{R}^d$ (this amounts to performing an orientation-preserving linear transformation of $\mathbb{R}^d$ to itself), and compute the corresponding spherical coordinates $\phi_1, \cdots, \phi_{d-1}, \rho$. In particular, if $\sum = \sigma_0, \cdots, \sigma_{d-2}$ is a face-flag of conv $(\mathscr{C})$ and if $\sigma_j = \langle P_{i_0}, \cdots, P_{i_j} \rangle$ for every j, we may choose points $P_{i_{d-1}}$, P_{i_d} such that $P_{i_{d-1}}$ is on one of the two $(d-1)$-faces of conv $(\mathscr{C})$ meeting along σ_{d-2} and P_{i_d} is off $\langle P_{i_0}, \cdots, P_{i_{d-1}} \rangle$, and such that $P_{i_0}, \cdots, P_{i_d}$ has *positive* orientation (this determines *which* of the two faces $P_{i_{d-1}}$ is on!). Then take for $\mathscr{V}$ in the construction above the basis

$$\overrightarrow{P_{i_0}P_{i_1}}, \cdots, \overrightarrow{P_{i_0}P_{i_d}}.$$

This determines a coordinate system, hence (as above) a spherical coordinate system, and it is easy to see that the canonical ordering of $\{1, \cdots, n\}$ determined by the face-flag $\sigma_0, \cdots, \sigma_{d-2}$ is nothing more than the lexicographic ordering by $\phi_{d-1}, \cdots, \phi_1, \rho$. In Fig. 3, for example, the face-flag $\langle 3 \rangle$, $\langle 3, 5 \rangle$ can be augmented by the points 7, 4 (since 3, 5, 7, 4 has positive orientation in the "right-hand screw" sense), and this gives the basis $\overrightarrow{35}$, $\overrightarrow{37}$, $\overrightarrow{34}$, which—in turn—produces the ordering 3, 8, 5, 7, 2, 4, 1, 6 already indicated.

It is now easy to give a fast algorithm for determining the canonical ordering of a configuration $\mathscr{C}$ induced by a face-flag. The idea, as already suggested, is to compute the (generalized) spherical coordinates of each point of $\mathscr{C}$ with respect to the affine coordinate system induced by the face-flag, and to sort the points of $\mathscr{C}$ lexicographically with respect to their spherical coordinates. For ease of computation, we have chosen to use the squares of the cosines of the angles, rather than the angles themselves, as a sorting key (see Analysis below).

ALGORITHM 4.1.

Input: $(P_i(x_{i1}, \cdots, x_{id}))$, $i = 1, \cdots, n$, and $i_0, \cdots, i_{d-2} \in \{1, \cdots, n\}$, where $(\langle P_{i_0}, \cdots, P_{i_j} \rangle)$, $j = 0, \cdots, d-2$, is a face-flag of conv $(\{P_1, \cdots, P_n\})$.

Output: $\pi(1), \cdots, \pi(n)$, the permutation of $1, \cdots, n$ that is canonically associated to the given face-flag.

1. Step 3 of Algorithm 3.1 (with $i_0, \cdots, i_{d-2}$ in place of $i_1, \cdots, i_{d-1}$), yielding α, β, σ. [N.B.: The determinant in question cannot vanish for all j, k.]

2. Step 4 of Algorithm 3.1, yielding $y_\alpha(j)$, $y_\beta(j)$ for every $j = 1, \cdots, n$.

3. If $\sigma = +1$, let $u'(j) = y_\alpha(j)$ and $v'(j) = y_\beta(j)$ for each $j = 1, \cdots, n$; if $\sigma = -1$, let $u'(j) = y_\beta(j)$ and $v'(j) = y_\alpha(j)$ for each $j = 1, \cdots, n$.

4. For each $j = 1, \cdots, n$, let $u_j = u'(j) - u'(i_1)$ and $v_j = v'(j) - v'(i_1)$.

5. For each $j = 1, \cdots, n$, if $(u_j, v_j) \neq (0, 0)$, call j "good".

6. For every good $j = 1, \cdots, n$, let $u_{n+j} = -u_j$, $v_{n+j} = -v_j$, and let $m_j = m_{n+j} = v_j/u_j$ if $u_j \neq 0$.

7. Sort the indices $\{j \mid j \text{ is good}\} \cup \{n+j \mid j \text{ is good}\}$ into subsets, as follows:

(a) first those for which $u_j > 0$, using m_j as key;

(b) next those for which $u_j = 0$ and $v_j > 0$;

(c) next those for which $u_j < 0$, using m_j as key;

(d) finally those for which $u_j = 0$ and $v_j < 0$.

(In (a) and (c) we get a list of subsets; the order within each subset is irrelevant.) Say the result is

$$J_{11}, \cdots, J_{1s_1}, \cdots, J_{r1}, \cdots, J_{rs_r},$$

where the points with indices $J_{k1}, \cdots, J_{ks_k}$ constitute an entire subset, and there are r subsets all together.

8. For each $k = 1, \cdots, r$, let

$$n_k = |\{m \mid 1 \leqq m \leqq s_k, J_{km} \leqq n\}|.$$

9. If $n_1 = 0$, let k' be the first k with $n_k \neq 0$ and let $k'' = k' + r/2 - 1$; if $n_1 \neq 0$, let k'' be (the first k with $n_k = 0$) -1, and let

$$k' = \begin{cases} k'' + r/2 + 1 & \text{if this is} \leqq r, \\ 1, & \text{otherwise.} \end{cases}$$

10. Let $i_{d-1} = j_{k'1}$, $i_d = j_{k''1}$.

11. For every $j = 1, \cdots, n$ and every $k = 1, \cdots, d$, let $z_{jk} = x_{jk} - x_{i_0 k}$.

12. For every $p = 1, \cdots, d$ and every $q = 1, \cdots, d$, let

$$A_{pq} = (-1)^{p+q} \det ((z_{i_k m})_{k \neq q, m \neq p}).$$

13. For every $j = 1, \cdots, n$, and every $q = 1, \cdots, d$, let

$$w_{jq} = \sum_{p=1}^{d} z_{jp} A_{pq}.$$

14. For every $j = 1, \cdots, n$ and every $q = 1, \cdots, d$, let

$$\Phi_{jq} = \begin{cases} \dfrac{(-\operatorname{sgn} w_{jq}) w_{jq}^2}{w_{jq}^2 + \cdots + w_{jd}^2} & \text{if the denominator} \neq 0, \\ 1, & \text{otherwise.} \end{cases}$$

15. For every $j = 1, \cdots, n$, let $R_j = w_{j1}^2 + \cdots + w_{jd}^2$.

16. Sort $j = 1, \cdots, n$ lexicographically by $\Phi_{d-1}, \cdots, \Phi_1, R$, to determine the permutation π such that $\pi(1) \leqq \cdots \leqq \pi(n)$ in this lexicographic order.

17. Stop.

Analysis. Steps 1 through 8, just as in Algorithm 3.1, determine the number of points in each ray when the configuration is projected along the flat $\langle P_{i_0}, \cdots, P_{i_{d-2}} \rangle$ into a coordinate plane meeting the flat transversally. Since we know in advance that $\langle P_{i_0}, \cdots, P_{i_{d-2}} \rangle$ is an edge-flat, it follows that the projected configuration has $f(P_{i_0})$ as an extreme point, where f is the projection. Steps 9 and 10 locate two vertices projecting onto the edges adjacent to that extreme point, and chooses them so that the orientation of $f(P_{i_0})$, $f(P_{i_{d-1}})$, $f(P_{i_d})$ will be positive, hence also the orientation of

$P_{i_0}, \cdots, P_{i_{d-2}}, P_{i_{d-1}}, P_{i_d}$. Step 11 translates P_{i_0} to the origin. Step 12 computes the adjoint A of the matrix of components of the basis vectors $\overrightarrow{P_{i_0}P_{i_1}}, \cdots, \overrightarrow{P_{i_0}P_{i_d}}$, so that in step 13 we can get the components $w_1, \cdots, w_d$ of each vector $\overrightarrow{P_{i_0}P_j}$ in terms of this basis, using the standard change-of-basis formula from linear algebra. If ϕ_q is the qth spherical coordinate, $1 \leqq q \leqq d-2$, as defined in the discussion preceding Algorithm 4.1, its cosine is given by

$$\cos \phi_q = \begin{cases} w_q(w_q^2 + \cdots + w_d^2)^{-1/2} & \text{if } w_q^2 + \cdots + w_d^2 \neq 0, \\ 1, & \text{otherwise.} \end{cases}$$

To avoid unnecessary computations, we can use the squares of these numbers, together with the appropriate signs, as a sorting key, rather than the numbers themselves. In a similar way, we can use $R = \rho^2$ instead of ρ. Since we want to sort in order of increasing ϕ, i.e. decreasing $\cos \phi$, the signs have been reversed in the definition of Φ_q in step 14, so that—in step 16—we can sort with respect to the increasing order in *all* the variables.

The time for steps 1 through 8, as for steps 3 through 10 of Algorithm 3.1 for a single set of indices, is $O(d^3 + n \log n)$, and steps 9 and 10 are shorter. Step 11 takes $O(dn)$, step 12 takes $O(d^5)$, and steps 13, 14 and 15 take $O(d^2 n)$. Step 16 takes $O(dn \log n)$, giving a total time of $O(d^5 + d^2 n + dn \log n)$ for Algorithm 4.1, i.e. $O(n \log n)$ for d fixed.

5. An algorithm for comparing randomly numbered configurations. Except for the way in which two sets of points on a line have been labeled, they "look" the same from the point of view of their order. This is not true in the plane, however, or in higher dimensions. There is no way to permute the labels on the sets in Fig. 4a and b to get their λ- or Λ-functions to match up. Thus, for *unlabeled* sets of n points in dimension $\geqq 2$, many essentially distinct order types are possible.

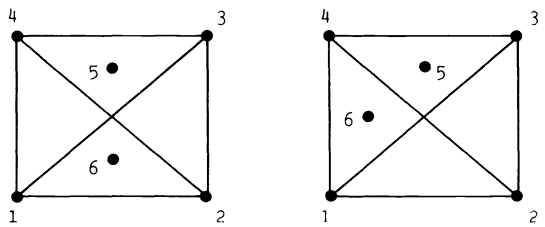
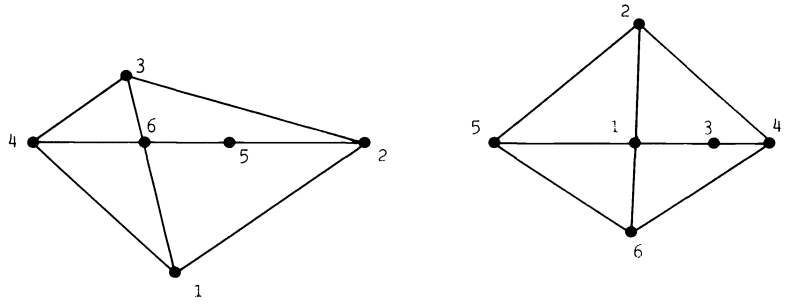


FIG. 4.

If $\mathscr{C}$ and $\mathscr{C}'$ are numbered configurations in $\mathbb{R}^d$, and we wish to compare them to see whether—*with the given numberings*—they are equivalent, we need only compare $\lambda(i_1, \cdots, i_d)$ for every $i_1, \cdots, i_d$. But suppose the given numberings do not match; how can we nevertheless determine whether they have the same order type, i.e., whether their points can be put in $1-1$ correspondence (and in fact find *all* such matchings) so that corresponding $(d+1)$-tuples are similarly oriented? If they are given in the form $\mathscr{C} = \{P_1, \cdots, P_n\}$, $\mathscr{C}' = \{P_1', \cdots, P_n'\}$, and we have encoded their (numbered) order types by calculating the λ-function of each, it would seem that we would have to try all the $n!$ possible renumberings of $\mathscr{C}'$ to be sure of finding all those (if any) in which the λ-function would agree with that of $\mathscr{C}$. But it turns out that we can eliminate most of these renumberings from consideration before we start. Let us



(a) $C = \{P_1, \cdots, P_6\}$ (b) $C' = \{P'_1, \cdots, P'_6\}$

FIG. 5.

illustrate this first by an example in the plane, where we can reduce the number of necessary renumberings to n, at most.

Fig. 5 shows two configurations that clearly have the same order type, but not in the numbering shown. How can we recognize this from their λ-matrices? We have

$$(\lambda_{ij}) = \begin{pmatrix} \omega & 4 & 1 & 0 & 3 & 1 \\ 0 & \omega & 4 & 1 & 1 & 1 \\ 2 & 0 & \omega & 4 & 1 & 2 \\ 4 & 1 & 0 & \omega & 1 & 1 \\ 1 & 1 & 3 & 1 & \omega & 1 \\ 2 & 1 & 1 & 1 & 1 & \omega \end{pmatrix}, \qquad (\lambda'_{ij}) = \begin{pmatrix} \omega & 1 & 1 & 1 & 1 & 2 \\ 2 & \omega & 1 & 0 & 4 & 2 \\ 1 & 3 & \omega & 1 & 1 & 1 \\ 1 & 4 & 1 & \omega & 1 & 0 \\ 1 & 0 & 1 & 1 & \omega & 4 \\ 1 & 1 & 3 & 4 & 0 & \omega \end{pmatrix},$$

and the problem is to find all permutations π of $\{1, \cdots, n\}$ such that $\lambda'_{\pi(i)\pi(j)} = \lambda_{ij}$ for every i, j. Consider the point P_1. Since it is a vertex of $\mathscr{C}$, $P'_{\pi(1)}$ will have to be a vertex of $\mathscr{C}'$ as well. *And the canonical ordering induced by $P'_{\pi(1)}$ in $\mathscr{C}'$ must agree (after the renumbering π) with that induced by P_1 in $\mathscr{C}$, namely* 125634. This means that if we list all of the vertices of $\mathscr{C}'$, and write down—for each—the canonical ordering it induces in $\mathscr{C}'$, π will have to map 125634 to one of those canonical orderings. Here we have 2, 4, 5 and 6 as vertices, inducing the following canonical orderings of $\mathscr{C}'$:

$$2{:}251634 \qquad 5{:}561342$$

$$4{:}423156 \qquad 6{:}643125.$$

Hence we need only try the four permutations

$$\begin{pmatrix} 1&2&5&6&3&4 \\ 2&5&1&6&3&4 \end{pmatrix}, \quad \begin{pmatrix} 1&2&5&6&3&4 \\ 4&2&3&1&5&6 \end{pmatrix}, \quad \begin{pmatrix} 1&2&5&6&3&4 \\ 5&6&1&3&4&2 \end{pmatrix}, \quad \begin{pmatrix} 1&2&5&6&3&4 \\ 6&4&3&1&2&5 \end{pmatrix}$$

to find π. Applying each of these in turn to λ gives

$$\begin{pmatrix} \omega & 1 & 3 & 1 & 1 & 1 \\ 3 & \omega & 1 & 0 & 4 & 1 \\ 1 & 2 & \omega & 4 & 0 & 2 \\ 1 & 4 & 0 & \omega & 1 & 1 \\ 1 & 0 & 4 & 1 & \omega & 1 \\ 1 & 2 & 1 & 1 & 1 & \omega \end{pmatrix}, \begin{pmatrix} \omega & 1 & 1 & 2 & 1 & 1 \\ 1 & \omega & 1 & 0 & 4 & 1 \\ 1 & 1 & \omega & 1 & 3 & 1 \\ 1 & 4 & 3 & \omega & 1 & 0 \\ 2 & 0 & 1 & 2 & \omega & 4 \\ 1 & 1 & 1 & 4 & 0 & \omega \end{pmatrix}, \begin{pmatrix} \omega & 1 & 1 & 3 & 1 & 1 \\ 1 & \omega & 1 & 0 & 4 & 1 \\ 1 & 1 & \omega & 1 & 2 & 1 \\ 1 & 4 & 2 & \omega & 2 & 0 \\ 3 & 0 & 1 & 1 & \omega & 4 \\ 1 & 1 & 1 & 4 & 0 & \omega \end{pmatrix}, \begin{pmatrix} \omega & 1 & 1 & 1 & 1 & 2 \\ 2 & \omega & 1 & 0 & 4 & 2 \\ 1 & 3 & \omega & 1 & 1 & 1 \\ 1 & 4 & 1 & \omega & 1 & 0 \\ 1 & 0 & 1 & 1 & \omega & 4 \\ 1 & 1 & 3 & 4 & 0 & \omega \end{pmatrix},$$

and we see that the last of these agrees with λ', so we must have

$$\pi = \begin{pmatrix} 1 & 2 & 5 & 6 & 3 & 4 \\ 6 & 4 & 3 & 1 & 2 & 5 \end{pmatrix}.$$

This idea extends to higher dimensions. There we must consider not only all the vertices of conv $(\mathscr{C})$, but all of its face-flags, as defined in § 4. Each of these induces a canonical ordering, and two configurations $\mathscr{C}$ and $\mathscr{C}'$ will have the same order type under the correspondence induced by $\pi : \{1, \cdots, n\} \to \{1, \cdots, n\}$ only if a fixed canonical ordering of $\{1, \cdots, n\}$ induced by the face-flag Σ (say) is transformed, by π, into the canonical ordering induced by the image $\Sigma^{(\pi)}$ of Σ in $\mathscr{C}'$. It is therefore sufficient to try all the permutations $\pi \in S_n$ which arise by associating to a fixed canonical numbering of $\mathscr{C}$ all the canonical numberings of $\mathscr{C}'$ (let us call such a renumbering of $\mathscr{C}$, induced by π, $\mathscr{C}^{(\pi)}$); among these will be all of the matching numberings, if any, of $\mathscr{C}$ and $\mathscr{C}'$. (If $\mathscr{C}$ has repeated points, then any renumbering of $\mathscr{C}$ which merely permutes the points within the groups of equal ones has no effect on its λ-function; for this reason it is sufficient, when obtaining the canonical numbering of a configuration associated to a face-flag, to list repeated points in any arbitrary order.)

How many face-flags are there for a configuration of n points in $\mathbb{R}^d$, and how do we find them all? It is not difficult to give a sharp upper bound on the number $\mathscr{F}(d, n)$ of face flags, hence of canonical orderings:

$$\mathscr{F}(d, n) \leqq \frac{d!}{2}\left[\binom{n - \lfloor (d+1)/2 \rfloor}{n-d} + \binom{n - \lfloor (d+2)/2 \rfloor}{n-d}\right].$$

This follows from the fact that there are no more than

$$\binom{n - \lfloor (d+1)/2 \rfloor}{n-d} + \binom{n - \lfloor (d+2)/2 \rfloor}{n-d}$$

faces of dimension $d-1$ for a simplicial polytope of dimension d, with equality for cyclic polytopes [15], and each such face gives rise to $d!$ face-flags $\sigma_0 \subset \sigma_1 \subset \cdots \subset \sigma_{d-1}$, of which precisely half have positive orientation; if a polytope is not simplicial, triangulating it will only increase the number of faces in each dimension, and it is easy to describe an injective map from the set of face-flags of the original polytope to the set of face-flags of its triangulaion. Hence we have

PROPOSITION 5.1. *A configuration of n points in the plane has $\leqq n$ canonical orderings, in 3-space has $\leqq 6n - 12$, and in $\mathbb{R}^d$ for d fixed has $O(n^{\lfloor d/2 \rfloor})$.*

As to the question of how to find all the face-flags, we shall describe four methods that can be used, depending on the circumstances:

Remark 5.2. (a) If $\mathscr{C}$ is a configuration in the plane, its face-flags are just its vertices, hence can be found by any algorithm which locates the vertices of the convex hull of a planar configuration [20]. They can also be found easily during the calculation of the λ-matrix, as in Algorithm 2.1, by inserting the following additional step after step 7:

7.1. If $\lambda(i, j) = 0$ and $n_{k(j)} = s_{k(j)}$, P_i is a vertex.

In fact, we can save time by calculating the canonical numberings themselves, during the sorting, as follows:

(i) Test each point P_i to see if it is an extreme point by checking whether $\lambda(i, j) = 0$ for some j and—if so—whether $n_{k(j)} = s_{k(j)}$ (i.e., whether the ray containing P_j contains no reflected points; this device is used again in method (b) below).

(ii) If i and j satisfy these conditions, sort the points $P_{J_{k1}}, \cdots, P_{J_{ks_k}}$ within the kth ray, for each k, by their distances from the origin (i.e., from P_i); let the result be $\bar{J}_{k1}, \cdots, \bar{J}_{ks_k}$.

(iii) Record the ordering

$$i_1, \cdots, i_p, \bar{J}_{k(j)+1,1}, \cdots, \bar{J}_{r,s_r}, \bar{J}_{11}, \cdots, \bar{J}_{k(j),s_{k(j)}},$$

where $i_1, \cdots, i_p$ are the "bad" indices, i.e., $P_{i_1} = \cdots = P_{i_p} = P_i$ (i will, of course, be among them). This will be the canonical ordering associated to the extreme point P_i.

(b) If $\mathscr{C}$ is a configuration in $\mathbb{R}^3$, its face-flags are just its directed edges, and there are algorithms (for example [3]) which locate all the edges of the convex hull of a 3-dimensional configuration. They can also be found during the calculation of the λ-function, by inserting the following additional steps after step 11 in Algorithm 3.1:

11.1. If $\lambda(i_1, i_2, j) \neq 0$ for all j, go to step 12; otherwise, let j_0 be the first j for which $\lambda(i_1, i_2, j) \neq 0$.

11.2. If $n_{k(j_0)} \neq s_{k(j_0)}$, go to step 12.

11.3. If $\gamma = \{1, 2, 3\} \backslash \{\alpha, \beta\}$, find $j_{\min}$ and $j_{\max}$, where

$$x_\gamma(j_{\min}) = \min \{x_\gamma(j) \mid j \text{ is not "good"}\}, \quad \text{and}$$

$$x_\gamma(j_{\max}) = \max \{x_\gamma(j) \mid j \text{ is not "good"}\}.$$

11.4. Enter the pairs $(j_{\min}, j_{\max})$ and $(j_{\max}, j_{\min})$ in the list of directed edges.

In addition, since we need the reflections of *all* the points in O in order to decide whether O is a vertex, Algorithm 3.1 should be further modified by executing steps 8, 9 and 11 for *every* good j, not merely those $\geqq i_1 + 1$; therefore we need only execute step 12 for the even permutations π that *fix the last index*.

(c) If $\mathscr{C}$ is a configuration in $\mathbb{R}^d$, for any $d \geqq 2$, and it is known that the points of $\mathscr{C}$ are in general position, i.e. that every set of $k+1 \leqq d$ points of $\mathscr{C}$ spans a k-flat,[1] then after sorting $\mathscr{C}$ we can get the face-flags of $\mathscr{C}$ very simply from the λ-function, as follows:

Whenever $\lambda(i_0, \cdots, i_{d-1}) = n - d$, then $i_0, \cdots, i_{d-2}$ determines a face-flag of $\mathscr{C}$.

If we let i_d be an arbitrary index $\neq i_0, \cdots, i_{d-1}$, then this gives, in fact, the *full* positively oriented face flags

$$\{P_{i_0}\} \subset \{P_{i_0}, P_{i_1}\} \subset \cdots \subset \{P_{i_0}, \cdots, P_{i_d}\},$$

so that we can omit steps 1 through 10 of Algorithm 4.1 if we wish to calculate the canonical orderings of $\mathscr{C}$.

Notice that if n points are given at random in $\mathbb{R}^d$, they *will* be in general position, in the sense above, since the set of n-tuples of points in $\mathbb{R}^d$ not in general position has measure zero in any bounded region of positive content. But all that is really needed to use method (c) is the weaker condition that conv ($\mathscr{C}$) be a *simplicial* polytope, i.e., that all of its $(d-1)$-faces be simplices. And it is very simple to check this from the λ-function: just check that whenever $\lambda(i_0, i_1, i_2, \cdots, i_{d-1}) = 0$ then $\lambda(i_1, i_0, i_2, \cdots, i_{d-1}) = n - d$. Thus method (c) will be applicable in many situations where the points of $\mathscr{C}$ are generated by some process with a random component.

[1] This can of course be checked from the λ-function itself: an obvious necessary and sufficient condition is that $\lambda(i_0, i_1, i_2, \cdots, i_{d-1}) + \lambda(i_1, i_0, i_2, \cdots, i_{d-1}) = n - d$ for every $i_0, \cdots, i_{d-1}$.

(d) If conv ($\mathscr{C}$) is not simplicial, or is not known to be, we cannot use method (c) to find its face-flags. If, in addition, the dimension of $\mathscr{C}$ is greater than 3, it then becomes necessary to resort to an algorithm, such as [3], that finds the $(d-1)$-faces of conv ($\mathscr{C}$), and to extend this recursively to find the complete lattice of faces of conv ($\mathscr{C}$). This will of course be rather time-consuming if d is large; it is, however, necessary if all the canonical orderings of $\mathscr{C}$ are desired, since these are in 1–1 correspondence with the face-flags, as we have seen, and the complete knowledge of these, in turn, is equivalent to the knowledge of the face lattice of conv ($\mathscr{C}$). Alternatively, we could use Algorithm 5.6 below, with step 2 modified by replacing the words "the first" by "each (in turn)"; this would give a time of $O(n^{d(d+3)/2})$ for finding all the face-flags, which is undoubtedly inefficient, albeit still polynomial.

We can now give the algorithm for sorting and comparing two randomly numbered configurations:

ALGORITHM 5.3.

Input: $\mathscr{C} = \{P_1(x_{11}, \cdots, x_{1d}), \cdots, P_n(x_{n1}, \cdots, x_{nd})\}$ and

$$\mathscr{C}' = \{P'_1(x'_{11}, \cdots, x'_{1d}), \cdots, P'_n(x'_{n1}, \cdots, x'_{nd})\}.$$

Output: $\{\pi \in S_n \mid \mathscr{C}^{(\pi)} \sim \mathscr{C}'\}$.

1. Sort $\mathscr{C}$ and $\mathscr{C}'$, using Algorithm 2.1 (if $d = 2$) or Algorithm 3.1 (if $d > 2$).
2. Using whichever of methods (a), (b), (c), (d) of Remark 5.2 applies, generate one face-flag $\Sigma = \langle i_0\rangle, \langle i_0, i_1\rangle, \cdots, \langle i_0, \cdots, i_{d-2}\rangle$ of $\mathscr{C}$.
3. Determine the permutation $\pi \in S_n$ corresponding to Σ, using Algorithm 4.1.
4. Similarly, generate *each* face-flag Σ' of $\mathscr{C}'$, in turn.
5. Determine the permutation $\pi' \in S_n$ corresponding to Σ', using Algorithm 4.1.
6. Let $\pi'' = \pi^{-1}\pi'$.
7. Compare $\lambda_{\mathscr{C}}(i_1, \cdots, i_d)$ and $\lambda_{\mathscr{C}'}(\pi''(i_1), \cdots, \pi''(i_d))$ for every choice of $i_1, \cdots, i_d$ with $i_1 < \cdots < i_{d-2} < \{i_{d-1}, i_d\}$. If they agree, record π''.
8. Return to step 4.
9. Stop.

Analysis. We have already seen what each of these steps does. As for the time required, it is quite variable. The sorting time (step 1), as we have seen, is $O(n^d \log n)$ for d fixed. Steps 2 and 4 can take anywhere from no appreciable extra time (if methods (a) or (b) apply) to an (indeterminately) long time if method (d) is invoked (but see Remark 5.5 below). In the case of a configuration in general position in $\mathbb{R}^d$, however (method (c)), we can always execute steps 2 and 4 in time $O(n^d)$. For each of the $O(n^{\lfloor d/2\rfloor})$ face-flags found in steps 2 and 4, steps 3 and 5 take time $O(n \log n)$ and step 7 takes $O(n^d)$; hence steps 3, 5, and 7—executed for all the face-flags—take $O(n^{\lfloor 3d/2\rfloor})$. This gives a total sorting and comparison time in the plane of $O(n^3)$, in $\mathbb{R}^3$ of $O(n^4)$, and in $\mathbb{R}^d (d \geqq 4)$—with a configuration in general position, or at least having a simplicial convex hull—of $O(n^{\lfloor 3d/2\rfloor})$.

Remark 5.4. This algorithm can also be used to find all the "order-symmetries" of a single configuration $\mathscr{C} = \{P_1, \cdots, P_n\}$ in $\mathbb{R}^d$, i.e., all permutations π of the indices giving equivalent configurations: just take $\mathscr{C}' = \mathscr{C}$.

Remark 5.5. When a configuration is sorted, if it is to be compared to other configurations, its canonical orderings should be determined at the same time. Then, when such a comparison is to be effected, only steps 6 and 7 of Algorithm 5.3 need be executed for each canonical ordering π' of $\mathscr{C}'$; this will cut down the comparison time considerably. This observation is particularly useful in applications where a dictionary of "standard" order types is to be encoded, and new configurations are to

be "looked up" in the dictionary (see for example Remark 6.4(a) below). In this case, when each "standard" configuration $\mathscr{C}'$ is encoded, we can compute all of its canonical orderings once and for all, and then whenever a new configuration $\mathscr{C}$ is offered for comparison it is only necessary to find *one* of its canonical orderings. Thus, after preprocessing the standard configurations, each new one will take only its sorting time $O(n^d \log n)$ to process, since *one* face-flag can easily be found in less time than that, no matter which of methods (a), (b), (c), or (d), is used.

Here, therefore, is an algorithm for finding *one* face-flag of a configuration in dimension $d \geqq 2$, directly from the λ-function of $\mathscr{C}$; it can be used when method (d) is needed:

ALGORITHM 5.6.

Input: The λ-function $(\lambda(i_1, \cdots, i_d))$, $1 \leqq i_j \leqq n$, $1 \leqq j \leqq d$, of a configuration $\mathscr{C}$.

Output: $j_0, \cdots, j_d$ such that $\langle j_0 \rangle, \cdots, \langle j_0, \cdots, j_d \rangle$ is a (positively oriented) face-flag of $\mathscr{C}$.

1. Let $D = d$ and let $S = \{1, \cdots, n\}$.
2. Find the first $i_1, \cdots, i_D \in S$ such that $\lambda(i_1, \cdots, i_D) = 0$.
3. Let $S = F$ and let $F = \{k \in S \mid \lambda(k, i_2, \cdots, i_D) = 0$ or $\lambda(i_1, k, i_3, \cdots, i_D) = 0$ or $\cdots$ or $\lambda(i_1, \cdots, i_{D-1}, k) = 0\}$.
4. Let j_D be the first index of S not in F.
5. For every $k_1, \cdots, k_{D-1} \in F$, let $\lambda_F(k_1, \cdots, k_{D-1}) =$ the number of indices $k \in F$ such that $\lambda(k_1, \cdots, k_{D-1}, k) = 0$; if $\lambda(k_1, \cdots, k_{D-1}, k) = \omega$ for *every* $k \in F$, let $\lambda_F(k_1, \cdots, k_{D-1}) = \omega$.
6. Replace D by $D-1$, and λ by λ_F.
7. If $D > 0$, go to step 2.
8. Let $j_0 =$ the first index in F.
9. Stop.

Analysis. Step 2 gives us points $P_1, \cdots, P_d$ spanning a face of conv $(\mathscr{C})$ that has no points of $\mathscr{C}$ on its positive side. Step 3 gives us the complete set of points lying on that face, i.e.,

$$F = \{k \mid P_k \in \langle P_{i_1}, \cdots, P_{i_d} \rangle\};$$

this follows from Lemmas 1.5 and 1.6. Step 4 will give us the face-flag: first we find a point off face $\langle P_{i_1}, \cdots, P_{i_d} \rangle$, next a point off one of *its* faces, and so on, down to dimension 0. Step 5 determines the λ-function for the face F, so that we can proceed inductively.

For each value of D from d down to 1, the time required for step 2 is $O(n^D)$, for step 3 $O(Dn)$, for step 4 $O(n)$, for step 5 $O(n^D)$, giving $O(n^D)$ all together. Hence the total time for Algorithm 5.6 with d fixed is $O(n^d)$.

6. Further remarks, applications, and open problems. How many different order types of numbered configurations of n points are there in $\mathbb{R}^d$? This is a difficult question, but we can easily find an upper bound, and we can find evidence to conjecture a lower bound. On the one hand, the fact that every such configuration can be geometrically sorted, i.e., encoded by an $n \times n \times \cdots \times n$ d-array of integers $\leqq n$, implies—for information-theoretic reasons—that we cannot have more than $\exp(cn^d \log n)$ distinct order types. On the other hand, we venture

CONJECTURE 6.1. *The number of inequivalent configurations of n points in* $\mathbb{R}^d$ *is at least* $\exp(cn^d)$.

Let us adduce some evidence for this conjecture, at least in the planar case.

Just as the λ-matrix can be used to encode the order type of a configuration of points in $\mathbb{R}^2$, it can be used, more generally, to encode the order type of what is called a *generalized configuration* [4], [11], [12]. This consists of n points in $\mathbb{R}^2$ which have been joined pairwise by *pseudolines* L_{ij} forming an *arrangement*, i.e., by simple curves that are straight lines outside of a bounded region, and any two of which meet just once (and necessarily cross there). The order type of such a generalized configuration can be described completely in terms of the connecting pseudolines L_{ij} [12]: the condition that P_i, P_j, P_k have positive orientation amounts to saying that the connecting pseudolines must have directions occurring in the cyclic order $L_{ij}, L_{ik}, L_{jk}, L_{ji}, L_{ki}, L_{kj}$ (see Fig. 6). (Since each L_{ij} is eventually straight we can speak unambiguously of its

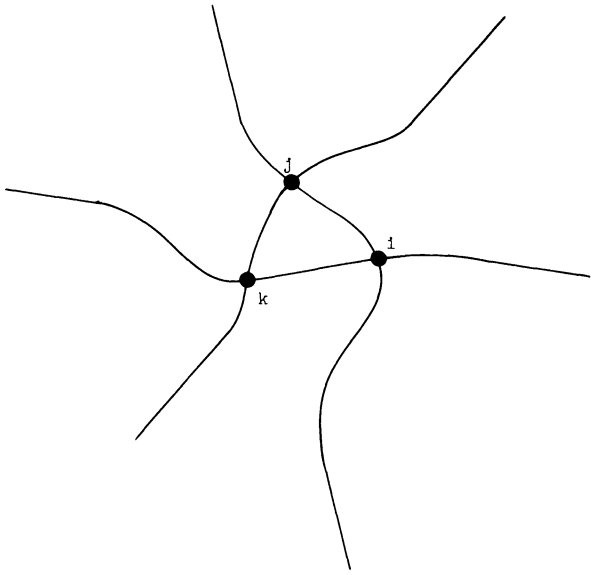


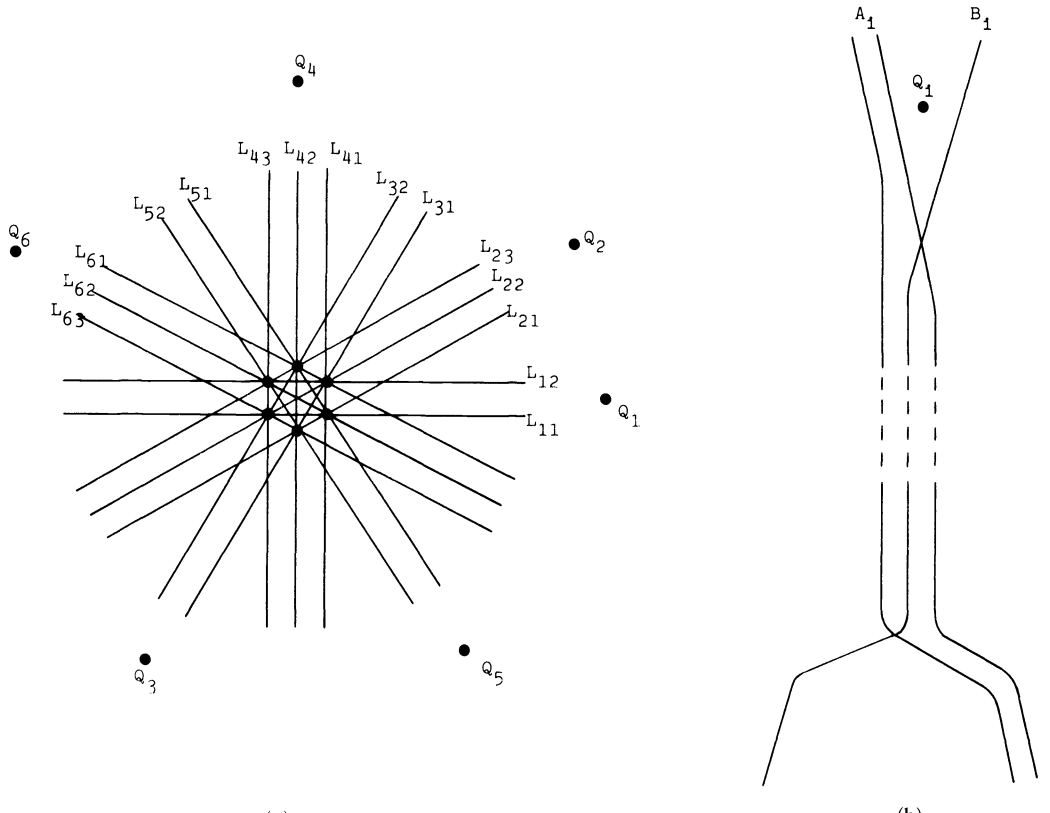
FIG. 6.

direction, which we take to be opposite to that of L_{ji}.) It follows from the main result of [10] that every generalized configuration of 8 or fewer points has the same order type as an ordinary configuration, while there are examples [9], [16] of 9 or more points for which this is not the case.

Theorem 1.8, which is the basis of our sorting procedure, holds for generalized configurations as well as ordinary ones [12], hence the order type of a generalized configuration is completely determined by its λ-matrix. While we cannot now prove Conjecture 6.1, even in the planar case, we do have

PROPOSITION 6.2. *There are at least* $2^{n^2/8}$ *generalized configurations of n points in the plane, for every n.*

Proof. Consider a regular k-gon with vertices $P_1, \cdots, P_k$, and draw all of its sides and diagonals extended fully (Fig. 7a). Let these be $L_{11}, \cdots, L_{1m_1}, \cdots, L_{k1}, \cdots L_{km_k}$, where $L_{i1}, \cdots, L_{im_i}$ form a complete set of parallels for each i. (If k is odd, we have $m_1 = \cdots = m_k = (k-1)/2$, while if k is even we have $m_1 = k/2$, $m_2 = k/2-1$, $m_3 = k/2$, $m_4 = k/2-1$, etc.) Now k new points, $Q_1, \cdots, Q_k$, are to be added, and we specify arbitrarily, for each i, which of $L_{i1}, \cdots, L_{im_i}$ are to come before Q_i, say L_{ij} for $j \in B_i$, and which after, say L_{ij} for $j \in A_i$, in the counterclockwise sense. In order to insert



(a) (b)

FIG. 7.

the Q_i, we first bend each L_{ij} with $j \in B_i$ slightly in the clockwise direction, and each L_{ij} with $j \in A_i$ slightly in the counterclockwise direction, and then insert Q_i after the two sets of L_{ij}'s have completely separated from each other (see Fig. 7b). Of course we must bend the other side of each L_{ij} as well, to ensure that the new L_{ij}'s, call them L_{ij}, still form an arrangement of pseudolines as defined above. Finally, we draw all the remaining connecting pseudolines P_iQ_j and Q_iQ_j, one at a time, with the help of the Levi enlargement lemma [16], which says that a pseudoline passing through two previously unconnected points can always be added to an arrangement to produce a new arrangement.

Since B_i was freely chosen for each i, and since two distinct choices of B_i for any i clearly give inequivalent configurations, regardless of the choices of the remaining B_j's, we have produced—for each even k—

$$(2^{k/2})^{k/2}(2^{k/2-1})^{k/2} = 2^{k^2/2-k/2},$$

and—for each odd k—

$$(2^{(k-1)/2})^k = 2^{k^2/2-k/2}$$

inequivalent generalized configurations, each having $2k$ points. Thus if $n = 2k$, we see that there are

$$2^{n^2/8-n/4}$$

inequivalent generalized configurations, each having n points, and if $n = 2k+1$ there are

$$2^{n^2/8-n/2+3/8}$$

such configurations. In each case, if we take into account the fact that each of the k new points can (independently) be placed on either side of the original k-gon, giving 2^k inequivalent versions of each generalized configuration, we can wipe out the linear terms in the exponent, and the result is that, as asserted, there are at least $2^{n^2/8}$ generalized configurations for every n.

Remark 6.3. By modifying the argument given, it is possible to improve the constant $\frac{1}{8}$ in Proposition 6.2 somewhat. If Conjecture 6.1 does turn out to be true, it will be an interesting problem to determine the constant c.

Remark 6.4. *Applications.* Since, as a result of Theorem 1.8 and Algorithm 5.3, it becomes possible to use a computer to sort and compare configurations of n points in $\mathbb{R}^d$ for moderate-sized n, a number of possible applications immediately suggest themselves. Among them are:

(a) *To pattern recognition.* There are various methods [1] of reducing an image to a point-pattern. Suppose we have a "dictionary" of standard images, each reduced to a black-and-white point pattern which has been encoded by its λ-matrix, and we wish to "look up" a new image, also encoded by its λ-matrix. If the transformation that has produced the new image is an affine orientation-preserving one, or at least one that preserves the relative orientations of points, such as a view of a solid object from a slightly different perspective than the "standard" view, the λ-matrices will agree, and they can be compared directly. If, on the other hand, the image has—in addition—undergone some local perturbations, as for example in hand-printed character analysis, its λ-matrix will not agree completely with a standard one, but will *correlate highly* with it. Thus one can measure the correlation of the λ-matrix with each one in the dictionary, and select the one giving the highest correlation; alternatively, if there are too many points to make such a comparison feasible, we can abstract *properties* of the standard λ-matrices (such as the proportion of extreme points, the number of hyperplanes cutting the configuration in half, and so on), and check off the corresponding properties of our new image against them. This procedure, especially when used in conjunction with other existing techniques (edge-detection, segmentation, noise-reduction), should prove highly useful in many scene- and pattern-analysis problems. For further details see [13].

(b) *To stereochemistry.* In [6] Dreiding and Wirth have suggested a method of encoding the order type of a configuration of points in $\mathbb{R}^d$, which they call a "multiplex", in order to provide an efficient way of distinguishing among stereoisomers, these being chemical compounds in which the same numbers of atoms are joined but with different orientations. (Each group of atoms that can exist in both a left- and a right-handed form is called a chirality element, and a single molecule may contain a number of these chemically distinguishable chirality elements.) Essentially, their encoding scheme consists of listing all the ordered $(d+1)$-tuples $i_1, \cdots, i_{d+1}$ with $i_1 < \cdots < i_{d+1}$ in lexicographic order, and writing 1 if the corresponding points are positively oriented and 0 if negatively. This gives a binary number for each order type, which they call its "signature", and amounts, essentially, to encoding what we called Λ earlier. (They consider only configurations in general position, and suggest a triadic representation if this is not the case.) Hence the storage (and therefore the minimum calculation time) for the signature of a configuration of n points in $\mathbb{R}^d$ is

$$\binom{n}{d+1} = \Omega(n^{d+1}) \quad \text{for } d \text{ fixed.}$$

Thus our sorting technique which consists of finding λ instead of Λ, whose time and storage requirement is only $O(n^d \log n)$, constitutes an improvement by a factor of $cn/\log n$, for n large, over the signature method. If, in applications to stereochemistry, one is interested in the orientations of only certain specified subsets (the chirality elements), a modification of our sorting scheme could easily be implemented, in which only each of these subsets is sorted; the result would then constitute an efficient means of encoding the various stereoisomers within a single class, which would facilitate their description and computer-aided identification.

(c) *To cluster analysis.* One of the problems in cluster analysis, for example in the method proposed in [7], is to find an efficient way of partitioning a set of n points in general position in $\mathbb{R}^d$ into two-disjoint subsets, separated by a hyperplane from one another [17]. This can be done in

$$\sum_{i=0}^{d} \binom{n-1}{i} \qquad (=2^{n-1} \text{ for } n-1 \leqq d)$$

distinct ways [17], [21], and Harding [17] suggests that an algorithm for enumerating these partitions "without effectively considering *en route* the remainder of the 2^{n-1} associations into two sets" would be of value. Such an algorithm follows immediately from our results. For all one has to do, at least in the case Harding is interested in, when $P_1, \cdots, P_n$ are in general position, is to modify our sorting algorithm, Algorithm 3.1, by *listing* the points P_k on the positive side of each hyperplane $\langle P_{i_1}, \cdots, P_{i_{d-1}}, P_j\rangle$ instead of *counting* them (i.e. calculating Λ instead of λ), and then adding, in turn, each subset of $\{P_{i_1}, \cdots, P_{i_{d-1}}, P_j\}$ to the points in $\Lambda(i_1, \cdots, i_{d-1}, j)$; these will be all the semispaces, i.e. the subsets of $\mathscr{C}$ lying on one side of some hyperplane, and the repetitions will not affect the order of magnitude in n. This procedure will clearly generate *all* the partitions by hyperplanes, since each hyperplane can be moved continuously without crossing any point of $\mathscr{C}$ until it passes through d points of $\mathscr{C}$.

The following are some problems that this work suggests:

Problem 6.5. Prove (or disprove) Conjecture 6.1, and—if it is true—find the value of c.

Problem 6.6. Find a criterion for a generalized configuration in the plane to be realizable by points, i.e., to be equivalent to a configuration in $\mathbb{R}^2$. This problem, which will shed light on the question of the optimality of our algorithms, can be thought of as a special case of the coordinatizability problem for oriented matroids: What we have been calling an equivalence class of generalized configurations is also known [2], [4], [8] as an orientation class of acyclic oriented matroids of rank 3, and our λ-classification gives a classification of oriented matroids of *every* rank $d+1$, as well as of ordinary configurations in every dimension d. A matroid that corresponds to an ordinary configuration is said to be *coordinatizable over* $\mathbb{R}$, and it is a difficult and long-outstanding problem to characterize these among all oriented matroids. In particular, what proportion of generalized configurations are equivalent to ordinary configurations? (One can conjecture that the proportion will approach zero as the number of points increases.)

Problem 6.7. Find a fast algorithm for generating the lattice of faces of conv $(\mathscr{C})$ for a configuration $\mathscr{C}$ *not in general position* in $\mathbb{R}^d$, possibly by using $\lambda_{\mathscr{C}}$. Since the function $\lambda_{\mathscr{C}}$ carries all the information about $\mathscr{C}$ essential to this question, it should be possible to do this, optimally in time $O(n^{\lfloor d/2\rfloor})$, since there are $O(n^{\lfloor d/2\rfloor})$ faces, as we have seen. The result will be useful, among other things, in shortening the time needed to compute the set of canonical orderings of $\mathscr{C}$, hence for the comparison algorithm, in the case where the configuration $\mathscr{C}$ is not in general position.

Problem 6.8. Characterize the function $\lambda_{\mathscr{C}}$. What are its defining properties as a function on all ordered d-tuples chosen from an n-set? How does it behave with respect to subconfigurations, intersections, and so on? This is related, of course, to the axiomatic description of oriented matroids in [2] and [8], and perhaps even more to the (equivalent) axiomatic description of *chirotopes* in [5], but it is far from clear how properties of the matroid structure, which are essentially properties of the function Λ, will carry over to properties of the more compact function λ.

Acknowledgments. We would like to express our appreciation to A. Dress and G. Purdy for some helpful discussions, and to the referees for their suggestions on how best to present the algorithms.

REFERENCES

[1] J. K. AGGARWAL, R. O. DUDA, AND A. ROSENFELD, eds., *Computer Methods in Image Analysis*, IEEE Press, New York, 1977.

[2] R. G. BLAND AND M. LAS VERGNAS, *Orientability of matroids*, J. Combin. Theory Ser. B, 24 (1978), pp. 94–123.

[3] D. R. CHAND AND S. S. KAPUR, *An algorithm for convex polytopes*, J. Assoc. Comput. Mach., 17 (1970), pp. 78–86.

[4] R. CORDOVIL, *Sur les matroïdes orientés de rang trois et les arrangements de pseudodroites dans le plan projectif réel*, European J. Combin., to appear.

[5] A. S. DREIDING, A. DRESS AND H. R. HAEGI, *Chirotopes, a combinatorial theory of orientation*, preprint.

[6] A. S. DREIDING AND K. WIRTH, *The multiplex—a classification of finite ordered point sets in oriented d-dimensional spaces*, Match, 8 (1980), pp. 341–352.

[7] A. W. F. EDWARDS AND L. L. CAVALLI-SFORZA, *A method for cluster analysis*, Biometrics, 21 (1965), pp. 362–375.

[8] J. FOLKMAN AND J. LAWRENCE, *Oriented matroids*, J. Combin. Theory Ser. B, 25 (1978), pp. 199–236.

[9] J. E. GOODMAN AND R. POLLACK, *On the combinatorial classification of nondegenerate configurations in the plane*, J. Combin. Theory Ser. A, 29 (1980), pp. 220–235.

[10] ———, *Proof of Grünbaum's conjecture on the stretchability of certain arrangements of pseudolines*, J. Combin. Theory Ser. A, 29 (1980), pp. 385–390.

[11] ———, *Helly-type theorems for pseudoline arrangements in* $\mathbb{R}^2$, J. Combin. Theory Ser. A, 32 (1982), pp. 1–19.

[12] ———, *Semispaces of configurations, cell complexes of arrangements*, to appear.

[13] ———, *The λ-matrix: a new tool for pattern recognition*, in preparation.

[14] R. L. GRAHAM, *An efficient algorithm for determining the convex hull of a planar set*, Inform. Process. Lett., 1 (1972), pp. 132–133.

[15] B. GRÜNBAUM, *Convex Polytopes*, Interscience-Wiley, London, 1967.

[16] ———, *Arrangements and Spreads*, CBMS Regional Conference Series in Applied Mathematics 10, American Mathematical Society, Providence, RI, 1972.

[17] E. F. HARDING, *The number of partitions of a set of n points in k dimensions induced by hyperplanes*, Proc. Edinburgh Math. Soc., 15 (1967), pp. 285–289.

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

[19] F. P. PREPARATA AND S. J. HONG, *Convex hulls of finite sets of points in two and three dimensions*, Comm. Assoc. Comput. Mach., 20 (1977), pp. 87–93.

[20] M. I. SHAMOS, *Computational geometry*, Ph.D. thesis, Yale Univ., New Haven, CT, 1978.

[21] T. ZASLAVSKY, *Facing Up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes*, Memoir 154, American Mathematical Society, Providence, RI, 1975.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0007 $01.25/0

SHELF ALGORITHMS FOR TWO-DIMENSIONAL PACKING PROBLEMS*

BRENDA S. BAKER† AND JERALD S. SCHWARZ‡

Abstract. This paper studies two approximation algorithms for packing rectangles, using the two-dimensional packing model of Baker, Coffman and Rivest [SIAM J. Comput., 9 (1980), pp. 846–855]. The algorithms studied are called next-fit and first-fit shelf algorithms, respectively. They differ from previous algorithms by packing the rectangles in the order given; the previous algorithms required sorting the rectangles by decreasing height or width before packing them, which is not possible in some applications. The shelf algorithms are a modification of the next-fit and first-fit decreasing height level algorithms of Coffman, Garey, Johnson and Tarjan [SIAM J. Comput., 9 (1980), pp. 808–826]. Each shelf algorithm takes a parameter r. It is shown that by choosing r appropriately, the asymptotic worst case performance of the shelf algorithms can be made arbitrarily close to that of the next-fit and first-fit level algorithms, without the restriction that items must be packed in order of decreasing height. Nonasymptotic worst case performance bounds are also investigated.

Key words. bin-packing, two-dimensional packing

1. Introduction. Two-dimensional packing problems arise in a variety of situations. For example, cutting-stock problems may involve cutting objects out of a sheet or roll of material so as to minimize waste. The scheduling of tasks with a shared resource involves two dimensions, the resource and time, and the problem is to pack (schedule) the tasks so as to minimize the total amount of time used. File management requires storing files in a storage medium, e.g., disk, while files may be created or destroyed over time. Baker, Coffman and Rivest [5] propose a combinatorial model for these situations. In this model, a rectangular bin with an open top (i.e., a semi-infinite dimension) is to be packed with a finite list of rectangles, of specified dimensions, such that the rectangles do not overlap and the total bin height used in the packing is as small as possible. The rectangles (also called "pieces") are to be packed with their sides parallel to the sides of the bin, and they may not be rotated. Such a packing is called an *orthogonal oriented* packing in the terminology of [5]. This model is illustrated in Fig. 1. For the cutting-stock problem, the width and height of the bin correspond to the width and length of the roll of material. For a computer scheduling application, where the shared resource is the main memory, the bin width corresponds to memory, while the semi-infinite height corresponds to time. This model is appropriate for studying the scheduling of tasks which share a resource such as memory, where each task requires a contiguous block of the resource. The scheduling of resources such as disk drives for which fragmentation is not an issue has been studied by Garey and Graham [8] using a fundamentally different multidimensional packing model.

For the Baker et al. model, the problem of determining an optimal packing of an arbitrary list of rectangles is NP-hard [5]. Therefore, Baker et al. [5], Golan [10] and Coffman et al. [7] studied polynomial time approximation algorithms for the problem. These algorithms all have the property that they sort the rectangles by decreasing height or width before packing them.

In many applications, the rectangles can be considered to form a queue which can be joined by new rectangles at any time. For example, in the computer scheduling

* Received by the editors January 29, 1979, and in final revised form July 9, 1982.
† Bell Laboratories, Murray Hill, New Jersey 07974.
‡ Bell Laboratories, Whippany, New Jersey 07981.

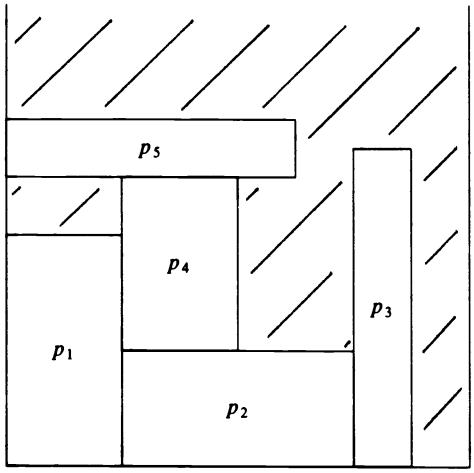


FIG. 1. *A possible packing of a list* $(p_1, p_2, p_3, p_4, p_5)$.

application with memory as the shared resource, a batch system could sort jobs by time or memory requirements before executing them, but an on-line system must handle jobs as they are created. In a warehouse, the boxes already there must be stored before new boxes arrive. In these situations it is not reasonable to wait for all the rectangles to enter the queue before packing them. Therefore, this paper studies polynomial time approximation algorithms (called *shelf* algorithms) which pack the rectangles in the order specified by the given list (queue), without sorting them first. Unlike the previous algorithms, the shelf algorithms can pack an infinite list of rectangles such that for any finite initial segment of the list, the height of the packing is within a constant times the height of an optimal packing of that segment.

The shelf algorithms are appropriate for situations such as the warehouse storage problem where (a) items to be packed are not all known in advance and (b) both dimensions represent resources fixed over time. The shelf algorithms will not handle applications in which one dimension is time, as in the computer scheduling application. Baker and Coffman [4] investigate algorithms for the situation where (a) items to be packed arrive over time, (b) one dimension is time and (c) preemption is allowed. The case in which items are not known in advance, one dimension is time and no preemption is allowed awaits further research.

Section 2 describes two kinds of shelf algorithms called next-fit and first-fit shelf algorithms. These are modifications of the next-fit decreasing height (NFDH) and first-fit decreasing height (FFDH) algorithms studied by Coffman et al. [7], but without the restriction of the previous algorithms that pieces be packed in order of decreasing height. Each shelf algorithm takes a parameter r, $0 < r < 1$, which is a measure of how much space is allowed in each shelf to handle variations in height of rectangles to come. The next-fit and first-fit algorithms with parameter r are called NFS_r and FFS_r, respectively.

Section 3 investigates the worst case bounds for NFS_r and FFS_r on lists of rectangles. The bounds obtained for the shelf algorithms fall into two classes: absolute worst case and asymptotic worst case. Let $A(L)$ denote the height used by algorithm A in packing list L, and let $\mathrm{OPT}(L)$ denote the height of an optimal packing of L. If α, β and H are constants such that for every list L of rectangles of height at most

H, $A(L) \leqq \alpha\ \mathrm{OPT}(L) + \beta H$, then α is called an *asymptotic* worst case bound for A. If α is a constant such that for every list L of rectangles, $A(L) \leqq \alpha\ \mathrm{OPT}(L)$, α is called an *absolute* worst case bound for A. The absolute worst case bound appears to be a better measure of performance when the number of rectangles to be packed is small, while the asymptotic bound is a better measure when the number of rectangles is large. Theorem 1 states that NFS_r has a tight asymptotic worst case bound of $2/r$. Corollary 1.1 obtains a tight absolute worst case bound of $2/r + 1/r(1-r)$ for NFS_r. Corollary 1.2 states that this absolute worst case bound is minimized at $r \approx .634$, at which value the bound is approximately 7.46. The main theorem is Theorem 2, which states that FFS_r has a tight asymptotic worst case bound of $1.7/r$. This theorem is important because it shows that we can get asymptotic performance arbitrarily close to the bound of 1.7 for FFDH [7], without the FFDH restriction that rectangles must be packed in order of decreasing height. Corollary 2.1 shows that $1.7/r + 1/r(1-r)$ is a tight absolute bound for FFS_r. Corollary 2.2 shows that this absolute worst case bound is minimal at $r \approx 622$, at which value it is approximately 6.99.

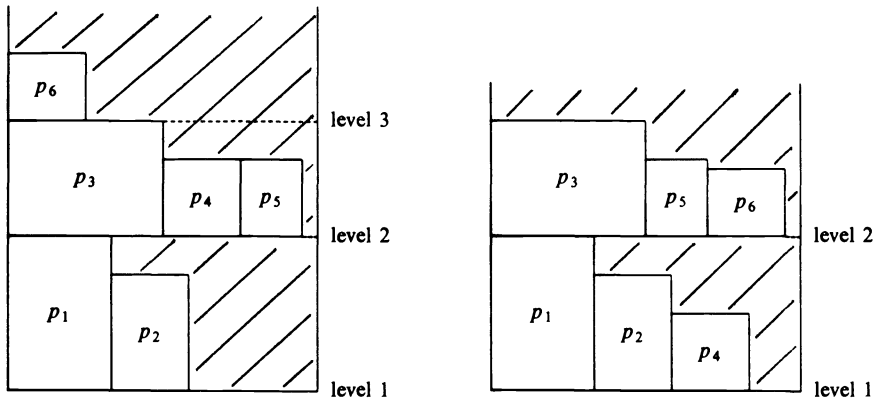
Section 4 investigates the worst case bounds for NFS_r and FFS_r on lists of squares. Theorem 3 proves a tight absolute worst case bound of $1/r + 1/r(1-r)$ for NFS_r and FFS_r on lists of squares. Note that this bound is lower than the absolute bound for arbitrary lists of rectangles. Corollary 3.1 shows that this absolute worst case bound for lists of squares is minimal at $r \approx .586$, at which value it is approximately 5.83. Proposition 1 and Theorems 4–6 investigate asymptotic worst case bounds on the NFS_r and FFS_r packings for squares. The bounds obtained are not tight, but they suggest that the asymptotic worst case performance is not significantly better for squares than for rectangles in general.

2. Shelf algorithms. Since the shelf algorithms are a modification of the next-fit and first-fit level algorithms, we begin by describing these.

The next-fit decreasing height (NFDH) level algorithm packs pieces in rows in order of decreasing height. The bottom of each row is called a *level*. The first level is the bottom of the bin. Pieces are packed from left to right, with their bottoms at the current level, until the next piece is too wide to fit. At this point, a new level is created along the horizontal line which coincides with the top of the tallest piece in the current level and packing continues at the new level. An NFDH packing is shown in Fig. 2a. For every list L of rectangles of height at most 1, $\mathrm{NFDH}(L) \leqq 2\ \mathrm{OPT}(L) + 1$, and this bound is tight [7]. However, for any $k > 1$, there is a list L such that the height of the NFDH packing of L is at least $k\ \mathrm{OPT}(L)$ if the pieces are packed in the order given by L rather than in order of decreasing height (for a bin of width 1 and sufficiently small ε, let $L = (r_0, s_0, r_1, s_1, \cdots, r_k, s_k)$, where each r_i has height $1 + (i-k)\varepsilon$ and width ε, while each s_i has width 1 and height ε).

The first-fit decreasing height (FFDH) level algorithm is similar to the NFDH algorithm except that each piece is packed on the lowest level where it will fit, rather than on the top level; a new level is created when a piece won't fit on any existing level. An FFDH packing is shown in Fig. 2b. For every list L whose rectangles have height at most 1, $\mathrm{FFDH}(L) \leqq 17/10\ \mathrm{OPT}(L) + 1$ [9]. However, if pieces are packed in the order given by L rather than by decreasing height, the performance can be arbitrarily bad compared to optimal.

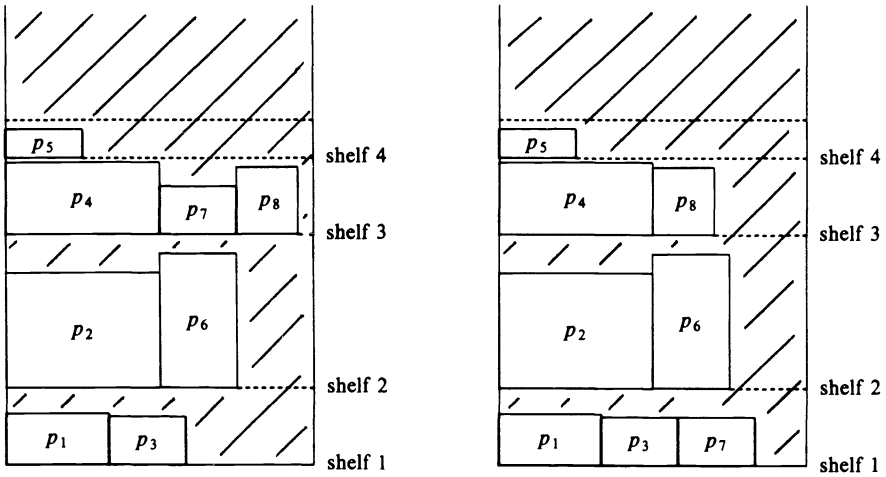
In order to get algorithms which pack rectangles in a reasonable fashion without sorting first, we design our shelf algorithms to waste space purposely so that taller pieces arriving later can be accommodated. The name shelf algorithm is chosen by analogy with bookshelves, which can be filled in any order. Each shelf algorithm takes a parameter r, which is a measure of how much wasted space can occur. The shelf



a) NFDH *packing* b) FFDH *packing*

FIG. 2. NFDH *and* FFDH *packings of a list* $L = (p_1, p_2, p_3, p_4, p_5, p_6)$.

algorithms pack pieces in rows (shelves), but each shelf is given a fixed height of r^k (for some integer k) when it is created. A rectangle of height h, $r^{k+1} < h \leqq r^k$, must be packed into a shelf of height r^k. The next-fit shelf algorithm with parameter r (denoted by NFS_r) packs each rectangle R as far left as possible into the top shelf of the appropriate height, if such a shelf exists and has room for R. Otherwise, a new shelf of the appropriate height is created on top of the top shelf in the bin, and R is placed at the left end of this shelf. An $\mathrm{NFS}_{.5}$ packing is shown in Fig. 3a. The first-fit shelf algorithm with parameter r (denoted by FFS_r) is similar except that each rectangle R is placed as far left as possible in the *lowest* shelf which is of the appropriate height and has room for R, if such a shelf exists, and into a new shelf of that height, otherwise. An $\mathrm{FFS}_{.5}$ packing is shown in Fig. 3b. It will be shown that as r approaches 1, the asymptotic performance of NFS_r and FFS_r improves. In fact, Theorem 2 shows that appropriate choice of r can bring FFS_r arbitrarily close to the asymptotic bound of



a) $\mathrm{NFS}_{.5}$ *packing* b) $\mathrm{FFS}_{.5}$ *packing*

FIG. 3. $\mathrm{NFS}_{.5}$ *and* $\mathrm{FFS}_{.5}$ *packings of a list* $L = (p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$.

17/10 obtained for the FFDH algorithm [7], even though the restriction of packing in order of decreasing height has been removed.

To simplify the proofs, it will be convenient to divide the shelves into two groups. The last shelf created for each height will be called *sparse*; the other shelves will be called *dense*. Let H_S denote the sum of the heights of the sparse shelves, and H_D the sum of the heights of the dense shelves.

3. Worst case bounds for lists of rectangles. This section investigates the asymptotic and absolute worst case bounds for NFS_r and FFS_r.

THEOREM 1. *For any constant r, $0<r<1$, and any list L whose rectangles have height at most H, $\mathrm{NFS}_r(L)<(2/r)\,\mathrm{OPT}(L)+H/r(1-r)$. Moreover, the asymptotic bound of $2/r$ is tight.*

Proof. We will show that $H_S<H/r(1-r)$ and that $H_D<(2/r)\,\mathrm{OPT}(L)$. Let the bin width be w.

First, we consider the sparse shelves. Let h be the size of the tallest sparse shelf. Within each shelf, the height of each piece is greater then r times the height of the shelf. Thus, $H>rh$ and

$$H_S<h\sum_{i=0}^{\infty} r^i=\frac{h}{1-r}<\frac{H}{r(1-r)}.$$

Now, we consider the dense shelves. For any dense shelf S, the next shelf S' of the same height as S contains a piece which would not fit in S. Consequently, the total width of the pieces in S and S' is at least w. Since each piece is taller than r times the height of its shelf, the total area of the pieces in S and S' is greater than $r/2$ times the area of S and S'. If we pair the shelves of each height in this manner (including the sparse shelf if the total number of dense shelves of that height is odd), we see that the total area of the rectangles (including the ones in the sparse shelves) is greater than $(rw/2)H_D$ and cannot be packed in a height less than $(r/2)H_D$. Consequently,

$$H_D<\frac{2\,\mathrm{OPT}(L)}{r}.$$

To show tightness, let the bin width be 1 and consider lists of the form $L=(r_1, s_1, r_2, \cdots, r_n, s_n)$, where $n>0$, each r_i has width $1/2-\Delta$ (for some Δ, $0<\Delta<1/3n$), each s_i has width 3Δ and all pieces have height $r+\varepsilon$ (for some ε, $0<\varepsilon<1-r$). The NFS_r packing places the r_i's and s_i's in pairs in shelves of height 1, using a total height of $n-1+r+\varepsilon$. An optimal packing is no worse than first packing the r_i's two to a row and then packing the s_i's in a single row on top for a total height of at most $(\lceil n/2\rceil+1)(r+\varepsilon)$. For $\alpha<2/r$ and β any constant, there exist sufficiently large n and sufficiently small ε that $n-1+r+\varepsilon>\alpha(\lceil n/2\rceil+1)(r+\varepsilon)+\beta(r+\varepsilon)$. □

Note that if $H=r^i$ for any integer i, the tallest shelf has height at most H rather than H/r, and the additive constant in Theorem 1 becomes $H/(1-r)$ rather than $H/r(1-r)$.

COROLLARY 1.1. *For any constant r, $0<r<1$, and any list L of rectangles,*

$$\mathrm{NFS}_r(L)<\left[\frac{2}{r}+\frac{1}{r(1-r)}\right]\mathrm{OPT}(L).$$

Moreover, this bound is tight.

Proof. Since $H \leqq \mathrm{OPT}(L)$,

$$\mathrm{NFS}_r(L) < \frac{2}{r}\mathrm{OPT}(L) + \frac{H}{r(1-r)} \leqq \left[\frac{2}{r} + \frac{1}{r(1-r)}\right]\mathrm{OPT}(L)$$

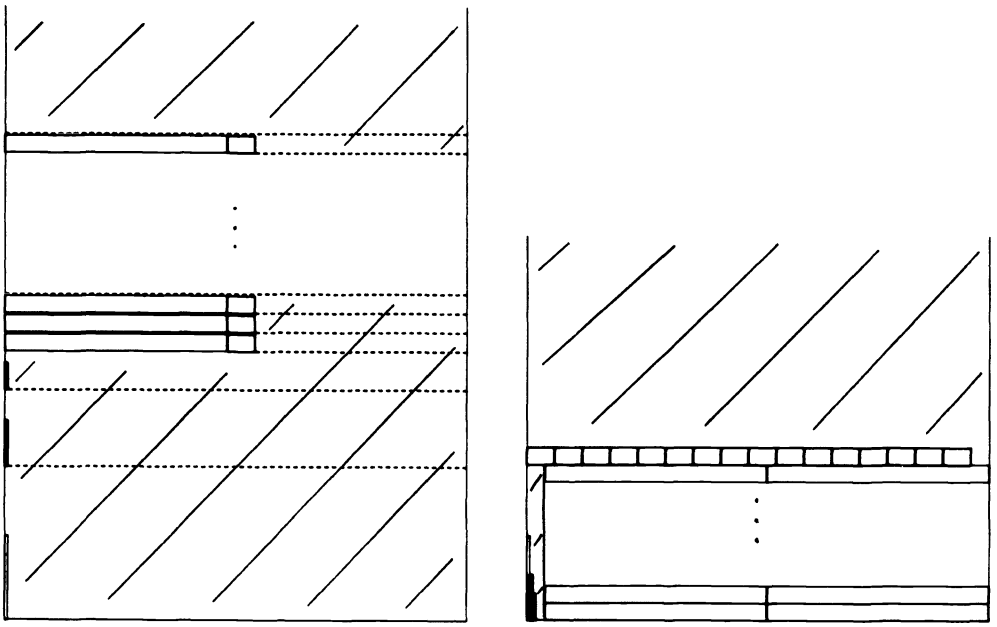
by Theorem 1. To show tightness, we will construct a list L such that in the NFS_r packing, the sparse shelves have a total height close to $1/r(1-r)$, the area of the dense shelves is close to $r/2$ times the area of the rectangles packed in them and the height of the optimal packing is close to the height of the tallest piece. Let the bin width be 1.

Let k be a positive integer. Let $n = 2\lceil r^{-k}\rceil$. Choose Δ and ε so that $3n\Delta < 1$ and $\varepsilon < r^k - r^{k+1}$. L will consist of pieces $(p_1, p_2, \cdots, p_k, t_1, s_1, t_2, s_2, \cdots, t_n, s_n)$, where each p_i has width Δ/k and height $r^i + \varepsilon$, each t_i has width $1/2 - \Delta$ and height $r^{k+1} + \varepsilon$ and each s_i has width 3Δ and height $r^{k+1} + \varepsilon$.

In the NFS_r packing, each p_i goes by itself in a shelf of height r^{i-1}, and the t_i's and s_i's are packed in pairs in shelves of height r^k (t_{i+1} will not fit into the shelf with t_i and s_i since their total width is $1/2 + 2\Delta$). The packing is shown in Fig. 4a. Thus,

$$\begin{aligned}\mathrm{NFS}_r(L) &\geqq \sum_{i=0}^{k-1} r^i + nr^k - (r^k - (r^{k+1} + \varepsilon)) \\ &= \frac{1}{1-r} - \frac{r^k}{1-r} + 2\lceil r^{-k}\rceil r^k - (r^k - (r^{k+1} + \varepsilon)) \\ &\geqq \frac{1}{1-r} - \frac{r^k}{1-r} + 2 - (r^k - r^{k+1}).\end{aligned}$$

An optimal packing of L is no worse than the following, depicted in Fig. 4b. Pack the p_i's at the bottom left of the bin, using width Δ and height $r + \varepsilon$. To the right



a) *The* $\mathrm{NFS}_{.5}$ *packing* b) *An optimal packing*

FIG. 4. *Packings used in the proof of Corollary* 1.1 *with* $r = .5$, $k = 3$.

of the p_i's, pack the t_i's in rows of width $1-2\Delta$ and height $r^{k+1}+\varepsilon$. Since two of the t_i's fit in each row, the t_i's use at most height $(n/2)(r^{k+1}+\varepsilon) \leqq (1+r^{-k})(r^{k+1}+\varepsilon) \leqq r^{k+1}+r+(1+r^{-k})\varepsilon$. Finally, pack the s_i's in a row above the other pieces, using height $r^{k+1}-\varepsilon$. The total height used is at most $2r^{k+1}+r+(2+r^{-k})\varepsilon$. For any $\delta>0$, by taking k sufficiently large and ε sufficiently small, we get $\mathrm{NFS}_r(L) > [2/r+1/r(1-r)-\delta]\,\mathrm{OPT}(L)$. □

It is interesting to note that the asymptotic performance of NFS_r is nearly as good as that of NFDH, which has a tight asymptotic bound of 2 [7]. In fact, r can be chosen to bring the asymptotic performance as close as desired to 2, without the NFDH restriction that pieces be packed in order of decreasing height. Unfortunately, the absolute worst case performance gets worse as the asymptotic performance gets better. However, absolute worst case performance is crucial only for small numbers of pieces, and the shelf algorithms are intended for packings of large numbers of pieces.

Since the bound of Corollary 1.1 is tight, we can find the value of r with the best absolute worst case performance by differentiating the expression for the bound to find its minimum.

COROLLARY 1.2. *The value of* r, $0<r<1$, *for which the absolute worst case performance bound of* NFS_r *is minimal is* $(3-\sqrt{3})/2 \approx .634$, *at which value* $\mathrm{NFS}_r(L) \approx 7.46\,\mathrm{OPT}(L)$ *for every list* L *of rectangles.*

One's intuition might suggest that the algorithm ought to do best at $r=1/2$. In fact, Theorem 2 shows that at $r=1/2$, $\mathrm{NFS}_r(L)$ can be arbitrarily close to 8 $\mathrm{OPT}(L)$, so that Corollary 1 gives a noticeable, if not large, improvement over $r=1/2$.

THEOREM 2. *For any* r, $0<r<1$, *and any list* L *of rectangles of height at most* H,

$$\mathrm{FFS}_r(L) < \frac{1.7}{r}\,\mathrm{OPT}(L) + \frac{H}{r(1-r)}.$$

Moreover, the asymptotic bound of $1.7/r$ *is tight.*

Proof. The proof is a modification of a proof from one-dimensional bin-packing. The one-dimensional bin-packing problem has a list L of numbers in the range $(0, 1)$ which are to be packed in bins, each of which can hold a total of at most 1, and the goal is to minimize the number of bins used. This problem is equivalent to the two-dimensional problem with the restriction that all rectangles have the same height. In particular, the first-fit (FF) one-dimensional algorithm is equivalent to FFDH when all the rectangles have the same height. Garey, Graham, Johnson and Yao [9] showed that $\mathrm{FF}(L) \leqq 1.7\,\mathrm{OPT}(L)+1$ for every list L, where $\mathrm{FF}(L)$ and $\mathrm{OPT}(L)$ represent the number of bins used by FF and an optimal packing, respectively. We generalize the proof to handle shelves of different heights. Since the one-dimensional proof deals with bins which can hold at most 1, we assume without loss of generality that the bin width has been normalized to 1.

The original proof uses the following weighting function $W:[0,1]\to[0,8/5]$.

$$W(\alpha) = \begin{cases} \frac{6}{5}\alpha & \text{for } 0 \leqq \alpha \leqq \frac{1}{6}, \\ \frac{9}{5}\alpha - \frac{1}{10} & \text{for } \frac{1}{6} < \alpha \leqq \frac{1}{3}, \\ \frac{6}{5}\alpha + \frac{1}{10} & \text{for } \frac{1}{3} < \alpha \leqq \frac{1}{2}, \\ \frac{6}{5}\alpha + \frac{4}{10} & \text{for } \frac{1}{2} < \alpha \leqq 1. \end{cases}$$

The original proof showed that if $\bar{W}$ represents the sum of the weights of all pieces in L, then $\mathrm{FF}(L) \leqq \bar{W}+1$ and $\bar{W} \leqq 1.7\,\mathrm{OPT}(L)$. We extend the weighting idea of two dimensions by considering weighted areas. For each piece p, let $w(p)$ and $h(p)$

represent the width and height of p, respectively. Let $\bar{A} = \sum_{p \in L} W(w(p))h(p)$. We will show that $\text{FFS}_r(L) < \bar{A}/r + H/r(1-r)$ and $\bar{A} \leqq 1.7\ \text{OPT}(L)$, and the result follows.

The original proof showed that for any packing of any list L, the sum of the weights of the pieces in any bin is at most 1.7 and deduced from this that $\bar{W} \leqq 1.7\ \text{OPT}(L)$. We can instead apply their proof along any horizontal line through the bin to find that the sum of $W(q_i)$ for the pieces q_i along this line is at most 1.7 and then integrate over the height of the optimal packing to obtain $\bar{A} \leqq 1.7\ \text{OPT}(L)$.

It remains to show that $\text{FFS}_r < \bar{A}/r + H/r(1-r)$. For $k \geqq 0$, let N_k be the number of shelves of height r^k. For $k \geqq 0$, let L_k be the sublist of L consisting of pieces whose heights are in the range $(r^{k+1}, r^k]$. If L_k' is the list of widths of pieces in L_k, then $\text{FF}(L_k') = N_k$. By [9],

$$\sum_{p \in L_k} W(w(p)) \geqq \text{FF}(L_k') - 1 = N_k - 1.$$

Since each piece is greater than r times the height of the shelf it is packed in,

$$\sum_{p \in L_k} W(w(p))h(p) \geqq r^{k+1}(N_k - 1).$$

Consequently, if the tallest shelf has height r^i,

$$\bar{A} \geqq \sum_{k=i}^{\infty} \sum_{p \in L_k} W(w(p))h(p) \geqq \sum_{k=i}^{\infty} r^{k+1}(N_k - 1)$$

$$= r\ \text{FFS}(L) - \frac{r^{i+1}}{(1-r)} > r\ \text{FFS}(L) - \frac{H}{(1-r)}$$

and

$$\text{FFS}(L) \leqq \frac{\bar{A}}{r} + \frac{H}{r(1-r)}.$$

Johnson et al. [11] proved that for every positive integer n, there is a list L for which $\text{OPT}(L) = n$ and the one-dimensional first-fit (FF) algorithm has $\text{FF}(L) \geqq 1.7n - 8$. If we turn L into a list of rectangles by letting each rectangle have height $H = r + \varepsilon$, where ε is sufficiently small, we get $\text{FFS}_r(L) \geqq 1.7n + r + \varepsilon - 9$ and $\text{OPT}(L) = n(r + \varepsilon)$. Thus, for every $\alpha < 1.7/r$ and every constant β, there are sufficiently large n and sufficiently small ε that $\text{FFS}_r(L) > \alpha\ \text{OPT}(L) + \beta H$. □

Note again that if $H = r^i$ for some integer i, the tallest shelf has height at most H and the additive constant in Theorem 2 becomes $H/(1-r)$ rather than $H/r(1-r)$.

COROLLARY 2.1. *For every constant r, $0 < r < 1$, and every list L of rectangles,*

$$\text{FFS}_r(L) < \left[\frac{1.7}{r} + \frac{1}{r(1-r)}\right] \text{OPT}(L).$$

Moreover this bound is tight.

Proof. Since $H \leqq \text{OPT}(L)$,

$$\text{FFS}_r(L) < \frac{1.7}{r}\,\text{OPT}(L) + \frac{H}{r(1-r)} \leqq \left[\frac{1.7}{r} + \frac{1}{r(1-r)}\right] \text{OPT}(L)$$

by Theorem 2. We know from [11] that for every r and every positive integer n, there exists a list L of rectangles, all of the same height d, such that the number of shelves used by FFS_r to pack L is at least $1.7n - 8$ while an optimal packing is no worse than placing the pieces in n rows each of height d. Let k and t be positive integers, $k < t$,

and let $\varepsilon < r^{t-1} - r^t$. Let $n = \lceil r^{1-t} \rceil$. If we take $d = r^t + \varepsilon$, we have

$$\text{FFS}_r(L) \geqq (1.7n - 9)r^{t-1} + r^t + \varepsilon, \qquad \text{OPT}(L) = n(r^t + \varepsilon).$$

Let P_1 and P_2 be the FFS_r packing of L and an optimal packing of L, respectively. Let m be the maximum number of rectangles which appear in any single shelf in P_1. For some $x > 0$, the width of the empty area at the bottom right end of each shelf is smaller by at least x than every piece packed higher. Choose Δ so that $\Delta < x/(m+1)k$. Form a list L' from L by shrinking the width of each piece by $k\Delta$. If we replace each piece in P_1 by its corresponding piece in L', the total extra space in each shelf has width at most $mk\Delta < x$, so that no piece packed in a higher shelf will fit into a lower shelf. Therefore, $\text{FFS}_r(L')$ uses the same number of shelves as $\text{FFS}_r(L)$. In P_2, if we replace each piece by its corresponding piece in L' and push each piece as far left as possible, we get an empty region of width $k\Delta$ at the right end of the bin.

Now, let $L'' = (s_1, s_2, \cdots, s_k)$ where s_i has height $r^i + \varepsilon$ and width Δ. Let L_0 be a list consisting of the pieces of L' followed by the pieces of L''.

An optimal packing of L_0 is no worse than packing L'' into the bin using width $1 - k\Delta$ and height $n(r^t + \varepsilon)$, and then packing $s_1, s_2, \cdots, s_k$ left to right into the remaining space of width $k\Delta$. Thus, $\text{OPT}(L_0) \leqq n(r^t + \varepsilon) = \lceil r^{1-t} \rceil (r^t + \varepsilon) \leqq (r^{1-t} + 1)(r^t + \varepsilon) \leqq r + r^t + r^{1-t}\varepsilon + \varepsilon$.

Now, FFS_r packs each s_i alone in a shelf of height r^{i-1}. These shelves use a total height of

$$\sum_{i=1}^{k} r^{i-1} = \frac{1}{1-r} - \frac{r^k}{1-r}.$$

The remaining shelves use a total height of at least $(1.7n - 9)r^{t-1} + r^t + \varepsilon$. Thus,

$$\begin{aligned}
\text{FFS}_r(L_0) &\geqq \frac{1}{1-r} - \frac{r^k}{1-r} + (1.7n - 9)r^{t-1} + r^t + \varepsilon \\
&\geqq \frac{1}{1-r} - \frac{r^k}{1-r} + (1.7r^{1-t} - 9)r^{t-1} \\
&\geqq \frac{1}{1-r} - \frac{r^k}{1-r} + 1.7 - 9r^{t-1}.
\end{aligned}$$

For any $\delta > 0$ and sufficiently small ε and sufficiently large k and t, we obtain $\text{FFS}_r(L_0) > [1.7/r + 1/r(1-r) - \delta]\,\text{OPT}(L_0)$. □

We have seen that the asymptotic performance of FFS_r can be made as close as desired to 1.7, which is the asymptotic bound for FFDH [7], without the FFDH restriction of packing pieces in order of decreasing height. For the absolute worst case performance, we can find the minimal value of the bound of Corollary 2.1 by differentiating to get the following Corollary.

COROLLARY 2.2. *The value of* r, $0 < r < 1$, *for which the absolute worst case bound of* FFS_r *is minimal is* $r \approx .622$. *At this value of* r, *for every list* L, $\text{FFS}_r(L) \approx 6.99\,\text{OPT}(L)$.

4. Worst case bounds for lists of squares. Many two-dimensional packing algorithms have been found to have lower worst case bounds for lists of squares than for arbitrary lists of rectangles. For absolute worst case bounds, this pattern also holds for FFS_r and NFS_r.

THEOREM 3. *Let* $0 < r < 1$. *For any list* L *of squares,* $\text{NFS}_r(L) \leqq [1/r + 1/r(1-r)]\,\text{OPT}(L)$ *and* $\text{FFS}_r(L) \leqq [1/r + 1/r(1-r)]\,\text{OPT}(L)$. *Moreover, these bounds are tight.*

Proof. Without loss of generality, let the bin width be 1. Let P be an NFS_r or FFS_r packing of a list L of squares, and let $|P|$ denote the height used by P. For $k \geqq 0$, let H_k denote the total height of shelves of height r^k and let A_k denote the total area of squares packed in shelves of height r^k. Suppose the height of the highest shelf is r^p. Define $h = r^p$. Let $A = \sum_k A_k$ be the total area of squares in L.

First, note that if $r^{k+1} > 1/2$, then every shelf of height r^k contains a square of size at least $r^{k+1} > 1/2$ and $A_k \geqq (r/2)H_k$.

Next, we claim that if there are $j > 1$ shelves of height r^k, then $A_k \geqq (r/2)H_k$. Note that the total width of squares packed in any two successive shelves of height r^k is at least 1 and each square on such a shelf has height at least r^{k+1}. Consequently, if j is even, $A_k \geqq (r/2)H_k$. So suppose j is odd. Pairing the first $j-3$ shelves of height r^k shows that the total area of squares in them is at least $(r/2)(H_k - 3r^k)$. If shelf $j-2$ is filled to width at least $1/2$, then this shelf plus the last pair of shelves contain squares of area at least $(r/2)(3r^k)$ and $A_k \geqq (r/2)H_k$. So suppose shelf $j-2$ is filled to width less than $1/2$. Then $r^{k+1} < 1/2$, shelf $j-1$ contains a square of size at least $1/2 + \delta$, for some $\delta > 0$, and the total width of squares in shelves $j-1$ and j is at least 1. The total area of squares in these three shelves is at least

$$\begin{aligned}(\tfrac{1}{2}-\delta)r^{k+1} + (\tfrac{1}{2}+\delta)^2 + (1-(\tfrac{1}{2}+\delta))r^{k+1} &\geqq (\tfrac{1}{2}-\delta)r^{k+1} + (\tfrac{1}{2}+\delta)r^{k+1} + (\tfrac{1}{2}+\delta)\delta + (\tfrac{1}{2}-\delta)r^{k+1} \\ &\geqq (\tfrac{1}{2}-\delta)r^{k+1} + (r^{k+1}+\delta)\delta + r^{k+1} \\ &\geqq \tfrac{3}{2}r^{k+1}.\end{aligned}$$

Again, we have $A_k \geqq (r/2)H_k$.

Suppose that either there are $m > 1$ shelves of height $h = r^p$ or $rh > \frac{1}{2}$. From above, $A_p \geqq (r/2)H_p$. By pairing the shelves of other heights, we see that for $k > p$, $A_k \geqq (r/2)(H_k - r^k)$. Recall that the sums of the heights of the sparse and dense shelves are denoted by H_S and H_D, respectively. Consequently,

$$A = \sum_{k \geqq p} A_k \geqq \frac{r}{2} \sum_{k \geqq p} (H_k - r^k) + \frac{r}{2} h \geqq \frac{r}{2}(H_D + h).$$

Since $H_S \leqq \sum_{i=0}^{\infty} r^i h = h/(1-r)$, we have

$$\begin{aligned}|P| = H_S + H_D \leqq \frac{h}{1-r} + \frac{2A}{r} - h &= \frac{rh}{1-r} + \frac{2A}{r} \\ &\leqq \left[\frac{1}{1-r} + \frac{2}{r}\right] \text{OPT}\,(L) \\ &= \left[\frac{1}{r} + \frac{1}{r(1-r)}\right] \text{OPT}\,(L),\end{aligned}$$

which is the desired result.

So we may assume that there is only one shelf of height h and $rh < 1/2$. Define

$$\alpha = \frac{r(1-r)H_S}{\text{OPT}\,(L)}.$$

Since $H_S \leqq h/(1-r) \leqq \text{OPT}\,(L)/r(1-r)$, we know that $0 < \alpha \leqq 1$.

Note that each dense shelf has height at most rh and hence is filled to width at least $1 - rh$. Moreover, the (sparse) shelf of height h contains a square of size at least

rh. Consequently,

$$\begin{aligned}\mathrm{OPT}(L) \geqq A &\geqq r(1-rh)H_D + (rh)^2 \\ &\geqq r(1-rh)H_D + r^2h(1-r)H_S \\ &\geqq r(1-rh)H_D + rh\alpha\ \mathrm{OPT}(L),\end{aligned}$$

and

$$\frac{H_D}{\mathrm{OPT}(L)} \leqq \frac{1-\alpha rh}{r(1-rh)}.$$

Then

$$\begin{aligned}\frac{|P|}{\mathrm{OPT}(L)} = \frac{H_D + H_S}{\mathrm{OPT}(L)} &\leqq \frac{1-\alpha rh}{r(1-rh)} + \frac{\alpha}{r(1-r)} \\ &= \frac{1}{r(1-rh)} + \alpha\left[\frac{1}{r(1-r)} - \frac{h}{1-rh}\right].\end{aligned}$$

Since $rh < 1/2$,

$$\frac{1}{r(1-r)} - \frac{h}{1-rh} = \frac{1-2rh+r^2h}{r(1-r)(1-rh)} > 0,$$

and from $0 \leqq \alpha \leqq 1$, we have

$$\begin{aligned}\frac{|P|}{\mathrm{OPT}(L)} = \frac{H_S + H_D}{\mathrm{OPT}(L)} &\leqq \frac{1}{r(1-rh)} + \frac{1}{r(1-r)} - \frac{h}{1-rh} \\ &= \frac{1}{r} + \frac{1}{r(1-r)}.\end{aligned}$$

It remains to show that the bound of $1/r + 1/r(1-r)$ is tight. Let t and u be positive integers, and let k be a positive integer such that $r^k < 1-r$. Let $0 < \varepsilon \ll 1$. The list L consists of two groups of squares. The first group has t squares $s_1, \cdots, s_t$, where s_i has size $r^{k+i-1} + \varepsilon$. The second group contains $\lceil r^{-k-2t-2u} \rceil$ squares of size $r^{k+t+u} + \varepsilon$.

Let P be an NFS_r or FFS_r packing of L. Each square s_i of the first group is packed into a shelf of height r^{k+i-2} in P. Let $s = \lfloor 1/(r^{k+t+u} + \varepsilon) \rfloor$. Exactly s squares of the second group are packed into each shelf of height $r^{k+t+u-1}$. Let $y = \lceil \lceil r^{-k-2t-2u} \rceil / s \rceil$. Note that if ε is sufficiently small,

$$r^{-k-t-u} - 2 \leqq s < r^{-k-t-u}$$

and

$$r^{-t-u} \leqq y \leqq \frac{r^{-k-2t-2u} + 1}{r^{-k-t-u} - 2} + 1.$$

Let $|P|$ denote the height of P. Since the top square in P has its top above the bottom of the top shelf,

$$\begin{aligned}|P| \geqq \sum_{i=1}^{t} r^{k+i-2} + (y-1)(r^{k+t+u-1}) &= r^{k-1} \sum_{i=0}^{t-1} r^i + (y-1)(r^{k+t+u-1}) \\ &= \frac{r^{k-1}}{1-r} - \frac{r^{k+t-1}}{1-r} + (y-1)(r^{k+t+u-1}).\end{aligned}$$

An optimal packing of L is no worse than the following. By choice of k and sufficiently small ε, the squares of the first group can be packed in a row of width $\leqq r^k/(1-r)$ across the bottom of the bin. The squares of the second group can be packed in rows wherever they fit to the right and above the squares of the first group. For any $\delta > 0$ it is possible to pick k and u large enough to make the total width and area of the first group small enough and the squares of the second group small enough for the total height of the packing to be at most $y(r^{k+t+u}+\varepsilon)+\delta$. Thus, we have

$$\frac{|P|}{\mathrm{OPT}(L)} \geqq \frac{\dfrac{r^{k-1}}{1-r} - \dfrac{r^{k+t-1}}{1-r} + (y-1)(r^{k+t+u-1})}{y(r^{k+t+u}+\varepsilon)+\delta}$$

$$\geqq \frac{\dfrac{r^k}{r(1-r)} - \dfrac{r^{k+t-1}}{1-r}}{\left(\dfrac{r^{-k-2t-2u}+1}{r^{-k-t-u}-2}+1\right)(r^{k+t+u}+\varepsilon)+\delta} + \frac{(y-1)(r^{k+t+u-1})}{y(r^{k+t+u}+\varepsilon)+\delta}.$$

For any $n > 0$ and $\delta' > 0$, it is possible to pick sufficiently small δ and ε and sufficiently large k, t and u such that $\mathrm{OPT}(L) > n$ and

$$\frac{|P|}{\mathrm{OPT}(L)} \geqq \frac{1}{r(1-r)} + \frac{1}{r} - \delta'.$$

Therefore, the asymptotic bound of $1/r(1-r)+1/r$ is tight. □

COROLLARY 3.1. *The value of* r, $0 < r < 1$, *for which the absolute worst case performance bound of* NFS_r *and* FFS_r *is minimal on lists of squares is* $2+\sqrt{2}$. *At this value of* r, NFS_r, $\mathrm{FFS}_r \approx 5.83\ \mathrm{OPT}(L)$ *for every list* L *of squares.*

The above theorem shows that the absolute worst case performance bounds of NFS_r and FFS_r improve from $2/r+1/r(1-r)$ or $1.7/r+1/r(1-r)$, respectively, to $1/r+1/r(1-r)$ for lists of squares. We do not have tight asymptotic worst case bounds of FFS_r and NFS_r on lists of squares. However, we will prove a number of lower bound results that are summarized in Tables 1 and 2. These results suggest that the asymptotic bounds are not in general much better for lists of squares than for arbitrary lists of rectangles.

TABLE 1

Lower bounds on the asymptotic worst case bound of NFS_r *on lists of squares for various values of* r. *Note that the asymptotic worst case bound for* NFS_r *on arbitrary lists of rectangles is* $2/r$.

r	result	squares LB
$r > 1/2$, root of $1/2$	Theorems 4, 6	$1.6/r$, 1.7
$r > 1/2$, not a root of $1/2$	Theorem 5	2.0
$r = 1/2$	Theorem 4	3.2
$r < 1/2$	Proposition 1	$1/r > 2$
root of $1/k$	Theorem 4	$(k+1)/kr$

TABLE 2

Lower bounds on the asymptotic worst case bound of FFS_r *on lists of squares for various values of* r. *Note that the asymptotic worst case bound for* FFS_r *on arbitrary lists of rectangles is* $1.7/r$.

r	result	squares LB
root of $1/3$ or $1/6$	Theorem 6	1.69
not a root of $1/3$ or $1/6$	Theorem 6	1.7
root of $1/2$	Theorem 4	$1.6/r$
root of $1/3$	Theorem 4	$1.36/r$
root of $1/k$	Theorem 4	$(k+1)/k$
not a root of $1/k$, for any integer $k>0$	Proposition 1	$1/r$

THEOREM 4. *Let* k *be a positive integer, and let* r *be a positive root of* $1/k$. *Let* α *and* β *be constants such that for every list* L *of squares of size at most* H

$$\mathrm{FFS}_r(L) \leqq \alpha\ \mathrm{OPT}(L) + \beta H$$

or for every list L *of squares of size at most* H

$$\mathrm{NFS}_r(L) \leqq \alpha\ \mathrm{OPT}(L) + \beta H.$$

Then

$$\alpha \geqq \frac{1}{r}\sum_{i=1}^{\infty}\frac{k-1}{k^i-1} > \frac{k+1}{kr}.$$

Proof. Let $0<\varepsilon \ll 1$. For integers j, $n>0$, the list L_{nj} will contain $(k-1)k^i n$ squares of size $1/k^i+\varepsilon$ for $1 \leqq i \leqq j$. If ε is sufficiently small, NFS_r or FFS_r will pack the squares of size $1/k^i+\varepsilon$ into $\lceil (k-1)k^i n/(k^i-1)\rceil$ shelves of height $1/k^i r$. Thus, the total height $|P|$ of the packing P is at least

$$\frac{1}{r}\sum_{i=1}^{j}\left\lceil\frac{(k-1)k^i n}{k^i-1}\right\rceil\frac{1}{k^i} - \frac{1}{k_j r} + \frac{1}{k^j} + \varepsilon \geqq \frac{n}{r}\sum_{i=1}^{j-1}\frac{1}{k^i-1}.$$

For sufficiently small ε, $(k-1)\sum_{i=1}^{j}(1/k^i+\varepsilon)<1$, and an optimal packing of L_{nj} is no worse than the following. For $1 \leqq i \leqq j$, pack the $(k-1)k^i n$ squares of size $1/k^i+\varepsilon$ in $k-1$ columns of equal height, with all of the $j(k-1)$ columns side by side. The total height of the packing is at most

$$\max_{1\leqq i\leqq j} k^i n\left(\frac{1}{k^i}+\varepsilon\right) = n + k^j n\varepsilon.$$

Thus,

$$\frac{|P|}{\mathrm{OPT}(L)} \geqq \frac{\dfrac{n}{r}\displaystyle\sum_{i=1}^{k}\frac{k-1}{k^i-1}}{n+k^j n\varepsilon}.$$

For any n and any $\delta > 0$, j can be chosen sufficiently large and ε can be chosen sufficiently small such that

$$\frac{|P|}{\mathrm{OPT}\,(L)} \geqq \frac{1}{r} \sum_{i=1}^{\infty} \frac{k-1}{k^i - 1} - \delta.$$

Therefore,

$$\alpha \geqq \frac{1}{r} \sum_{i=1}^{\infty} \frac{k-1}{k^i - 1}.$$

Letting

$$f_1(k) = \sum_{i=0}^{\infty} \frac{1}{k^i}$$

and

$$f_2(k) = \sum_{i=1}^{\infty} \frac{k-1}{k^i - 1} = \sum_{i=0}^{\infty} \frac{1}{k^i + k^{i-1} + \cdots + k + 1},$$

we see that

$$f_1(k) - f_2(k) = \sum_{i=1}^{\infty} \frac{k^{i-1} + k^{i-2} + \cdots + 1}{k^i (k^i + \cdots + k + 1)} < \sum_{i=1}^{\infty} \frac{1}{k^{i+1}} = \frac{1}{k^2} f_1(k).$$

Consequently,

$$f_2(k) > \left[\frac{k^2 - 1}{k^2}\right] f_1(k) = \left[\frac{k^2 - 1}{k^2}\right]\left[\frac{k}{k-1}\right] = \frac{k+1}{k},$$

and

$$\alpha \geqq \frac{1}{r} f_2(k) > \frac{k+1}{kr}.$$ □

For $k = 1/2$, $1/3$ and $1/4$,

$$\frac{1}{r} \sum_{i=1}^{\infty} \frac{k-1}{k^i - 1}$$

evaluates to more than $1.6/r$, $1.36/r$ and $1.26/r$, respectively. Since $1.6/r$ is very close to $1.7/r$, which is the asymptotic worst case bound of FFS_r on rectangles, it appears that FFS_r does not perform significantly better in the worst case for squares than for rectangles.

For values of r that are not roots of $1/k$ for some integer k, the following proposition gives a simple lower bound.

PROPOSITION 1. *Let $0 < r < 1$, and let α and β be constants such that for every list L of squares of size at most H,*

$$\mathrm{FFS}_r\,(L) \leqq \alpha\ \mathrm{OPT}\,(L) + \beta H$$

or for every list L of squares of size at most H

$$\mathrm{NFS}_r\,(L) \leqq \alpha\ \mathrm{OPT}\,(L) + \beta H.$$

Then $\alpha \geqq 1/r$.

Proof. For $\varepsilon > 0$ and any integer $n > 0$, consider a list L of $n\lfloor 1/(r+\varepsilon)\rfloor$ squares of size $r+\varepsilon$. An NFS_r or FFS_r packing P uses height at least $n-1+r+\varepsilon$, while an optimal packing uses height at most $n(r+\varepsilon)$. Thus, for any $\delta > 0$ and for sufficiently large n and sufficiently small ε,

$$\frac{|P|}{\mathrm{OPT}(L)} \geqq \frac{n-1+r+\varepsilon}{n(r+\varepsilon)} > \frac{1}{r} - \delta. \qquad \square$$

The next two theorems are different from the last two results in that the lower bounds are constants rather than constants divided by r. Tables 1 and 2 show that these theorems are weaker for some values of r than the above, but stronger for other values of r.

THEOREM 5. *Let $0 < r < 1$, where r is not a root of $1/2$. Let α and β be constants such that for any list L of squares of size at most H,*

$$\mathrm{NFS}_r(L) \leqq \alpha\ \mathrm{OPT}(L) + \beta H.$$

Then $\alpha \geqq 2$.

Proof. Let $0 < \varepsilon \ll 1$. Let L_n consist of $2n$ squares $s_1, s_2, \cdots, s_{2n}$ such that for $j \geqq 1$, s_{2j-1} has size $1/2 - j\varepsilon$ and s_{2j} has size $1/2 + (j+2)\varepsilon$. For sufficiently small ε, all of these squares will be packed in shelves of the same height h. Note that $s_{2j-1} + s_{2j} = 1 - j\varepsilon + (j+2)\varepsilon > 1$ and $s_{2j} + s_{2j+1} = s_{2j} + s_{2(j+1)-1} = 1/2 + (j+2)\varepsilon + 1/2 - (j+1)\varepsilon > 1$. Therefore, NFS_r packs each square alone in a shelf of height $h > 1/2$ and $\mathrm{NFS}_r(L) \geqq n$.

An optimal packing is no worse than the following. Pack s_1 and s_3 next to each other, then for $1 \leqq j < n-2$ pack the pairs s_{2j} and s_{2j+3} next to each other and finally pack s_{2n-2} and s_{2n} on top. This packing uses a height of $(n+1)/2 + O(n\varepsilon)$.

For any $\delta > 0$, sufficiently large n and sufficiently small ε can be selected such that

$$\frac{\mathrm{NFS}_r}{\mathrm{OPT}(L)} > \frac{n/r}{\frac{(n+1)}{2} + O(n\varepsilon)} > 2 - \delta. \qquad \square$$

The next result shows that the asymptotic bounds for NFS_r and FFS_r for lists of squares are at least 1.6 for all r and at least 1.7 for values of r other than roots of $1/3$ or $1/6$. Note that 1.7 is the limit of the asymptotic bound $1.7/r$ of FFS_r as $r \to 1$.

Define

$$t_1 = 2, \quad t_2 = 3, \quad t_{i+1} = t_i(t_i - 1) + 1 \quad \text{for } i \geqq 2$$

and

$$a_i = t_i - 1 \quad \text{for } i \geqq 1.$$

Define

$$\gamma = \sum_{i=1}^{\infty} \frac{1}{a_i}.$$

The above sequence has been studied in [3], [6], [11] and is closely related to sequences studied in [1].

THEOREM 6. *Let $0 < r < 1$, and let α and β be such that for every list L of squares of size at most H,*

$$\mathrm{FFS}_r(L) \leqq \alpha\ \mathrm{OPT}(L) + \beta H$$

or for every list L of squares of size at most H,

$$\mathrm{NFS}_r(L) \leqq \alpha\ \mathrm{OPT}(L) + \beta H.$$

If r is not a root of $1/3$ *or* $1/6$, *then* $\alpha \geqq 1.7$. *If r is a root of* $1/3$ *or* $1/6$, *then* $\alpha \geqq \gamma > 1.69$.

Proof. First, we show that for every r, $0 < r < 1$, $\alpha > \gamma$. Then we show that if r is not a root of $1/3$ or $1/6$, $\alpha \geqq 1.7$.

First, suppose $r < 2/3$. Let $0 < \varepsilon \ll 1$. For $n \geqq 1$, let L_n be a list containing n squares of size $2/3$ and $\lfloor 1/3\varepsilon \rfloor \lfloor 2n/3\varepsilon \rfloor$ squares of size ε. Note that $\mathrm{OPT}(L_n) = 2n/3$. Since FFS_r and NFS_r pack the large squares into shelves of height 1 and the small squares into shelves of total height at least $\lfloor 1/3\varepsilon \rfloor \lfloor 2n/3\varepsilon \rfloor \varepsilon^2 \geqq (1/3\varepsilon - 1)(2n/3\varepsilon - 1)\varepsilon^2 = 2n/9 - O(\varepsilon) + O(\varepsilon^2)$,

$$\mathrm{FFS}_r(L_n) = \mathrm{NFS}_r(L_n) > \tfrac{11}{9}n - O(\varepsilon) - O(\varepsilon^2).$$

Thus, for any δ, ε may be packed sufficiently small such that

$$\frac{\mathrm{FFS}_r(L_n)}{\mathrm{OPT}(L_n)} = \frac{\mathrm{NFS}_r(L_n)}{\mathrm{OPT}(L_n)} > \left(\frac{11}{9} \cdot \frac{3}{2}\right) - \delta > \gamma - \delta.$$

Since this is true for every $n > 0$, FFS_r and NFS_r cannot have an asymptotic bound $< \gamma$.

Now, suppose $r \geqq 2/3$. Let $\varepsilon \ll 1$. For n, $s \geqq 1$, define $L_{n,s}$ to be a list containing nt_i squares of size $1/t_i + \varepsilon/i$, $1 \leqq i \leqq s$. It may be easily verified that $\sum_{i=1}^{\infty} 1/t_i = 1$. Thus, for sufficiently small ε, $L_{n,s}$ size $1/t_i + \varepsilon/i$, $1 \leqq i \leqq s$. It may be easily verified that $\sum_{i=1}^{\infty} 1/t_i = 1$. Thus, for sufficiently small ε, $L_{n,s}$ can be packed into s columns, such that the ith column contains the squares of size $1/t_i + \varepsilon/i$. Thus, $\mathrm{OPT}(L_{n,s}) \leqq n + nt_s/s\varepsilon$.

Now, consider a NFS_r or FFS_r packing of $L_{n,s}$. For $i \geqq 1$ and sufficiently small ε,

$$\frac{\dfrac{1}{t_{i+1}} + \dfrac{\varepsilon}{i+1}}{\dfrac{1}{t_i} + \dfrac{\varepsilon}{i}} < \frac{\dfrac{1}{t_2} + \dfrac{\varepsilon}{2}}{\dfrac{1}{t_1} + \varepsilon} = \frac{\dfrac{1}{3} + \dfrac{\varepsilon}{2}}{\dfrac{1}{2} + \varepsilon} < \frac{2}{3}.$$

Therefore, squares of distinct sizes are packed into shelves of distinct heights. Since only $t_i - 1$ squares of size $1/t_i + \varepsilon/i$ can fit into a shelf,

$$\begin{aligned}\mathrm{NFS}_r(L_{n,s}) = \mathrm{FFS}_r(L_{n,s}) &\geqq \sum_{i=1}^{s} \frac{nt_i}{t_i - 1}\left(\frac{1}{t_i} + \frac{\varepsilon}{i}\right) \\ &= n \sum_{i=1}^{s} \frac{1}{t_i - 1} + n \sum_{i=1}^{s} \frac{t_i \varepsilon}{i(t_i - 1)} > n \sum_{i=1}^{s} \frac{1}{a_i}.\end{aligned}$$

Consequently, for any $\delta > 0$, there are a sufficiently small ε and sufficiently large s such that

$$\frac{\mathrm{NFS}_r(L)}{\mathrm{OPT}(L)} = \frac{\mathrm{FFS}_r(L)}{\mathrm{OPT}(L)} > \frac{n \sum_{i=1}^{s} \dfrac{1}{t_i}}{n(1+\varepsilon)} > \gamma - \delta.$$

Since this is true for any n, FFS_r and NFS_r cannot have an asymptotic bound smaller than γ.

Now, consider any r which is not a root of $1/3$ or $1/6$. We modify the list used in the proof that the asymptotic bound for FF is at least 1.7 [11]. Let n be a positive integer divisible by 17 and let $0 < \delta \ll 18^{-6n/17}$. For $1 \leqq i \leqq 6n/17$, define $\delta_i = 18^{(6n/17)-i}\delta$. For $1 \leqq i \leqq 6n/17$, define

$$\begin{aligned} a_{0i} &= 1/6 + 33\delta_i, & a_{4i} &= 1/6 - 13\delta_i, \\ a_{1i} &= 1/6 - 3\delta_i, & a_{5i} &= 1/6 + 9\delta_i, \\ a_{2i} &= a_{3i} = 1/6 - 7\delta_i, & a_{6i} &= a_{7i} = a_{8i} = a_{9i} = 1/6 - 2\delta_i. \end{aligned}$$

The first part of L will consist of squares of size $a_{01}, a_{11}, \cdots, a_{91}, a_{02}, \cdots, a_{92}, \cdots, a_{0(6n/17)}, \cdots, a_{9(6n/17)}$. Note that for sufficiently small δ, FFS$_r$ will pack all of these squares into shelves of the same height, since r is not a root of $1/6$. Also, note that $\sum_{j=0}^{4} a_{ji} = 5/6+3\delta_i$ and $\sum_{j=5}^{9} a_{ji} = 5/6+\delta_i$. Since $5/6+\delta_i = 5/6+18\delta_{i+1}$, a_{jk} will not fit into a shelf containing $\sum_{j=0}^{4} a_{ji}$ or $\sum_{j=5}^{9} a_{ji}$ for $k>i$. Therefore, for $1 \leqq i \leqq 6n/17$, $a_{1i}, \cdots, a_{4i}$ are packed by FFS$_r$ into shelf $2i-1$ and $a_{5i}, \cdots, a_{9i}$ are packed into shelf $2i$. Thus, the total height used for these shelves is at least $2n/17$.

For $1 \leqq i \leqq 3n/17$, define

$$b_{0i} = 1/3+46\delta_{2i}, \qquad b_{4i} = 1/3+12\delta_{2i},$$

$$b_{1i} = 1/3-34\delta_{2i}, \qquad b_{5i} = 1/3-10\delta_{2i},$$

$$b_{2i} = b_{3l} = 1/3+6\delta_{2i}, \qquad b_{6i} = b_{7i} = b_{8i} = b_{9i} = 1/3+\delta_{2i}.$$

Note that for each i, $b_{0i}+b_{1i} = b_{2i}+b_{3i} = 2/3+12\delta_{2i}$ and $b_{4i}+b_{5i} = b_{6i}+b_{7i} = b_{8i}+b_{9i} = 2/3+2\delta_{2i}$. For $k>i$, $2/3+2\delta_{2i} > 2/3+36\delta_{2k}$ and b_{jk} cannot fit into a shelf already containing b_{ji}, $b_{(j+1)i}$ for j even. Also, b_{jk} cannot be packed into a shelf containing five a_{rs}'s, for $r, s \geqq 0$. Therefore, FFS$_r$ packs each group $b_{0i}, \cdots, b_{9i}$ into 5 successive bins, using a total height of at least $5(3n/17)(1/3) = 5n/17$.

The remainder of the list consists of $20n/17$ squares of size $1/2+\delta$. The shelves for these squares use height at least $10n/17$.

The total height used by the FFS$_r$ packing is thus at least $2n/17+5n/17+10n/17 = n$.

An optimal packing is no worse than the following. Pack the squares of size $1/2+\delta$ in a column at the left side of the bin. The remaining squares will be packed next to these n groups of three, with a pair of squares of size about $1/6$ packed one above the other next to a square of size about $1/3$. The groups of three are as follows:

(a) $a_{j(2i-1)}, a_{j(2i)}, b_{ji}$, $j = 2, 3, 4, 6, 7, 8, 9$, $1 \leqq i \leqq 3n/17$,
(b) $a_{5(2i-1)}, a_{5(2i)}, b_{5(i-1)}$, $1 < i \leqq 3n/17$,
(c) $a_{0(2i-1)}, a_{0(2i)}, b_{1(i-1)}$, $1 < i \leqq 3n/17$,
(d) $a_{1(2i-1)}, a_{1(2i)}, b_{0(i+1)}$, $1 \leqq i < 3n/17$,
(e) $a_{1(6n/17-1)}, a_{1(6n/17)}, b_{5(3n/17)}$.

Note that each triple requires height at most $1/3+66\delta_1$, and the above $7(3n/17)+3(3n/17-1)+1$ triples use height $\leqq (30n/17-2)(1/3+66\delta_1)$. This leaves a_{51}, a_{52}, a_{01}, a_{02}, $b_{1(3n/17)}$ and b_{01}. The $b_{1(3n/17)}$ and b_{01} can be packed above each other using height at most $2(1/3+46\delta_1)$. At this point, the right side of the packing uses height

$$\begin{aligned}(30n/17-2)(1/3+66\delta_1)+2(1/3+46\delta_1) &= 10n/17+(30n/17-2)(66\delta_1)+92\delta_1 \\ &= 10n/17+[(30n/17-2)66+92]\delta \cdot 18^{(6n/17)-1}\end{aligned}$$

which is greater than the height of the left side, which is $(20n/17)(1/2+\delta) = 10n/17+(20n/17)\delta$. The remaining 4 a_{ij}'s can be packed above everything else using height $1/6+33\delta_1$. Thus, the total height is at most $10n/17+1/6+(30n/17)66\delta \cdot 18^{(6n/17)-1}$.

Since FFS$_r$ $(L) \geqq n$ and δ may be picked to be as small as desired, the asymptotic bound cannot be less than $17/10$. □

5. Conclusions. Theorem 2 shows that the asymptotic performance of FFS$_r$ is no worse than $(1.7/r)$ OPT $(L)+H/r(1-r)$, where H is the height of the tallest piece. Thus, we can get asymptotic performance arbitrarily close to the 1.7 bound of FFDH

[7] without having to sort the rectangles first. Corollary 2.1 shows that the absolute worst case performance is worse than for FFDH (Golan [10] proved that NFD $(L) \leqq 3\ \mathrm{OPT}(L)$ and the same proof works for FFDH). Absolute worst case bounds are a good measure of performance only for small numbers of rectangles. For large numbers of rectangles, the asymptotic bounds are a better measure. The shelf algorithms are intended primarily for situations requiring packing large numbers of rectangles, and the absolute worst case performance is not a primary consideration.

The previous papers on two-dimensional packing problems have not required that rectangles be packed in the order given; the algorithms order the lists before packing them. Since the FFS_r algorithm can pack pieces in any order, it is natural to ask whether ordering the lists would improve its worst case bounds.

In one-dimensional bin packing, the first-fit algorithm has an asymptotic bound of 17/10 if, the pieces are packed in the order given, and 11/9 if they are packed in order of decreasing size [11]. Now, the decreasing size of the one-dimensional case corresponds to decreasing width in the two-dimensional case. The one-dimensional results suggest that packing rectangles by decreasing width using the FFS algorithm might obtain asymptotic bounds better than 17/10. Unfortunately, it is easily seen that even using decreasing width, the asymptotic bound of FFS_r cannot be better than 3/2 for any r (consider, for example, lists with n pieces of height 1 and width $\varepsilon/2$ and $n\lfloor 1/\varepsilon \rfloor$ pieces of height 1/2 and width ε). Since the up-down algorithm [2] has an asymptotic bound of 5/4, such an asymptotic bound would not be an improvement over existing algorithms which reorder the pieces before packing them.

Acknowledgment. The authors would like to thank D. S. Johnson for his comments and especially for generalizing their original version of Theorem 3.

REFERENCES

[1] A. V. Aho and N. J. A. Sloane, *Some doubly exponential sequences*, Fibonacci Quart., 11 (1973), pp. 429–437.

[2] B. S. Baker, D. J. Brown and H. P. Katseff, *A 5/4 algorithm for two-dimensional packing*, J. Algorithms, 2 (1981), pp. 348–368.

[3] B. S. Baker and E. G. Coffman, Jr., *A tight asymptotic bound for next-fit decreasing bin-packing*, SIAM J. Alg. Discr. Meth., 2 (1981), pp. 147–152.

[4] B. S. Baker and E. G. Coffman, Jr., *A two dimensional bin-packing model of preemptive, FIFO storage allocation*, J. Algorithms, to appear.

[5] B. S. Baker, E. G. Coffman, Jr. and R. L. Rivest, *Orthogonal packings in two dimensions*, SIAM J. Comput., 9 (1980), pp. 846–855.

[6] D. J. Brown, *A lower bound for on-line one-dimensional bin packing algorithms*, Tech. Report ACT-19 (1979), Coordinated Science Laboratory, University of Illinois, Urbana, Illinois.

[7] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson and R. E. Tarjan, *Performance bounds for level-oriented two-dimensional packing algorithms*, SIAM J. Comput. 9 (1980), pp. 808–826.

[8] M. R. Garey and R. L. Graham, *Bounds on multiprocessing scheduling with resource constraints*, SIAM J. Comput. 4 (1975), pp. 187–200.

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

[10] I. Golan, *Orthogonal oriented algorithms for packing in two dimensions*, draft, 1978.

[11] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey and R. L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM J. Comput., 3 (1974), pp. 299–326.

[12] F. M. Liang, *A lower bound for on-line bin packing*, Informat. Processing Lett., 10 (1980), pp. 76–79.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0008 $01.25/0

ON THE SELECTION OF TEST DATA FOR VECTOR-VALUED RECURSIVE SUBROUTINES*

JOHN H. ROWLAND† AND LESLIE E. SHADER‡

Abstract. This investigation deals with the selection of test data for vector-valued sequences which can be generated by a linear first order system of difference equations with rational coefficients of limited degree. Let $\mathcal{F}(\nu, k, l, m)$ be the class of ν-dimensional sequences y which satisfy an equation of the form $P(n)y(n+1)=A(n)y(n)+R(n)$ where A, P, and R are polynomial matrices of degree $\leqq k, l$, and m, respectively, and P is diagonal. It is shown that there is a finite sample which uniquely identifies members of $\mathcal{F}(\nu, k, l, m)$. A finite sample also exists for the case where y is considered to be a function of its initial value. If y satisfies two distinct equations of the above type, then there is a transformation under which certain coordinates must eventually be rational functions, or even polynomials if P is the identity.

Key words. testing, sampling, verification, difference equations

1. Introduction. This paper deals with the selection of test data for vector-valued sequences which can be generated by means of a linear first order system of difference equations with polynomial or rational coefficients of limited degree. The general philosophy is to assume that the desired program and the program to be tested both produce sequences which satisfy difference equations of the above type. It will be shown that a finite sample exists which uniquely identifies such sequences; hence any errors which do not violate the basic assumptions will be detected by testing on this sample. A more complete discussion of this philosophy as well as further background on the theory of testing can be found in [6], [5], [3], and [2].

The results presented here generalize those in [6] where it was shown that a finite sample can be used to identify sequences generated by a single difference equation with polynomial coefficients, or by a system with constant coefficients and polynomial forcing function. Our theory is also applicable to linear higher order equations with polynomial or rational coefficients because these equations can be converted to a first order system of equations [6].

2. Mathematical background. The symbol N will denote the nonnegative integers, $\mathcal{P}_k$ the polynomials of degree $\leqq k$, and C_ν complex ν-dimensional space. By an initial segment of N we will mean a finite consecutive set of integers starting with zero; the symbol $I(M)$ will denote the initial segment $\{0, 1, \cdots, M\}$.

The concept of a polynomial matrix (often called a λ-matrix in the literature) is central to this paper. A polynomial matrix A is a matrix whose entries are polynomials; that is,

$$A(n)=(a_{ij}(n)), \qquad i=1, 2, \cdots \nu_1, \quad j=1, 2, \cdots, \nu_2,$$

where each a_{ij} is a polynomial in n. For our purposes the coefficients of the polynomials will be from the field of complex numbers. The degree of A is the maximum degree of the a_{ij}'s and the rank is the size of the largest square submatrix whose determinant does not vanish identically.

Our polynomial matrices occur as coefficients in systems of difference equations. Consider the system

$$P(n)y(n+1)=A(n)y(n)+R(n), \tag{2.1}$$

* Received by the editors October 13, 1981, and in revised form August 5, 1982.

† Departments of Computer Science and Mathematics, University of Wyoming, Laramie, Wyoming 82071.

‡ Department of Mathematics, University of Wyoming, Laramie, Wyoming 82071.

where A is a ν by ν matrix of degree $\leqq k$, P is a ν by ν diagonal matrix of degree $\leqq l$, R is a ν by 1 matrix of degree $\leqq m$, and y is a complex-valued ν-dimensional sequence. In order to simplify various arguments we will consider changes of variable having the form

$$y(n) = V(n)w(n),$$

where V is a polynomial matrix whose inverse is also a polynomial matrix. Assuming P^{-1} exists, this leads to the equivalent system

$$w(n+1) = F(n)w(n) + T(n), \tag{2.2}$$

where

$$F(n) = V^{-1}(n+1)P^{-1}(n)A(n)V(n),$$
$$T(n) = V^{-1}(n+1)P^{-1}(n)R(n).$$

The polynomial matrices will be reduced to certain special forms (diagonal, triangular, etc.) in order to simplify the system (2.1). This is carried out by the Gauss reduction algorithm described in Gantmacher [1]. We will be interested in the simultaneous reduction of one matrix to diagonal form and another to lower Hessenberg form. (A polynomial matrix A is in lower Hessenberg form if $a_{ij} \equiv 0$ for $j > i+1$.) We will also want to determine the degree of the matrices involved in these reductions.

Gantmacher [1] describes three types of elementary row transformations:

(i) interchange of two rows,

(ii) multiplication of one row by a constant,

(iii) addition of a polynomial multiple of one row to another.

For each of these transformations Gantmacher displays a corresponding elementary polynomial matrix whose determinant is a nonzero constant and whose inverse is a polynomial matrix of the same type and degree. We will refer to a product of these elementary matrices as an elementary product matrix. Column operations are defined in a like manner with analogous elementary matrices used on the right instead of the left.

The fundamental operation in our reduction process will be to replace all entries except the first in one row of a matrix by zeros. This process is described in Gantmacher, but we must compute the degree of certain matrices involved in the reduction. Consider an arbitrary polynomial matrix B of degree $\leqq k$. By interchanging columns if necessary we may assume that b_{11} has the smallest degree of the nonzero entries of row 1. Let the elementary product matrix T_1 be defined by

$$T_1 = \left(\begin{array}{c|ccc} 1 & -q_{12} & \cdots & -q_{1\nu} \\ \hline 0 & & & \\ \vdots & & I & \\ 0 & & & \end{array}\right), \tag{2.3}$$

where q_{1j} and r_{1j} are determined by the division algorithm to satisfy $b_{1j} = q_{1j}b_{11} + r_{1j}$. The first row of BT_1 is $(b_{11}, r_{12}, \cdots, r_{1\nu})$. Repeated application of this process gives after at most $k+1$ steps a first row of the form $(d, 0, \cdots, 0)$, where d is the greatest common divisor of $b_{11}, \cdots, b_{1\nu}$. Let P_i be the permutation matrix which interchanges columns at the ith step or let $P_i = I$ if no interchange is needed. Also let $T_i = I$ for $i > i_0$ if the process terminates in i_0 steps. Let $V_1 = P_1T_1P_2T_2 \cdots P_{k+1}T_{k+1}$. Then BV_1 has the desired form. Let η_0 and η_i be the smallest degree of the nonzero elements in the first row of B and $BP_1T_1 \cdots P_iT_i$, respectively. We observe the following

concerning the degrees of the polynomials involved in this process:

(i) $\deg P_i = 0$, $i = 1, 2, \cdots, k+1$,

(ii) $\deg T_1 \leqq k - \eta_0$, $\deg T_{i+1} \leqq \eta_{i-1} - \eta_i$, $i = 1, 2, \cdots, k$,

(iii) $\deg V_1 \leqq k - \eta_k \leqq k$,

(iv) $\deg BV_1 \leqq 2k$.

Part (iii) follows from (ii) by observing that $\sum_{i=0}^{k} \deg T_{i+1}$ is bounded above by the telescopic sum

$$k - \eta_0 + \sum_{i=1}^{k} (\eta_{i-1} - \eta_i).$$

Using these facts we can establish the following lemma.

LEMMA 2.1. *Let A and C be ν by ν polynomial matrices of degree $\leqq k$ and suppose C has nonzero entries in only the first row. Let P be a diagonal ν by ν polynomial matrix of rank ν and degree $\leqq l$. Then there exists an elementary product matrix V and a polynomial d of degree $\leqq k$ such that*

$$C(n)V(n) = \operatorname{diag}(d(n), 0, \cdots, 0)$$

and

$$F(n) = V^{-1}(n+1)P^{-1}(n)A(n)V(n)$$

is in lower Hessenberg form (with rational function entries). Furthermore, each rational function f_{ij} has a representation of the form $f_{ij} = \alpha_{ij}/\beta_{ij}$, where α_{ij} and β_{ij} are polynomials satisfying the conditions

(i) $\deg \alpha_{ij} \leqq 3^{\nu-2}[3k + (\nu-1)l]$,

(ii) $\deg \beta_{ij} \leqq (\nu-1)l$,

(iii) $\deg \alpha_{i,i+1} \leqq 3^{i-1}[3k + (\nu-1)l]$.

Proof. Let V_1 be the matrix which produces zeros in the first row of C and hence reduces C to diagonal form. Then $CV_1 = \operatorname{diag}(d, 0, \cdots, 0)$ where $d = gcd(c_{11}, \cdots, c_{1\nu})$. Thus $\deg d \leqq k$. Let p_0 be the product of the diagonal elements of P and note that $p_0 P^{-1}$ is a polynomial matrix of degree $\leqq (\nu-1)l$. Let

$$A^*(n) = V_1^{-1}(n+1)p_0(n)P^{-1}(n)A(n)V_1(n).$$

Note that $\deg A^* \leqq 3k + (\nu-1)l$. Partition A^* in the form

$$A^* = \left(\begin{array}{c|c} \begin{matrix} a_{11}^* \\ \vdots \\ a_{\nu-1,1}^* \end{matrix} & A_1 \\ \hline a_{\nu,1}^* & a_{\nu 2}^* \quad \cdots \quad a_{\nu\nu}^* \end{array}\right).$$

Let T_2 be a matrix (analogous to T_1 in (2.3)) for which $A_1 T_2$ has zeros in the off-diagonal positions of the first row and define

$$V_2 = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & T_2 \end{array}\right).$$

Now $CV_1V_2 = CV_1$ because the first row of V_2 is $(1, 0, \cdots, 0)$ and $CV_1 = \operatorname{diag}(d, 0, \cdots, 0)$. Note that

$$V_2^{-1} = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & T_2^{-1} \end{array}\right).$$

Thus since the first row of $A^*(n)V_2(n)$ has Hessenberg form, so does $V_2^{-1}(n+1)A^*(n)V_2(n)$ because the first row of V_2^{-1} is $(1, 0, \cdots, 0)$. Next, form the partition

$$V_2^{-1}(n+1)A^*(n)V_2(n) = \left(\begin{array}{cc|c} \alpha_{11} & \alpha_{12} & 0 \\ \hline * & * & \\ \vdots & \vdots & A_2 \\ * & * & \\ \hline * & * & * \end{array}\right),$$

where α_{11} and α_{12} are defined by this equation and the *'s represent elements or submatrices whose values are not important to our discussion. Now continue the above procedure inductively for $\nu - 2$ steps. Let $V = V_1 V_2 \cdots V_{\nu-1}$. Then

$$V^{-1}(n+1)p_0(n)P^{-1}(n)A(n)V(n) = (\alpha_{ij}(n))$$

is in lower Hessenberg form and $CV = \operatorname{diag}(d, 0, \cdots, 0)$. But the scalar $1/p_0(n)$ commutes with any matrix and does not change any zero elements; so

$$F(n) = \frac{1}{p_0(n)} V^{-1}(n+1)p_0(n)P^{-1}(n)A(n)V(n)$$

is also in lower Hessenberg form. Define β_{ij} by

$$\beta_{ij} = p_0 \tag{2.4}$$

and note that $f_{ij} = \alpha_{ij}/\beta_{ij}$.

It remains to bound the degrees of the polynomials in question. Note that the degrees of A^*, V_2, and α_{12} are bounded by $3k + (\nu - 1)l$. Thus

$$\deg V_2^{-1}(n+1)A^*(n)V_2(n) \leqq 3[3k + (\nu - 1)l].$$

This implies that the degrees of V_3 and α_{23} are bounded above by $3[3k + (\nu - 1)l]$ and hence

$$\deg V_3^{-1}(n+1)V_2^{-1}(n+1)A^*(n)V_2(n)V_3(n) \leqq 3^2[3k + (\nu - 1)l].$$

Inequalities (i) and (iii) are then obtained by induction. Part (ii) follows from (2.4) and the proof is complete.

We will also need to bound the degree of the polynomials involved in reducing a matrix to lower triangular form.

LEMMA 2.2. *If C is a ν by ν polynomial matrix of degree $\leqq k$, then there exists an elementary product matrix V such that $H = CV$ is in lower triangular form. Furthermore,* $\deg h_{ij} \leqq 2^{i-1}k$ *and* $\deg H \leqq 2^{\nu-1}k$.

Proof. Apply the Gauss reduction procedure in a manner similar to that used in Lemma 2.1.

Finally, let us show that any equation of the form (2.1) can be converted to a homogeneous equation.

LEMMA 2.3. *Any equation of the form* (2.1) *can be converted to an equivalent homogeneous system involving $\nu + m + 1$ variables.*

Proof. The key here is the fact that the powers of n satisfy a homogeneous system of difference equations with constant coefficients. This system can be appended to (2.1) and each coordinate of R can be replaced by a linear combination of the powers

of n to obtain a homogeneous system. Specifically, let

$$y_i(n) = n^{i-\nu-1}, \qquad i = \nu+1, \cdots, \nu+m+1.$$

From the binomial theorem we see that

$$y_i(n+1) = \sum_{j=0}^{i-\nu-1} g_{ij} n^j = \sum_{j=0}^{m} g_{ij} y_{\nu+j+1}(n), \tag{2.5}$$

where

$$g_{ij} = \binom{i-\nu-1}{j}, \qquad i = \nu+1, \cdots, \nu+m+1, \quad j = 0, \cdots, i-\nu-1,$$

and $g_{ij} = 0$, $j = i-\nu, \cdots, m$.

Let the coefficients of R_i be represented by h_{ij} so that

$$R_i(n) = \sum_{j=0}^{m} h_{ij} n^j = \sum_{j=0}^{m} h_{ij} y_{\nu+1+j}(n), \tag{2.6}$$

$i = 1, \cdots, \nu$; $j = 0, \cdots, m$. Combining (2.1), (2.5), and (2.6) we obtain the homogeneous system

$$Q(n)y(n+1) = E(n)y(n),$$

where

$$E = \begin{pmatrix} A & H \\ 0 & G \end{pmatrix} \quad \text{and} \quad Q = \begin{pmatrix} P & 0 \\ 0 & I \end{pmatrix}.$$

The proof is completed by noting that E and Q are $\nu+m+1$ by $\nu+m+1$ matrices of degree $\leqq k$ and l, respectively.

3. Main results. In this section we will investigate linear first order systems of difference equations whose coefficients are rational functions of limited degree. Each equation in such a system can be multiplied by the least common denominator to clear fractions; so we will consider two systems of the form

$$P(n)y(n+1) = A(n)y(n) + R(n), \tag{3.1}$$

$$Q(n)z(n+1) = B(n)z(n) + S(n), \tag{3.2}$$

where P, Q are ν by ν diagonal polynomial matrices of degree $\leqq l$, A, B are ν by ν polynomial matrices of degree $\leqq k$, and R, S are ν by 1 polynomial matrices of degree $\leqq m$. Let $\mathcal{F}(\nu, k, l, m)$ represent the collection of all sequences which can be generated by (3.1) as the entries of A, P, and R vary over $\mathcal{P}_k$, $\mathcal{P}_l$, and $\mathcal{P}_m$, respectively. Our principal theorem states that if two sequences from $\mathcal{F}(\nu, k, l, m)$ agree on a sufficiently large initial segment and the diagonal matrices have no singularities on the nonnegative integers, then the two sequences agree forever. Let us first consider the uniqueness properties of solutions to (3.1).

THEOREM 3.1. *Let $t \in C_\nu$ and let n_0 be the first member of N for which $P(n_0)$ is singular (or $n_0 = \infty$ if P is never singular on N). The system* (3.1) *has a unique solution on $I(n_0)$ which satisfies the intial condition $y(0) = t$.*

Proof. For $n \in I(n_0 - 1)$ we can multiply both sides of (3.1) by $P^{-1}(n)$ to obtain the equivalent equation

$$y(n+1) = P^{-1}(n)[A(n)y(n) + R(n)]. \tag{3.3}$$

If y and z both satisfy (3.3) and $y(0)=z(0)=t$, then it follows inductively that $y(n)=z(n)$, $n=0, 1, \cdots, n_0$.

If $P(n_0)$ is singular, then (3.1) does not have a unique solution on $I(n_0+1)$. To see this consider the equation

$$P(n_0)y(n_0+1)=A(n_0)y(n_0)+R(n_0).$$

Since $P(n_0)$ is singular, this equation will either be inconsistent or there will be infinitely many values for $y(n_0+1)$ which satisfy the equation. Thus (3.1) has either no solution or infinitely many solutions on $I(n_0+1)$.

The system (3.1) can be converted to a homogeneous system by the method described in Lemma 2.3. Let us now consider two homogeneous systems,

$$P(n)y(n+1)=A(n)y(n), \tag{3.4}$$

$$Q(n)z(n+1)=B(n)z(n), \tag{3.5}$$

where A, B are ν by ν polynomial matrices of degree $\leqq k$, and P, Q are ν by ν diagonal polynomial matrices of degree $\leqq l$.

THEOREM 3.2. *Suppose y and z satisfy* (3.4) *and* (3.5), *respectively. Let*

$$M=(k+1)\nu+[3^{\nu}-2\nu-1][3k+(\nu-1)l]/4,$$

m_0, n_0 be the first members of N which are singularities of P and Q, respectively, and let N_0 be the initial segment $I(\min(m_0, n_0))$. If $y(n)=z(n)$, $n=0, 1, \cdots, M$, then $y(n)=z(n)$ for all $n \in N_0$.

Proof. Note that $PQ=QP$ since both P and Q are diagonal. Multiply (3.5) by P, (3.4) by Q, and subtract to obtain

$$C(n)y(n)=0, \qquad n=0, 1, \cdots, M-1, \tag{3.6}$$

where $C=PB-QA$. If C is identically zero, one can multiply (3.5) by P to obtain

$$P(n)Q(n)z(n+1)=P(n)B(n)z(n)=Q(n)A(n)z(n). \tag{3.7}$$

But Q^{-1} commutes with P, so multiplication of (3.7) by Q^{-1} yields the equivalent equation

$$P(n)z(n+1)=A(n)z(n).$$

Thus z satisfies (3.4) and the result follows from Theorem 3.1.

Now suppose C is not identically zero. Let us temporarily suppose that C has nonzero entries in only one row which for convenience we will take to be the first row. Let V be the elementary product matrix from Lemma 2.1 which converts C to diagonal form and A to lower Hessenberg form. Let $y=Vw$ and $z=Vx$. Now $V(n)$ is nonsingular for every n; so $w(n)=x(n)$ if and only if $y(n)=z(n)$. Note that w and x satisfy the equations

$$w(n+1)=F(n)w(n), \tag{3.8}$$

$$x(n+1)=G(n)x(n), \tag{3.9}$$

where $F(n)=V^{-1}(n+1)P^{-1}(n)A(n)V(n)$ and $G(n)=V^{-1}(n+1)Q^{-1}(n)B(n)V(n)$. Furthermore, we have from (3.6)

$$D(n)w(n)=0, \qquad n=0, 1, \cdots, M-1, \tag{3.10}$$

where $D=CV$ is diagonal and zero except in the first row; say $D=\operatorname{diag}(d, 0, \cdots, 0)$.

Now F is in lower Hessenberg form. Let $r=\nu$ if none of the elements $f_{i,i+1}$ immediately above the diagonal vanish identically; Otherwise, let r be the first index i for which $f_{i,i+1}$ is identically zero. Let $f_{ij}=\alpha_{ij}/\beta_{ij}$ as in Lemma 2.1 and define

$$\phi_0(n)=\prod_{j=0}^{r-1} d(n+j),$$

$$\phi_i(n)=\prod_{j=0}^{r-i-1} \alpha_{i,i+1}(n+j), \qquad i=1,2,\cdots,r-1,$$

$$\phi(n)=\prod_{i=0}^{r-1} \phi_i(n).$$

From Lemma 2.1 we see that

$$\deg \phi_0 \leqq kr, \qquad \deg \phi_i \leqq (r-i)3^{i-1}[3k+(\nu-1)l].$$

It follows that

$$\deg \phi \leqq kr+\sum_{i=1}^{r-1}(r-i)3^{i-1}[3k+(\nu-1)l].$$

Since $r\leqq\nu$, we can replace r by ν and use the formulas for the sum of a geometric progression and its derivative to obtain

$$\deg \phi \leqq k\nu+\frac{3^{\nu}-2\nu-1}{4}[3k+(\nu-1)l].$$

Note that $\phi\in\mathscr{P}_{M-\nu}$, so there is an index $n^*\in I(M-\nu)$ such that $\phi(n^*)\neq 0$. From (3.10) we have for all $n\in I(M-1)$,

$$d_1(n)w_1(n)=0.$$

But $n^*+r-1\leqq n^*+\nu-1\leqq M-1$ and $d_1(n^*+j)\neq 0, j=0,1,\cdots,r-1$. It follows that $w_1(n^*+j)=0$, $j=0,1,\cdots,r-1$. From the fact that $f_{r,r+1}\equiv 0$ and F is in lower Hessenberg form we have

$$w_i(n+1)=\sum_{j=1}^{\lambda(i)} f_{ij}(n)w_j(n), \qquad i=1,\cdots,r, \tag{3.11}$$

where $\lambda(i)=i+1$ for $i<r$ and $\lambda(r)=r$. In particular

$$w_1(n^*+j+1)=f_{11}(n^*+j)w_1(n^*+j)+f_{12}(n^*+j)w_2(n^*+j)$$

for $j=0,1,\cdots,r-2$. But $f_{12}(n^*+j)\neq 0$ and $w_1(n^*+j+1)=0=w_1(n^*+j)$; so $w_2(n^*+j)=0, j=0,1,\cdots,r-2$. Continuing in this fashion we infer that

$$w_i(n^*+j)=0, \quad j=0,1,\cdots,r-i, \quad i=1,2,\cdots,r.$$

In particular, $w_i(n^*)=0, i=1,2,\cdots,r$. It follows inductively from (3.11) that $w_i(n)=0, i=1,2,\cdots,r$ if $n^*\leqq n\leqq \min(m_0,n_0)$. Thus $D(n)w(n)=0$, $n^*\leqq n\leqq\min(m_0,n_0)$, and hence (using (3.10)) for all $n\in N_0$. Note that $G(n)-F(n)=V^{-1}(n+1)Q^{-1}(n)P^{-1}(n)D(n)$. Thus

$$[G(n)-F(n)]w(n)=0$$

for $n\in N_0$. If $x(n)=w(n)$, then

$$x(n+1)=G(n)x(n)=G(n)w(n)=F(n)w(n)=w(n+1).$$

It follows by induction that $x(n)=w(n)$ and hence $y(n)=z(n)$ for all $n\in N_0$.

Finally, we must remove the restriction that QA and PB differ in only one row. To this end let B_j be the matrix consisting of the first j rows of B and the last $\nu - j$ rows of A; and let Q_j consist of the first j rows of Q and the last $\nu - j$ rows of P. Let z^j be the solution to the equation

$$Q_j(n)z^j(n+1) = B_j(n)z^j(n), \qquad z^j(0) = z(0). \tag{3.12}$$

Now on $I(M-1)$ y satisfies both (3.4) and (3.5); hence it must satisfy (3.12) for any j. But Q_1A and PB_1 differ in only one row; so $y(n) = z^1(n)$ for all $n \in N_0$. Continuing in this fashion we see that

$$y(n) = z^1(n) = z^2(n) = \cdots = z^\nu(n) = z(n)$$

for all $n \in N_0$ and the proof is complete.

Next, we consider nonhomogeneous equations.

THEOREM 3.3. *Let*

$$M = (k+1)(\nu+m+1) + [3^{\nu+m+1} - 2(\nu+m) - 3][3k + (\nu+m)l]/4.$$

If $y, z \in \mathcal{F}(\nu, k, l, m)$ *and* $y(n) = z(n)$ *for* $n = 0, 1, \cdots, M$, *then* $y(n) = z(n)$ *for all* $n \in N_0$.

Proof. This follows from Theorem 3.2 and Lemma 2.3 after noting that the number of variables in the equivalent homogeneous system is $\nu + m + 1$.

The fact that the sample set starts with zero is unimportant, but the fact that it contains a consecutive set of integers is crucial. To see this consider the one-dimensional example:

$$y(n+1) = 1 - y(n), \qquad y(0) = 1,$$
$$z(n+1) = 1, \qquad z(0) = 1.$$

Note that $y(n) = z(n)$ when n is even, but $y(n) \neq z(n)$ when n is odd.

It was shown in [6] that a one-dimensional sequence y satisfies two different first order linear equations with polynomial coefficients if and only if y is eventually a polynomial. (That is, there exist a polynomial p and an index n^* such that $y(n) = p(n)$ for all $n \geqq n^*$.) One would not expect as strong a result for higher dimensional systems since the equations might be uncoupled and generate factorials and exponentials in some coordinates and polynomials in others. We will show that if the coefficients of the systems are rational, there is a change of variable under which the coordinates of the solution corresponding to equations which differ in the two systems are eventually rational functions. Furthermore, if the coefficients of one of the systems are polynomials, (that is, P or Q is the identity matrix) then after an appropriate change of variable these coordinates are eventually polynomials.

THEOREM 3.4. *Suppose* y *satisfies both the equations* (3.1) *and* (3.2) *for all* $n \in N$. *Let* $C = PB - QA$ *and let* r *be the rank of* C. *Then there is a transformation* $y = Vw$, *where* V *is an elementary product matrix, under which*

(i) $w_1, w_2, \cdots, w_r$ *are eventually rational functions with numerators and denominators of degree* $\leqq (2^i - 2)(k+l) + l + m$ *and* $(2^i - 1)(k+l)$, *respectively.*

(ii) *The equations for* $w_{r+1}, \cdots, w_\nu$ *can eventually be uncoupled from those for* $w_1, \cdots, w_r$, *and these equations are identical in both systems.*

Proof. According to Lemma 2.2 there exists an elementary product matrix V such that CV is in lower triangular form. Multiply (3.2) by P and (3.1) by Q and subtract to obtain $Cy = T$, where $T = QR - PS$. This leads to the equation

$$H(n)w(n) = T(n), \tag{3.13}$$

where $H = CV$ is in lower triangular form, has rank r and $\deg h_{ij} \leqq 2^{i-1}(k+l)$. The proof of part (i) is completed by solving (3.13) for $w_1, w_2, \cdots, w_r$.

To prove part (ii) note that w satisfies both the equations

$$w(n+1) = F(n)w(n) + U(n), \tag{3.14}$$

$$w(n+1) = G(n)w(n) + X(n), \tag{3.15}$$

where $F(n) = V^{-1}(n+1)P^{-1}(n)A(n)V(n)$, $U(n) = V^{-1}(n+1)P^{-1}(n)R(n)$, $G(n) = V^{-1}(n+1)Q^{-1}(n)B(n)V(n)$, and $X(n) = V^{-1}(n+1)Q^{-1}(n)S(n)$. Now CV has zeros in columns $r+1$ through ν; so

$$G(n) - F(n) = V^{-1}(n+1)P^{-1}(n)Q^{-1}(n)C(n)V(n)$$

has zeros in the same columns. Thus we can rewrite (3.14) and (3.15) in the form

$$w_i(n+1) = \sum_{j=1}^{r} f_{ij}(n)w_j(n) + \sum_{j=r+1}^{\nu} f_{ij}(n)w_j(n) + U_i(n), \tag{3.16}$$

$$w_i(n+1) = \sum_{j=1}^{r} g_{ij}(n)w_j(n) + \sum_{j=r+1}^{\nu} f_{ij}(n)w_j(n) + X_i(n), \tag{3.17}$$

$i = 1, 2, \cdots, \nu$. Note that (3.16) and (3.17) imply

$$\sum_{j=1}^{r} f_{ij}(n)w_j(n) + U_i(n) = \sum_{j=1}^{r} g_{ij}(n)w_j(n) + X_i(n),$$

$i = 1, 2, \cdots, \nu$. Furthermore, these quantities are eventually rational functions. Let $\phi_i(n)$ be the rational function which is eventually the same as $\sum_{j=1}^{r} f_{ij}(n)w_j(n) + U_i(n)$. Then we eventually have from (3.16) and (3.17)

$$w_i(n+1) = \sum_{j=r+1}^{\nu} f_{ij}(n)w_j(n) + \phi_i(n), \qquad r = r+1, \cdots, \nu$$

for both systems and the proof is complete.

We would like to show that $w_1, \cdots, w_r$ are eventually polynomials if one of the systems has polynomial coefficients. This will follow from the next lemma which shows that none of the coordinates of the solution to a polynomial system can be a proper rational function.

LEMMA 3.5. *Let y be a solution to the equation*

$$y(n+1) = A(n)y(n) + R(n) \tag{3.18}$$

for all $n \in N$. If one of the coordinates of y is eventually a rational function, then that coordinate is eventually a polynomial.

Proof. For convenience of notation assume that y_1 is eventually rational; say $y_1(n) = \phi_1(n)/\psi_1(n)$ where ϕ_1 and ψ_1 are polynomials. Let V be the elementary product matrix from Lemma 2.1 which converts A to lower Hessenberg form and leaves the first coordinate of y unchanged. Make the change of variable $y = Vw$; then eventually $w_1 = y_1 = \phi_1/\psi_1$. Now w satisfies the system

$$w(n+1) = F(n)w(n) + U(n), \tag{3.19}$$

where $F(n) = V^{-1}(n+1)A(n)V(n)$ and $U(n) = V^{-1}(n+1)R(n)$. As in the proof of Theorem 3.2, let r be the first index i for which $f_{i,i+1}$ vanishes identically or $r = \nu$ if $f_{i,i+1}$ is always nontrivial. Then we have

$$w_i(n+1) = \sum_{j=1}^{\lambda(i)} f_{ij}(n)w_j(n) + U_i(n), \qquad i = 1, 2, \cdots, r, \tag{3.20}$$

where $\lambda(i)=i+1$ for $i<r$ and $\lambda(r)=r$. Letting $i=1$ in (3.20) we see that w_2 is eventually rational, say $w_2=\phi_2/\psi_2$. Continuing in this fashion we see that $w_1, \cdots, w_r$ are all eventually rational, say $w_i=\phi_i/\psi_i$, $i=1, 2, \cdots, r$, where ϕ_i and ψ_i are polynomials.

Let ϕ_i^*/ψ_i^* represent the rational function ϕ_i/ψ_i after common factors, if any, have been removed. Eventually $w_i=\phi_i^*/\psi_i^*$, so (3.20) yields

$$\frac{\phi_i^*(n+1)}{\psi_i^*(n+1)}=\sum_{j=1}^{\lambda(i)} f_{ij}(n)\frac{\phi_i^*(n)}{\psi_i^*(n)}+U_i(n), \tag{3.21}$$

$i=1, 2, \cdots, r$. Now (3.21) holds for infinitely many integers; so it must hold in the entire complex plane except for the zeros of $\psi_1^*, \cdots, \psi_r^*$. We claim that $\psi_1^*, \cdots, \psi_r^*$ must be constants. To see this suppose the contrary and let ψ_μ^* be the denominator having the zero, say z_0, with smallest real part of all the zeros of $\psi_1^*, \cdots, \psi_r^*$. Define

$$\Pi(z)=\prod_{i=1}^{\lambda(\mu)} \psi_i^*(z), \qquad \Pi_j(z)=\prod_{\substack{i=1\\ i\neq j}}^{\lambda(\mu)} \psi_i^*(z).$$

Clearing fractions from the μth equation in (3.21) and setting $n=z_0-1$, we obtain

$$\begin{aligned}&\phi_\mu^*(z_0)\Pi(z_0-1)\\ &=\psi_\mu^*(z_0)\left[\sum_{j=1}^{\lambda(\mu)} f_{\mu j}(z_0-1)\Pi_j(z_0-1)\phi_j^*(z_0-1)+\Pi(z_0-1)U_\mu(z_0-1)\right].\end{aligned} \tag{3.22}$$

Now the right side of (3.22) vanishes, but $\phi_\mu^*(z_0)\neq 0$ because ϕ_μ^* and ψ_μ^* have no common factors, and $\Pi(z_0-1)\neq 0$ because z_0 has the smallest real part of all the zeros of $\psi_1^*, \cdots, \psi_r^*$. This contradiction shows that $\psi_1^*, \cdots, \psi_r^*$ are constants. It follows that $w_1, \cdots, w_r$ are eventually polynomials and (since $y_1=w_1$) the proof is complete.

COROLLARY 3.6. *Suppose the hypotheses of Theorem* 3.4 *are satisfied. If P or Q is the identity matrix, then* $w_1, \cdots, w_r$ *are eventually polynomials.*

Proof. Suppose for definiteness that P is the identity. Then the system (3.14) has polynomial coefficients, and the result follows from Lemma 3.5 and the fact that $w_1, \cdots, w_r$ are eventually rational.

We will now turn our attention to sequences in $\mathcal{F}(\nu, k, l, m)$ considered as a function of their initial value. Let $y(\cdot, t)$ and $z(\cdot, t)$ be sequences which satisfy (3.1) and (3.2), respectively, along with the initial conditions $y(0)=t=z(0)$. Let N_0 be as defined in Theorem 3.2. We will show that there is a finite sample J of integers and T of complex vectors such that $y(n, t)=z(n, t)$ for all $n\in J$ and $t\in T$ implies that $y(n, t)=z(n, t)$ for all $n\in N_0$ and $t\in C_\nu$. Let us first consider homogeneous systems.

THEOREM 3.7. *Let* $T=\{t_1, t_2, \cdots, t_\nu\}$ *be a basis for* C_ν, M *be as in Theorem* 3.2, *and suppose y and z satisfy* (3.4) *and* (3.5), *respectively. If* $y(n, t)=z(n, t)$ *for all* $n\in I(M)$ *and* $t\in T$, *then* $y(n, t)=z(n, t)$ *for all* $n\in N_0$ *and* $t\in C_\nu$.

Proof. We will first show that y and z are linear in the second variable. Let α, β be complex numbers and $t, u\in C_\nu$. Then

$$\begin{aligned}&P(n)[\alpha y(n+1, t)+\beta y(n+1, u)]\\ &\quad=P(n)[\alpha P^{-1}(n)A(n)y(n, t)+\beta P^{-1}(n)A(n)y(n, u)]\\ &\quad=A(n)[\alpha y(n, t)+\beta y(n, u)].\end{aligned}$$

Thus $\alpha y(n,t)+\beta y(n,u)$ satisfies (3.4). But $\alpha y(0,t)+\beta y(0,u)=\alpha t+\beta u$; so the initial condition is also satisfied. Thus

$$y(\cdot,\alpha t+\beta u)=\alpha y(\cdot,t)+\beta y(\cdot,u).$$

Theorem 3.2 implies that $y(n,t_j)=z(n,t_j)$ for all $n\in N_0$. Let t be an arbitrary vector from C_ν. Then t can be expressed in the form $t=\sum_{j=1}^{\nu}\alpha_j t_j$, where $\alpha_i,\cdots,\alpha_\nu$ are constants. For any $n\in N_0$,

$$y(n,t)=\sum_{j=1}^{\nu}\alpha_j y(n,t_j)=\sum_{j=1}^{\nu}\alpha_j z(n,t_j)=z(n,t)$$

and the proof is complete.

THEOREM 3.8. *Let T be a basis for C_ν, M_1 and M_2 be the values given for M in Theorems* 3.2 *and* 3.3, *respectively, and suppose y and z satisfy* (3.1) *and* (3.2), *respectively. If $y(n,0)=z(n,0)$ for all $n\in I(M_2)$ and $y(n,t)=z(n,t)$ for all $n\in I(M_1)$ and $t\in T$, then $y(n,t)=z(n,t)$ for all $n\in N_0$ and $t\in C_\nu$.*

Proof. Define u and v by

$$u(n,t)=y(n,t)-y(n,0),$$

$$v(n,t)=z(n,t)-z(n,0).$$

Note that u and v satisfy the homogeneous equations (3.4) and (3.5), respectively; hence by Theorem 3.7, $u(n,t)=v(n,t)$ for all $n\in N_0$ and $t\in C_\nu$. By Theorem 3.3, $y(n,0)=z(n,0)$ for all $n\in N_0$ and the result follows.

4. Examples. Let us present some examples to illustrate our theory.

Example 4.1. Consider the Taylor series whose partial sums are given by

$$y_1(n)=\sum_{j=0}^{n}(j+1)x^j.$$

Note that y_1 can be generated by the system

$$\begin{aligned} &y_1(n+1)=y_1(n)+(n+2)y_2(n), && y_1(0)=1,\\ &y_2(n+1)=xy_2(n), && y_2(0)=x. \end{aligned} \tag{4.1}$$

This system is homogeneous and has polynomial coefficients of degree $\leqq 1$. For x fixed, Theorem 3.2 would suggest that a program for this computation be tested for $n=0,1,\cdots,7$ ($\nu=2, k=1, l=0$). The theory given here does not provide a test to determine whether or not x is handled correctly. However, the equation for y_2 is uncoupled from y_1; so Theorem 3.6 of [6] would suggest that y_2 be tested for

$$n=0,1,\quad x=1,2,\quad y_2(0)=0,1.$$

It is interesting to note that the partial sums of this series can also be generated by

$$\begin{aligned} &z_1(n+1)=z_1(n)+z_2(n), && z_1(0)=1,\\ &z_2(n+1)=\frac{n+3}{n+2}xz_2(n), && z_2(0)=2x. \end{aligned} \tag{4.2}$$

The question arises as to whether one can show that (4.1) and (4.2) are equivalent by sampling. One cannot directly apply Theorem 3.2 because $y_2(n)\neq z_2(n)$. However, by attaching the equation for y_2 to (4.2) and the equation for z_2 to (4.1) (and

renumbering the coordinates), one can obtain two 3 by 3 systems to which Theorem 3.2 can be applied. Thus one can show (4.1) and (4.2) are equivalent by sampling on the initial segment $I(31)$ ($\nu = 3, k = l = 1$).

Example 4.2. Levy and Lessman [4, p. 234] derive the difference equation

$$U_{n+2} = (n+1)[U_n + U_{n+1}]$$

for the number of ways n parcels and n labels can be so muddled that no parcel has its own label. Letting $y_1(n) = U_n$ and $y_2(n) = U_{n+1}$ we obtain the polynomial system

$$y_1(n+1) = y_2(n),$$
$$y_2(n+1) = (n+1)y_1(n) + (n+1)y_2(n).$$

Theorem 3.2 would suggest that a program for this equation should be tested on the initial segment $I(7)$ ($\nu = 2, k = 1, l = 0$).

Example 4.3. Let us illustrate Theorems 3.4 and 3.6 by the two systems:

$$\begin{aligned} &y_1(n+1) = y_1(n) + ny_2(n) + 1, && y_1(0) = 1, \\ &y_2(n+1) = (n+1)y_2(n), && y_2(0) = 1, \end{aligned} \tag{4.3}$$

$$\begin{aligned} &z_1(n+1) = (n+1)z_2(n) + n + 1, && z_1(0) = 1, \\ &z_2(n+1) = (n+1)z_2(n), && z_2(0) = 1. \end{aligned} \tag{4.4}$$

These two systems have a common solution; namely, $y_1(n) = z_1(n) = n! + n$, $y_2(n) = z_2(n) = n!$ The matrix $C = PB - QA$ from Theorem 3.4 is given by

$$C = \begin{pmatrix} -1 & 1 \\ 0 & 0 \end{pmatrix}.$$

This can be transformed to lower triangular form if we multiply on the right by

$$V = \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix}.$$

After making the change of variable $y = Vw$ we obtain (3.13) with

$$H(n) = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \qquad T(n) = \begin{pmatrix} -n \\ 0 \end{pmatrix}.$$

This implies that $w_1(n) = -n$. Since $w_2 = y_2$ we have $w_2(n) = n!$. After computing F, G, U, and X we obtain from (3.14) and (3.15) the new systems:

$$\begin{aligned} &w_1(n+1) = w_1(n) - 1, && w_1(0) = 0, \\ &w_2(n+1) = (n+1)w_2(n), && w_2(0) = 1, \end{aligned} \tag{4.5}$$

$$\begin{aligned} &w_1(n+1) = -(n+1), && w_1(0) = 0, \\ &w_2(n+1) = (n+1)w_2(n), && w_2(0) = 1. \end{aligned} \tag{4.6}$$

Note that w_1 is a polynomial, and the equations for w_2 are the same in (4.5) and (4.6).

5. Concluding remarks. Bounds have been given for the size of an initial segment which can be used to uniquely identify solutions to systems of difference equations with polynomial coefficients of limited degree. These bounds are given in the spirit of existence theorems; it would be interesting to know whether they are sharp.

The forcing function R in (3.1) can be any function which satisfies a linear first order polynomial system of difference equations. To see this simply imbed the system

which generates R into a larger system as was done in Lemma 2.3. It would be interesting to know whether the coefficients of A could also be taken to be functions satisfying polynomial difference equations. Another direction for further research concerns the case where the coefficients are functions of a parameter as illustrated by Example 4.1 and [6, Thm. 4.8].

REFERENCES

[1] F. R. GANTMACHER, *The Theory of Matrices*, Vol. 1, Chelsea, New York, 1959.

[2] J. B. GOODENOUGH AND S. L. GERHART, *Toward a theory of test data selection*, IEEE Trans. Software Engng., SE-1 (1975), pp. 156–173.

[3] W. E. HOWDEN, *Elementary algebraic program testing techniques*, Tech. Rep. 12, Computer Science Dept., Univ. California at San Diego, La Jolla, CA, 1976.

[4] H. LEVY AND F. LESSMAN, *Finite Difference Equations*, Pitman, London, 1959.

[5] J. H. ROWLAND AND P. J. DAVIS, *On the use of transcendentals for program testing*, J. Assoc. Comput. Mach., 28 (1981), pp. 181–190.

[6] ———, *On the selection of test data for recursive mathematical subroutines*, this Journal, 10 (1981), pp. 59–72.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0009 $01.25/0

ONE STEP TRANSFORMATION OF PERIODIC SEQUENCES BY CELLULAR AUTOMATA*

HISAO YAMADA† AND MASATOSI IMORI†

Abstract. Consider a cellular automaton in one dimension, having m letters as the states of the constituent finite state automata, called cells. It is shown that, once the number of neighbors connected to each cell is fixed, there exist periodic sequences in m letters such that the cellular automaton is not capable of transforming them in one step into periodic sequences in $(m-1)$ letters, preserving at the same time their primitive periods, regardless of the choices of transformation functions. Our proof is by the construction of sequences which prevent any given cellular automaton from performing the task.

Key words. alphabet reduction, cellular automata, multiple neighborhood, prime pattern, primitive period, periodic sequence, sequence mapping, transformation

1. Introduction. In this note, we shall study a property of the simplest and perhaps the most basic cellular automata, namely, those in one dimension having contiguous neighbors for each constituent cell. In particular, we are concerned with a connective property of periodic configurations in such automata. Since the next state which each cell of a cellular automaton will assume is determined in terms of the state information of a fixed number of its neighbor cells, there is an intrinsic connection between periodic sequences constituted by the states of cells and the behavior of the cellular automaton having such a repetitive structure in its cell interconnections. Hence we regard as one of its basic properties what a cellular automaton is or is not capable of doing with repetitive sequences.

In the present note, we shall study the following question. Suppose the set of all periodic sequences in an alphabet of m letters is given, $m \geqq 3$. Does there exist a one-dimensional m-state cellular automaton with the neighbors of a finite and fixed number of contiguous cells such that, when its configuration is a periodic sequence, the cellular automaton is always capable of transforming *in one step* the given periodic configuration into another periodic configuration consisting of $m-1$ letters and having the same primitive period as before, by a suitable choice from its possible transition maps.

As stated, the type of cellular automaton we consider here is not autonomous; that is, we may choose any transformation among all possible ones, *after* the present configuration is given.

For degenerate cases where the number of neighbors connected to each cell is one (i.e., the cases which are shift equivalent to the one in which each cell is isolated without having any other neighbors but itself), we have given a negative answer as well as the greatest lower bound on the primitive periods of those sequences which elude the action of such cellular automata (Imori and Yamada (1981)).

In what follows, a negative answer is given also for the cases where the number of the neighbors is any fixed integer. The proof is by exhibiting for any given cellular automaton a way to construct an example of periodic sequences which will baffle its action. However, we have not yet succeeded in obtaining the greatest lower bounds for these primitive periods.

The negative results we have obtained are not surprising after the fact, because a cellular automaton has a fixed neighborhood size for all cells, yet it is required to

* Received by the editors August 14, 1981, and in revised form September 9, 1982.
† Faculty of Science, University of Tokyo, Bunkyô-ku Hongô, Tokyo 113, Japan.

process in one step any sequence whose primitive period is not bounded. However, our overall proof is not a simple one. It is along the line of a lengthy proof for the degenerate case worked out in our previous note mentioned above.

In recent years interest in theoretical aspects of uniform arrays seems to have declined in spite of the fact that most of what has been shown in the theory so far appears to have solid substance and beauty. The reason for such a waning may be that theoretical treatment of cellular spaces is tedious and difficult because it demands concurrent processing of unbounded numbers of parallel elements. In order to develop a well coordinated theory of cellular spaces in all directions, we feel that systematic examination of this parallelism from the elementary structure upward is first needed. At the most fundamental level of this lies the study of parallel actions of maps, namely the case for the neighborhood size of one, although this obvious fact is often overlooked in the study of cellular spaces. Our previous work mentioned above was an attempt into such a direction, and the present note is its extension to cases for a larger neighborhood.

2. Cellular spaces and cellular automata. Cellular spaces and cellular automata have been defined in several different ways. In what follows, we shall use basically the same definition as in Yamada and Amoroso (1971), but only in one dimension, and also with certain constraints, which will serve for the present purpose.

Let Z be the set of integers. A one-dimensional *cell space*, also denoted by Z, is defined as the set of all integer coordinates, which may be called *cells*, in one-dimensional Euclidean space. Let X be a finite subset of Z called an *index set of neighbor cells*, or a *neighborhood index*. For any cell $z \in Z$, set $\{z+x \mid x \in X\}$ is called the set of neighbor cells of z with respect to X.

When X is fixed, we have a mapping from a cell to the set of its neighbor cells as:

$$N : Z \to Z^{|X|}$$

explicitly defined by

$$N(z) = \{z+x \mid x \in X\} \quad \text{for all } z \in Z.$$

A *polyautomaton* is a system which consists of regularly connected identical or similar automata. A *cellular automaton* is a polyautomaton in which each cell is a finite state semiautomaton with *alphabet* A, called *state set* (whose members are interchangeably called *states*, *symbols*, or *letters*) and with the input consisting of the states of its neighbor cells. Hence the transition function of the semiautomaton is a map

$$\sigma : A^{|X|} \to A.$$

When we have a need, we shall express the cardinality m of A as its subscript, i.e., A_m when $|A_m| = m$. (Note that this definition implies the deterministic nature of the semiautomaton.)

Furthermore, all cells of a cellular automaton, one at each $z \in Z$, are required to change their states according to the same transition function σ at any given step in time, although σ may be different at different time steps.

In order to describe such global transitions of cell states, we employ the notion of configurations which are distributions of states over all of Z. Namely, for a given state set A of finite state semiautomata, a *configuration* over Z is a map

$$c : Z \to A.$$

The set of all such configurations is represented by

$$C = A^Z = \{c \mid c : Z \to A\}.$$

Hence a transition in the cell space is a transition among configurations.

Let $\tau : C \to C$ be a transition function on configurations. Then τ may be defined in terms of N, X and σ as

$$\tau(c) = c' \Leftrightarrow c' : Z \xrightarrow{N} Z^{|X|} \xrightarrow{c^{|X|}} A^{|X|} \xrightarrow{\sigma} A,$$

where

$$\begin{aligned} c^{|X|}[z^{|X|}] &= c^{|X|}[\{z_1, z_2, \cdots, z_{|X|}\}] \\ &= \{c(z_1), c(z_2), \cdots, c(z_{|X|})\} \qquad (z_i \in N(z),\ 1 \leqq i \leqq |X|). \end{aligned}$$

σ and τ are called a *local map* and a *parallel map* (or a global map), respectively. The term "cellular automaton" is sometimes used to define a more restricted system with a fixed neighborhood index and a single parallel function for all cells for all time steps. In such cases, our present system with a fixed neighborhood index but with a set of parallel functions to be used at different time steps is called a *tessellation automaton*. We shall use cellular automaton to refer to both.

In summary, a cellular automaton is a system to be defined by a 4-tuple

$$M_{A,X} = (A, Z, X, T),$$

where

A = nonempty state set,
Z = one-dimensional cell space,
X = neighborhood index, $X \subset Z$,
T = the set of all parallel functions $\tau : C \to C$, arising from A and X.

In this note, the interconnection pattern among cells is arbitrarily chosen so that cell i receives interconnections *from* $(n-1)$ cells; cells $(i-n+1), (i-n+2), \cdots, (i-1)$, for $n \geqq 1$. (However, as long as n neighborhood cells are contiguous, they may be shifted in either direction for any distance with the resulting image shift.) It follows then that cell i is connected *to* cells $(i+1), (i+2), \cdots, (i+n-1)$. The information transmitted through each connection is the present state of the cell from which the interconnection originates. Each cell also utilizes its own present state in addition to the state information from $(n-1)$ other cells. Hence cells $(i-n+1), (i-n+2), \cdots, i$ are the neighbor cells of cell i, called the *neighborhood* collectively, and $|X| = n$. Furthermore, it suffices to write $M_{|A|,|X|}$, instead of $M_{A,X}$. Hence we also write X_n for X with $|X| = n$.

3. Sequences and the problem. Because of the nature of the problem we will study, configurations will be called *sequences*, which they are, for one-dimensional cases.

The image of $i \in Z$ under the sequence c is written $c(i)$, and referred to as the ith letter of the sequence. When it is necessary to explicitly indicate the size of alphabet A, we shall use a subscript, for example, A_m for A. By a *pattern*, we mean a partition of Z. For any $a_i \in c(Z)$, let $B_i = c^{-1}(a_i)$, then $B_i \neq \varnothing$, $B_i \cap B_j = \varnothing$ if $i \neq j$, and $\bigcup_{a_i \in c(Z)} B_i = Z$. Hence c^{-1} defines a partition $\Pi(c)$ on Z, called *pattern* c. B_i is called a *block* of $\Pi(c)$. Conversely, for a given m block partition $\Pi^{(m)}$ of Z (i.e., a pattern), there exists a sequence $c \in A_m^Z$ such that $\Pi(c) = \Pi^{(m)}$, which is unique up to the

permutation of letters in A_m. Hence, in the following sections we will often study sequences in terms of their patterns. However, we on occasion retain nomenclatures from sequences.

For any $S \subseteq Z$ and $k \in Z$, we let $S+k$ denote $\{x+k \mid x \in S\}$, which is a *translation* of S by k. Let the *period set* of c be defined by $P(c)=\{k \mid (\forall z \in Z)\ (c(z+k)=c(z))\}$. Similarly the period set of B, by $P(B)=\{k \mid B+k=B\}$. Whenever it is convenient, we denote by t_k a translation by k, i.e., a mapping $z \mapsto z+k$. Unless explicitly so stated, we assume from now on that t_k is not the identity translation. Define the period set $P(\Pi)$ of a pattern Π by $\{k \mid (\forall B_i \in \Pi)(t_k(B_i)=B_i)\}$. Clearly, for any sequence $c \in A^Z$, $P(\Pi(c))=P(c)$. It is easy to see that these period sets are free modules with operator ring Z. For each $c \in A^Z$, define the *primitive period*, denoted by $\pi(c)$, as the smallest positive element of $P(c)$. If such $\pi(c)$ does not exist, i.e., $P(c)=\{0\}$, we may let $\pi(c)$ be ω, the smallest transfinite ordinal. However, we assume $\pi(c) \neq \omega$ throughout the rest of this note. Similarly, the primitive period $\pi(B)$ of B and the primitive period $\pi(\Pi)$ of Π are defined. Let $\tilde{A}^Z=\{c \mid c \in A^Z$ and $\pi(c) \neq \omega\}$. The elements of $\tilde{A}^Z$ are called *periodic sequences*. Note that c with $\pi(c)=1$ are also included.

Let $T_{m,n}$ be the set of all parallel maps. The choice of parallel maps available to $M_{m,n}$ may be thought of as the input to $M_{m,n}$, originating external to $M_{m,n}$. Viewed in this way, a cellular automaton may also be represented as $M_{m,n}=(A_m^Z, T_{m,n})$, where A_m^Z is the set of all configurations and $T_{m,n}$ is the set of all inputs which cause transitions among configurations. If $m \geqq 2$, then $M_{m,n}$ is a *semiautomaton* with a nondenumerable state alphabet and a finite input alphabet. Our study herein is on a particular problem of the connectivity among periodic configurations of this semiautomaton.

Formally, the problem is: Given m and n, does there exist c in $\tilde{A}_m^z$ such that for any parallel map τ with m symbol alphabet and contiguous neighborhood of size n, either $\tau(c)$ has fewer than m symbols and the primitive period of $\tau(c)$ is smaller than that of c, or else $\tau(c)$ has all m symbols?

Intuitively, we are concerned with whether or not the number of constituent letters of periodic sequences can be reduced by a cellular automaton without changing the periodicity. For example, if we take the sequence

$$\ldots AAB|CACBAACBCAAB|CAC \ldots$$

of primitive period 12, consisting of $\{A, B, C\}$, and consider mappings $\{A, B, C\} \to \{A, B\}$, $\{A, C\}$, or $\{B, C\}$, (which may be collectively thought of as a cellular automaton, having $|X|=n=1$), then,

(1) $\{A \mapsto B, B \mapsto B, C \mapsto C\}$ reduces it to

$$\ldots BBB|CBCBBB|CBCBBB|CBC \ldots .$$

Similarly

(2) $\{A \mapsto C, B \mapsto B, C \mapsto C\}$ to

$$\ldots CCB|CCCB|CCCB|CCCB|CCC \ldots ,$$

(3) $\{A \mapsto A, B \mapsto A, C \mapsto C\}$ to

$$\ldots AAA|CACAAA|CACAAA|CAC \ldots ,$$

(4) $\{A \mapsto A, B \mapsto C, C \mapsto C\}$ to

$$\ldots AAC|CACCAACCCAAC|CAC \ldots ,$$

(5) $\{A \mapsto A, B \mapsto B, C \mapsto A\}$ to

$$\ldots AAB|AAAB|AAAB|AAAB|AAA \ldots,$$

(6) $\{A \mapsto A, B \mapsto B, C \mapsto B\}$ to

$$\ldots AAB|BABBAABBBAAB|BAB \ldots.$$

The cases where $n = 1$ are reported in Imori and Yamada (1981). For the case where $m = 2$, the nonexistence of such c is obvious for any n because all $c \in \tilde{A}_1^Z$ have period 1. We shall show below that, for all $m \geqq 3$ and $n \geqq 2$, a $c \in \tilde{A}_m^Z$ exists whose primitive period is reduced by any τ of the parallel maps if the σ reduces the number of letters.

4. Prime patterns and channel matrices. Let $M_{m,n} = (A_m, Z, X_n, T_{m,n})$ be a cellular automaton, each cell of which has the state alphabet of m letters, and n contiguous neighbor cells. We say that an m letter sequence $c \in \tilde{A}_m^Z$ is an (m, n)-*prime pattern* if, for any $\tau \in T_{m,n}$ such that $\tau(c) \in \tilde{A}_{m-1}^Z$, $\tau(c)$ has a smaller primitive period than that of c. In the next section, we shall establish the existence of (m, n)-prime patterns for any $m \geqq 3$ and $n > 1$. The proof is by showing a procedure for the construction of an (m, n)-prime pattern for any such m and n, which we now begin.

Let $L_{m,n}$ be the set of all local maps $\sigma : A_m^n \to A_m$, from which the set of parallel maps $T_{m,n}$ are derived. Define L and L', which are subsets of $L_{m,n}$ such that

(i) $\sigma \in L \Leftrightarrow \sigma(A_m^n) = A_m$,

(ii) $L' = L_{m,n} - L$.

Then

PROPOSITION 4.1. *The number z of distinct maps in L' is given by*

$$z = \#L' = m \cdot (m-1)^{m^n}.$$

Proof. There are m^n possible combinations of letters for n arguments. Since $\sigma \in L'$ can have at most $(m-1)$ letter values for each of these, there are $(m-1)^{m^n}$ different assignments of letter values. Also there are m different choices for the missing letter. □

We shall assume that all maps of L' are arbitrarily ordered, then indexed in that order.

Let A_m^- be the set of all semi-infinite sequences, to the left, of letters in A_m, and let A_m^+ be likewise to the right. For any $\sigma \in L'$, let τ_σ be its parallel map. Then

PROPOSITION 4.2. $(\forall x \in A_m^-)$ $(\exists k \geqq n)$ $(\forall v \in A_m^{n-1})$ $(\exists u^{(1)}, u^{(2)} \in A_m^k)$ $(\forall y \in A_m^+)$ $(u^{(1)} \neq u^{(2)}$ *and* $\tau_\sigma(xu^{(1)}vy) = \tau_\sigma(xu^{(2)}vy))$.

Proofs. In the images $\tau_\sigma(xu^{(1)}vy)$ and $\tau_\sigma(xu^{(2)}vy)$, the subsequences of the image of x from the left are obviously identical. Hence the possible difference starts from the leftmost point of u on. Take the image subsequences from there to the end of the images of $u^{(1)}$ and $u^{(2)}$. Since $u^{(1)}$ and $u^{(2)} \in A_m^k$, $u^{(1)}$ and $u^{(2)}$ may be chosen from m^k possible choices for any fixed $v \in A_m^{n-1}$. On the other hand, the number of all possible choices of the images of $u^{(1)}v$ and $u^{(2)}v$, whose length is $k+n-1$, is $(m-1)^{k+n-1}$. Hence, if we choose the smallest k such that

$$m^k > (m-1)^{k+n-1},$$

which may be always satisfied by choosing k large enough, then there is at least one image which has at least two distinct preimages having the same right end of length $n-1$. □

We shall make use of this k below.

We will make use of a De Bruijn sequence of order m and degree n below, which is the shortest sequence consisting of m letters such that every possible length n sequence in m or less letters appears at least once as its subsequence (De Bruijn (1946)). It may be given in a circular form, or it may be opened into a finite linear sequence.

PROPOSITION 4.3 (Good (1946)). *The shortest linear De Bruijn sequence of order m and degree n has the length m^n+n-1.*

The first step of the construction of an (m, n)-prime pattern is to set a framework for the pattern. For any given cellular automaton $M_{m,n}=(A_m, Z, X_n, T_{m,n})$, take a $[(k+n-1)\cdot m\cdot(m-1)^{m^n}+m^n+n-1]\times q$ matrix, called a *channel matrix*, where q is to be determined later. Its rows are partitioned into *region I*, consisting of $(k+n-1)\cdot m\cdot(m-1)^{m^n}$ rows, and *region* II, consisting of m^n+n-1 rows. Each row will be called a *channel.* Region I is in turn partitioned into $z=m\cdot(m-1)^{m^n}$ *bands* consisting of $k+n-1$ channels. The overall structure of these channels is shown in Fig. 1. Note that the number of bands is the same as the number of maps in L' and $k+n-1$ is the length of $u^{(i)}v$ in Proposition 4.2.

Next, take band i consisting of $n+k-1$ channels in region I, $1\leqq i\leqq z$, and look at the first two columns, referring to Fig. 2. In order to fill these two columns, assume that the lower (left) ends of the first two columns of band $i-1$ are filled with the identical vertical sequence v_{i-1} of length $n-1$. Using this v_{i-1} as the tail end of x in Proposition 4.2, determine $u_i^{(1)}$, $u_i^{(2)}$ and (arbitrary) v_i for $\sigma_i\in L'$ as was done in the

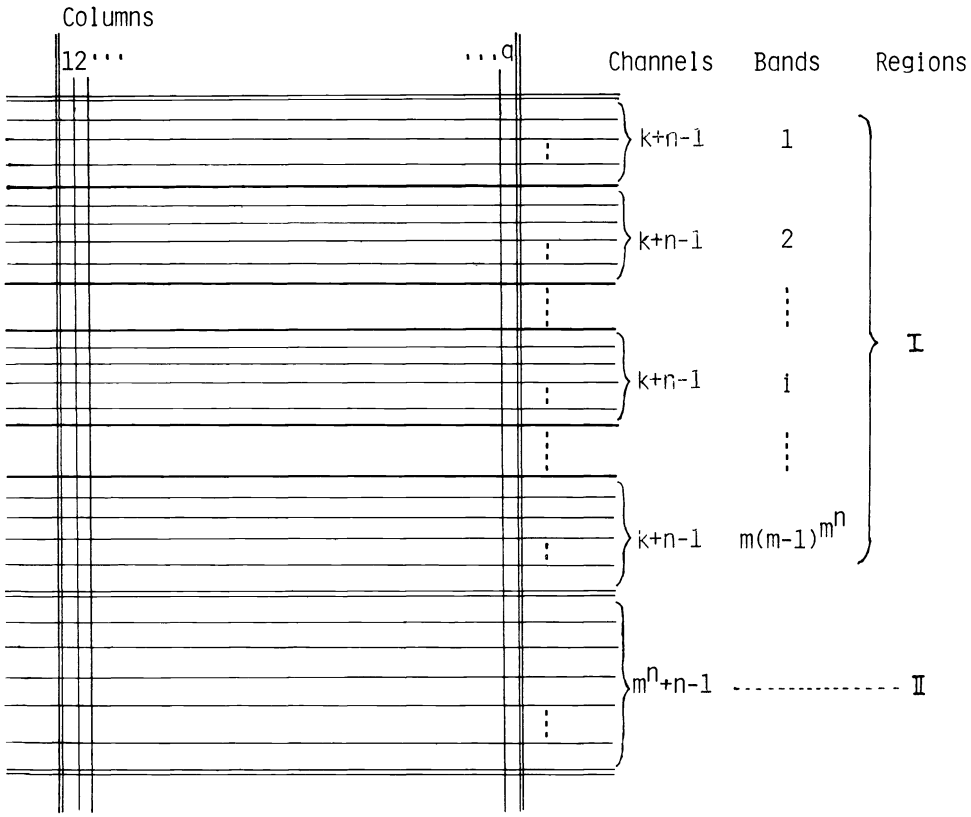


FIG. 1. *Structure of channel matrix.*

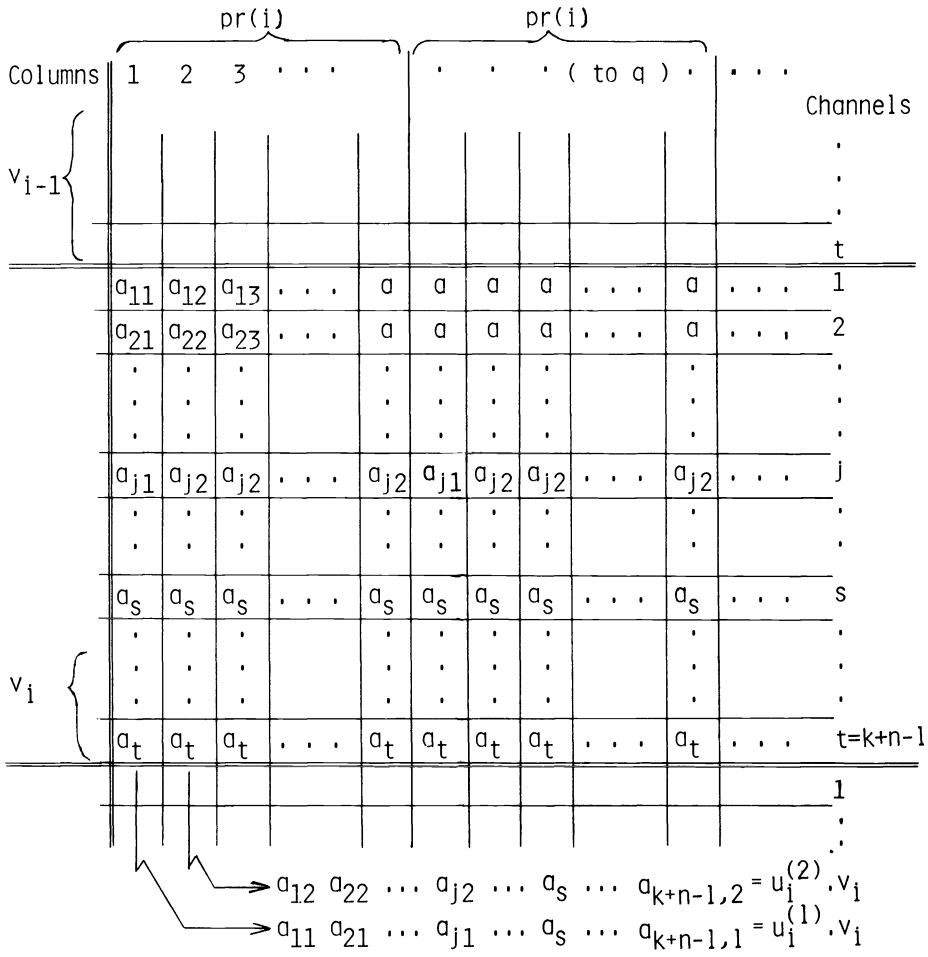


FIG. 2. *Structure of band i in region* I.

proof of the proposition, and fill columns 1 and 2 respectively with $u_i^{(1)}v_i$ and $u_i^{(2)}v_i$ vertically. Clearly this procedure fills the lower ends of column 1 and 2 with the identical vertical sequence v_i of length $n-1$, satisfying the assumption for band $i+1$. Hence, if we only assume that the first two columns of band 1 are vertically preceded by an identical sequence of length $n-1$, a_1^{n-1} $(a_1 \in A_m)$, then the assumption inductively holds down to the last band, and we are able to fill the first two columns completely from band 1 down to band z.

In order to complete the filling of region I, start from band 1. For band i, choose the ith smallest prime number p_i, and check channel by channel whether or not the letter in column 1 is the same as that of column 2. If not, fill the channel from the left by the repetition of $a_{j_1}a_{j_2}^{(p_i-1)}$ up to column $q = \prod_{i=1}^{z} p_i$, where z is the number of bands, p_i is the ith prime, a_{j_1} and a_{j_2} are the letters found in columns 1 and 2, respectively, and $a_{j_2}^{(p_i-1)}$ denotes the (p_i-1) repetition of letter a_{j_2}. Using the same p_i, do similarly for all channels of the band having different letters in columns 1 and 2. If two letters found are identical, say a_s, between channels 1 and 2, then the entire channel will be filled with a_s. This procedure is repeated for all bands from 1 to z.

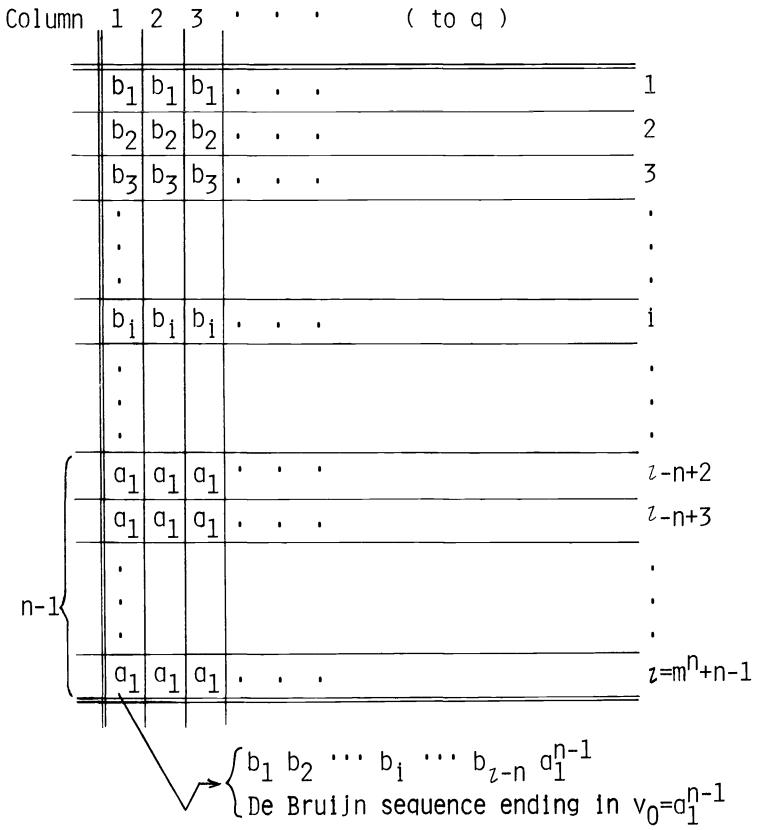


FIG. 3. *Structure of channels in region* II.

In order to fill the channels of region II, take a linear De Bruijn sequence of order m and degree n, and fill with it each column of region II vertically from the top to the bottom (see Fig. 3). The number of channels thus required is m^n+n-1, as was stated by Proposition 4.3. Any one of the De Bruijn sequences of order m and degree n will suffice, and the one we choose in Fig. 3 is the one which ends in a_1^{n-1}, in order to match with the arbitrarily chosen a_1^{n-1} end of the prefix for $u_1^{(i)}v_1$ of band 1, for that is what this De Bruijn sequence would become in the construction below.

Referring back to Fig. 1, we now have the entire channel matrix filled. Each of m^n+n-1 channels of region II is filled with an identical letter. In addition, there are at least $(n-1)\cdot m\cdot(m-1)^{m^n}$ channels of region I so filled because of the existence of v_i, $1\leqq i\leqq m\cdot(m-1)^{m^n}$, whose length is $n-1$. Hence:

PROPOSITION 4.4. *There are at most* $k\cdot m\cdot(m-1)^{m^n}$ *channels in the matrix which contain periodic sequences whose primitive periods are prime numbers, distinct from band to band, but identical among channels within a band.*

Let us call the actual number of such channels r,

$$r=\sum_{i=1}^{\#L'} D(u_i^{(1)},u_i^{(2)})\leqq k\cdot m\cdot(m-1)^{m^n},$$

where $D(x, y)$ is the distance between sequence x and y (i.e., the number of locations

where x and y have different letters). Furthermore let d be the actual number of channels which are filled with a respective single letter, then

$$r+d=(k+n-1)\cdot m\cdot(m-1)^{m^n}+m^n+n-1.$$

If we take $q=PP(z)$, which denotes the product of first z prime numbers, where $z=m\cdot(m-1)^{m^n}$, the number of bands, then all periodic sequences appearing among the channels of Fig. 1 would for the first time end their periods on the qth column together. Therefore, if we continue periodic sequences in channels to the $(q+1)$th column on, the same channel matrix would repeat itself to the right over every q columns.

5. (m, n)-prime patterns. In order to construct an (m, n)-prime pattern, refer to the completed chanel matrix of Fig. 1, and let w_j be the sequence made up of letters of column j from the top down. By a *standard pattern* (with respect to a channel matrix), we shall mean the (infinite) periodic sequence h made up of the repetition of

$$f=w_1w_2\cdots w_j\cdots w_q,$$

where f is the concatenation of all column sequences w_j in the order of column numbers, and $q=PP(z)$ as discussed in the preceding section. We shall call this concatenating process the *multiplexing* of channels.

The standard pattern h thus constructed here is a specific example of a more general class of patterns called *d-diluted R-patterns*, which we have studied in Imori and Yamada (1981), where d is the number of channels consisting of single letters, as discussed in the preceding Section, and R is a set of positive integers which are relatively prime in pairs. In the present case, R is the set of the lengths of basic sequences from which the sequences of those channels containing periodic patterns are generated. Thus, in the present case, R is the set of the smallest z prime numbers, leading to $\#R=z$, where z was first given in Proposition 4.1.

PROPOSITION 5.1. *The primitive period of standard pattern h is given by $\pi(h)=(r+d)\cdot PP(z)$.*

Proof. Take a channel in the channel matrix which contains more than one letter in it, namely a channel whose sequence is a repetition of, say $a_{j_1}a_{j_2}a_{j_2}\cdots a_{j_2}$ (see Fig. 2), to be denoted by $b_1b_2\cdots b_{e_j}$. After the multiplexing of channels to form h, the letters of this $b_1b_2\cdots b_{e_j}$ have their respective places in the multiplexed sequence h, to be denoted by $\bar{b}_1,\bar{b}_2,\cdots,\bar{b}_{e_j}$. Now take $\bar{b}_1$ and translate it by $\pi(h)$, then we naturally find another one of the same letter b_1 there, to be denoted by $\bar{b}'_1$. It arose through multiplexing from one of the channels in the channel matrix. Denote by b'_1 the "preimage" of letter $\bar{b}_1$ with respect to the multiplexing.

Next, take the original b_2 and repeat the same procedure, and find $\bar{b}_2$, $\bar{b}'_2$ and b'_2. Since $\bar{b}_1$ and $\bar{b}_2$ are $(r+d)$ apart after multiplexing, $\bar{b}'_1$ and $\bar{b}'_2$ are also $(r+d)$ apart, hence b'_1 and b'_2 must be on the same channel in the matrix. Continuing the same procedure to the last b_{e_j} and then repeating the whole procedure once more, we will obtain $(b'_1b'_2\cdots b'_{e_j})^2$ on the same channel of the matrix as its subsequence. Clearly $b'_1b'_2\cdots b'_{e_j}=a_{j_1}a_{j_2}\cdots a_{j_2}$ and the period of $b'_1b'_2\cdots b'_{e_j}$ is the same as that of $a_{j_1}a_{j_2}\cdots a_{j_2}$. Hence the channel for $b_1b_2\cdots b_{e_j}$ and the channel for $b'_1b'_2\cdots b'_{e_j}$ belong to the same band, because each band is given a unique prime number for the period of the sequences in the channels. We denote by p the period of the band.

Let the channel for $b_1b_2\cdots b_{e_j}$ be row m in the channel matrix and the channel for $b'_1b'_2\cdots b'_{e_j}$ be row m'. We show that $m=m'$. If $m\neq m'$, then all the channels for row $m+n\cdot d$, where $d=m-m'$, $0<|d|<k$ and $n\in Z$, have the prime number p as the period of the sequences. But there exists n_0 such that the channel for row

$m+n_0 \cdot d$ belongs to a band which has prime number p' not equal to p as the period of the sequences in the channels, which is a contradiction. Hence the channel for $b_1 b_2 \cdots b_{e_j}$ and the channel for $b'_1 b'_2 \cdots b'_{e_j}$ are the same one. This means that $\pi(h)$ is a multiple of $(r+d)e_j$, and, as far as channel j is concerned, translation $t_{(r+d)e_j}$ by $(r+d)e_j$ maps its sequence onto itself, and $(r+d)e_j$ is the smallest such displacement.

Each of those channels which consist of a single letter is clearly mapped onto itself by translation $t_{(r+d)}$ although $(r+d)$ may not be the smallest such displacement for some channels.

Finally, $\pi(h)$ is the smallest common multiple of the displacements of all channels in the matrix, which is clearly $(r+d) \cdot PP(z)$. □

PROPOSITION 5.2. *The primitive period $\pi(c)$ of a standard pattern is reduced to some number less than $\pi(c)$ by the parallel map τ_σ induced by any local map in L', the set of maps $\sigma : A_m^n \to A_{m-1}$, for all $A_{m-1} \subset A_m$.*

Proof. From the construction of the channel matrix, there exists for each $\sigma_i \in L'$ a corresponding band i in its region I (see Fig. 1). Again from the construction, band i contains at least one channel whose columns 1 and 2 contain distinct letters which will map onto an identical letter under τ_{σ_i} (see channel j in Fig. 2). Let us say that the *length* of sequence $a_{j_1} a_{j_2} a_{j_2} \cdots a_{j_2}$ of the channel is a prime number, $\mathrm{pr}(i)$. This means that the primitive period of the standard pattern after the application of τ_{σ_i} reduces at least by the factor $\mathrm{pr}(i)$, which may be arrived at by an argument similar to the one in the proof of Proposition 5.1. □

PROPOSITION 5.3. *A standard pattern does not reduce the number of letters in it after the transformation by the parallel map τ_σ which is induced by any local map in L, the set of surjections*

$$\sigma : A_m^n \to A_m.$$

Proof. Region II of the channel matrix consists of columns of a De Bruijn sequence of order m and degree n, containing as its substrings all possible sequences of length n, consisting of m distinct letters or less. Since σ is a surjection, each letter of A_m must appear somewhere in the image of the De Bruijn sequence, which makes up parts of the standard pattern, in fact $q = PP(z)$ times, after the multiplexing. □

THEOREM 5.4. *A standard pattern is an (m, n)-prime pattern, with primitive period $(r+d) \cdot PP(z)$.*

Proof. Any $\sigma \in L$ will induce a parallel map τ_σ which is ineffective to reduce the number of letters of the pattern, by Proposition 5.3. On the other hand, any parallel map τ_σ, induced by $\sigma \in L'$ reduces the primitive period of the standard pattern by Proposition 5.2, as well as the number of letters. □

Note that region II is incorporated into the standard pattern in order to cope with the possibility, though unlikely, that the pattern generated by the multiplexing of region I alone might retain the primitive period *and* yet reduce the number of letters when acted upon by τ_σ arising from one of $\sigma \in L$.

If we compute values r and $r+d$ for $m=3$ and $n=2$, we obtain $r+d=4618$ and $z=1536$. Since $PP(54) \cong 7.09 \times 10^{100}$, $PP(1536)$ is beyond our imagination. However, this upper bound is a very generous one, arising from the fact that we wanted to show a simplest construction procedure of an (m, n)-prime pattern first and chose a standard pattern as a possible such example.

As an attempt to produce a shorter (m, n)-prime pattern, we note that, although $\#L'$ is $m \cdot (m-1)^{m^n}$ as we saw in Proposition 4.1, we do not have to give a band to all σ in L'. Because of the symmetry, if we take only those maps $L'_m =$

$\{\sigma|\sigma : A_m^n \to A_m - \{a_m\}\}$, then all other maps which would map into $A_m - \{a_i\}$ for $a_i \in A_m$, $a_i \neq a_m$, will be automatically taken care of.

Now take a channel matrix which has taken the above factor into consideration, construct a pattern from it by multiplexing, and call it a *reduced S-pattern*. Then;

PROPOSITION 5.5. *A reduced S-pattern is an* (m, n)*-prime pattern with primitive period* $w \cdot PP(z')$, *where*

$$w = (k+n-1) \cdot (m-1)^{m^n} + m^n + n - 1, \qquad z' = (m-1)^{m^n}.$$

For this reduced S-pattern, $m = 3$ and $n = 2$ gives $z' = 512$, which is better than the above, and the effect of the smaller z' will be much more pronounced for larger n.

We further note that the k we used in the discussion above does not have to be a common constant for all bands, and it may be changed from band to band.

In Yamada and Imori (1982) we show a much more "economical" example of (3, 2)-prime patterns with the primitive period of 2.33×10^{39}, based on 58 channels and 26 primitive generator sequences, which most likely is not the minimal, although far shorter than the value obtained from the procedure described above.

6. Concluding remarks. We have established in this note that for any given one-dimensional cellular automaton $M_{m,n}$ of m states and n neighbors, $m \geqq 3$ and $n \geqq 2$, there exist periodic sequences such that no transformation of $M_{m,n}$ can reduce the number of distinct letters in those sequences, without at the same time reducing their primitive periods. So far we have failed to obtain the exact value of the minimal primitive period of such (m, n)-prime patterns, except for $n = 1$ which is shown in Imori and Yamada (1981). Even for such restricted cases, our proof is involved and its extension to the general (m, n) case is expected to be much more involved.

M. Nasu has pointed out to us the following two facts:

(1) Although we have investigated the effect, on the period, of only those functions which reduce the number of letters of periodic sequences by one, the technique we used can be extended to include also those functions which preserve the number of letters but have mutually erasable configurations (i.e., those configurations having the same image) (Moore 1962), Myhill (1963) and Richardson (1972)).

(2) Region II of Figure 1 was used to place all possible sequences of length n in the form of a De Bruijn sequence. However, by increasing the number of channels in each band to $k+n$, we may transfer all m^n subsequences of length n in region II to region I as the tail part of length n in arbitrary chosen m^n of $m \cdot (m-1)^{m^n}$ bands.

There are a number of interesting related problems which we have not investigated. Some of them are:

(A) When $m \geqq 3$ and $n \geqq 2$ are given, what is the shortest primitive period which admits an (m, n)-prime pattern? For $n = 1$, we have shown in Imori and Yamada (1981) that it is ${}_mC_2 \cdot PP({}_mC_2)$. However, the bound we gave in Proposition 5.5 herein is much larger than the minimum because it is a mere by-product of a specific construction procedure we employed.

(B) When $m \geqq 3$, $n \geqq 2$ and $k \in Z$ are given, what is the necessary and sufficient condition $P_{m,n}$ for k such that, for all $c \in \tilde{A}_m$, if $\pi(c) = k$, then there exists $M_{m,n}$ which maps c into $\tilde{A}_{m-1}$ while retaining the primitive period? One obvious example of k is that k is a prime number (in this case for all m and n), but very little is known beyond this trivial fact.

(C) In a more general formulation, let $\tilde{A}_m(k)$ be the set of all periodic sequences in m letters whose primitive period is k. Then, each $\tilde{A}_m(k)$ may be partitioned according

to destination under each transformation of $M_{m,n}$, for each $n \geqq 1$, and the cardinalities of each class be found.

(D) As the counterpart of (C) above, prime sequences may be classified according to their structural properties, and their relation to the classification of (C) above may be studied. The study of general properties of periodic sequences may be a prerequisite to this study.

(E) In this note we have been concerned with one-step transformations. However, a more general problem is first to transform sequences within $\tilde{A}_m^Z$ by a successive application of various transformations, then reduce them into $\tilde{A}_{m-1}^Z$, all the while keeping the primitive period intact. (This is a problem of the connectivity of periodic configurations of $M_{m,n}$. The connectivity question has been studied by some authors, but mostly within the set of "finite" configurations.)

(F) The above problem (E) may be restricted to a repetition of a single transformation in $L = \{\sigma | \sigma : A_m^n \to A_m\}$. That is, the study of $M_{m,n}$ to see, after $c \in \tilde{A}_m^Z$ is given, whether or not σ exists such that the successive application of τ_σ will take c into $\tilde{A}_{m-1}^Z$, while keeping the primitive period intact.

Acknowledgment. We would like to extend our sincere appreciation to Professor Masakazu Nasu of the Research Institute of Electrical Communication, Tôhoku University, who carefully read an earlier version of this report and pointed out some errors in the original proof. We are also very much indebted to two anonymous referees who have carefully read our manuscript and made numerous suggestions to improve it.

REFERENCES

[1] N. G. De Bruijn (1946), *A combinatorial problem*, Nederl. Akad. Wetensch. Proc., 49, pp. 758–764.

[2] I. J. Good (1946), *Normal recurring decimals*, J. London Math. Soc., 21, pp. 167–169.

[3] M. Imori and H. Yamada (1981), *A noninjective mapping of periodic sequences by cellular automata*, Proc. 4th Australian Computer Science Conference, Brisbane, Queensland, Australia, May 11–13, 1981, pp. 222–250. (Also available as Technical Report 81–11, Dept. Information Science, Faculty of Science, Univ. of Tokyo, April 1981.)

[4] E. F. Moore (1962), *Machine models of self reproduction*, Proc. Symp. Appl. Math, 14, pp. 17–33.

[5] J. Myhill (1963), *The converse of Moore's Garden of Eden theorem*, Proc. Amer. Math. Soc., 14, pp. 685–686.

[6] D. Richardson (1972), *Tessellations with local transformations*, J. Comp. System Sci., 6.

[7] H. Yamada and S. Amoroso (1971), *Structural and behavioral equivalences of tessellation automata*, Inform. Contr., 18, pp. 1–31.

[8] H. Yamada and M. Imori (1982), *One-step transformation of periodic sequences by cellular automata*, Technical Report 82-06, Dept. Information Science, Faculty of Science, University of Tokyo, March 1982.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0010 $01.25/0

TREE TRAVERSAL RELATED ALGORITHMS FOR GENERATING INTEGER PARTITIONS*

TREVOR I. FENNER† AND GEORGHIOS LOIZOU†

Abstract. We define and characterize orderings of the set of partitions of a positive integer n obtained by traversing the binary tree of the set of partitions of n in each of the three principal orders. Loop-free algorithms for generating the set of partitions in each of these three orders and also in the three corresponding reverse orders are derived. The asymptotic behaviour of each of these six algorithms is examined and compared.

Key words. integer partitions, binary tree traversal orders, asymptotic analysis, loop-free algorithms

1. Introduction. In [5] a representation of the partitions of an integer n as a binary tree T_n was obtained. It was shown that preorder traversal of T_n yields $\mathscr{P}_n$, the set of partitions of n, in lexicographic order and this relationship was then utilized to obtain a loop-free algorithm for generating $\mathscr{P}_n$ in this order. It was further shown how the same approach could be used to derive an algorithm for generating $\mathscr{P}_n$ in the order corresponding to the inorder traversal of T_n.

In this paper, extending the results of [5], we derive these and other algorithms in a unified manner and compare their asymptotic behaviour. After § 2, which is mainly devoted to definitions, in § 3 we systematically examine and characterize the orderings of $\mathscr{P}_n$ induced by the traversal of T_n in the three principal orders [9, p. 316]. Then, in § 4, we proceed to derive loop-free algorithms for generating $\mathscr{P}_n$ in each of these orders as well as in the three corresponding reverse orders. In § 5, following the approach employed in [7], we investigate the asymptotic behaviour of each of these algorithms for large n.

Finally, in § 6, we conclude that the asymptotic costs of all six algorithms will be fairly similar for any reasonable computational model, although in the nonasymptotic case this may not be so, even for quite large values of n.

2. Preliminaries. A *partititon* of a positive integer n is a finite sequence $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_k)$ of positive integers (the *parts* of α) such that

$$\alpha_1+\alpha_2+\cdots+\alpha_k = n, \tag{1}$$

$$\alpha_1 \geqq \alpha_2 \geqq \cdots \geqq \alpha_k. \tag{2}$$

We use $\mathscr{P}_n$ to denote the set of partitions of n. We will usually find it more convenient to use a *multiplicity representation* in terms of the *distinct* parts of the partition α and their respective multiplicities, i.e. if α has m_1 parts equal to β_1, m_2 parts equal to β_2, and so on, and if d is the number of distinct parts of α, then instead of (1) and (2) we have

$$m_1\beta_1+m_2\beta_2+\cdots+m_d\beta_d = n, \tag{3}$$

$$\beta_1>\beta_2>\cdots>\beta_d, \qquad m_i \geqq 1 \quad \text{for } 1 \leqq i \leqq d, \tag{4}$$

and we write

$$\alpha = \beta_1^{m_1}\beta_2^{m_2}\cdots\beta_d^{m_d}. \tag{5}$$

* Received by the editors August 25, 1981, and in revised form July 27, 1982.

† Department of Computer Science, Birkbeck College, University of London, Malet Street, London WC1E 7HX, England.

A variant of this notation that we will utilize in § 4 is to express α as

$$\alpha = \beta_1^{m_1}\beta_2^{m_2}\cdots\beta_\Delta^{m_\Delta}1^{\mu}, \tag{6}$$

where $\Delta \geqq 0$ is the number of distinct parts greater than 1 and $\mu \geqq 0$ is the number of parts equal to 1, i.e.

$$\begin{aligned} &\Delta = d-1, \qquad \mu = m_d \quad \text{if } \beta_d = 1, \\ &\Delta = d, \qquad\quad \mu = 0 \quad\ \ \text{if } \beta_d > 1. \end{aligned} \tag{7}$$

Thus (3) and (4) are now replaced by

$$m_1\beta_1 + m_2\beta_2 + \cdots + m_\Delta\beta_\Delta + \mu = n, \tag{8}$$

$$\beta_1 > \beta_2 > \cdots > \beta_\Delta > 1, \qquad m_i \geqq 1 \quad \text{for } 1 \leqq i \leqq \Delta, \quad \mu \geqq 0. \tag{9}$$

We refer to the three notations above as *α-notation*, *β-notation* and *$\beta\mu$-notation*, respectively.

The number of cases in the algorithms that we derive in § 4 is reduced if we define β_0 and m_0 so that $\beta_0 > n$ and $m_0 > 1$ (cf. [11, p. 193]). It is also convenient to define α_0 to be the same as β_0. We therefore let

$$\alpha_0 = \beta_0 = m_0 = n+1. \tag{10}$$

A partition $\hat{\alpha}$ is a *subpartition* of α if the multiplicity of any distinct part of $\hat{\alpha}$ does not exceed its multiplicity in α. Σ_l denotes the subpartition of α represented by $\beta_1^{m_1}\beta_2^{m_2}\cdots\beta_l^{m_l}$, so that for any l, $0 \leqq l < d$,

$$\alpha = \Sigma_l \beta_{l+1}^{m_{l+1}}\cdots\beta_d^{m_d},$$

where Σ_0 is understood to be the vacuous partition of zero.

If $\mathscr{C}(\alpha)$ is some condition on α then $\mathscr{P}_n[\mathscr{C}(\alpha)]$ denotes the subset of $\mathscr{P}_n$ satisfying $\mathscr{C}(\alpha)$, i.e. $\mathscr{P}_n[\mathscr{C}(\alpha)] = \{\alpha \in \mathscr{P}_n | \mathscr{C}(\alpha)\}$. Furthermore, given two conditions $\mathscr{C}_1(\alpha)$ and $\mathscr{C}_2(\alpha)$, we write $\mathscr{P}_n[\mathscr{C}_1(\alpha), \mathscr{C}_2(\alpha)]$ for the subset satisfying both conditions, i.e. $\{\alpha \in \mathscr{P}_n | \mathscr{C}_1(\alpha)$ and $\mathscr{C}_2(\alpha)\}$. We also extend this notation to more than two conditions, e.g. $\mathscr{P}_n[\beta_d = 1, m_d = 1, \beta_{d-1} > 2]$ is the set of partitions of $n > 1$ which have one part of size 1 and no parts of size 2.

P_n denotes $\#\mathscr{P}_n$, the cardinality of $\mathscr{P}_n$, and, similarly, we denote the cardinality of $\mathscr{P}_n[\mathscr{C}(\alpha)]$ by $P_n[\mathscr{C}(\alpha)]$, etc. Hardy and Ramanujan [8] derived a number of asymptotic formulae for P_n, the simplest of these being

$$P_n \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{2n/3}},$$

where the relative error is $O(n^{-1/2})$. Starting from a more refined asymptotic formula, we have shown [7] that, for small values of j independent of n,

$$\frac{P_{n-j}}{P_n} = 1 - \frac{\pi j}{\sqrt{6n}} + O\left(\frac{1}{n}\right). \tag{11}$$

In [5] we obtained a representation of $\mathscr{P}_n$ as a binary tree T_n. For any $\alpha \in \mathscr{P}_n$, we define the left and right sons of α, $L(\alpha)$ and $R(\alpha)$, respectively, as follows:

> $L(\alpha)$ is the partition obtained from α by replacing two parts of size 1 by one part of size 2, but $L(\alpha) =$ **nil** if α has fewer than two parts of size 1.

$R(\alpha)$ is the partition obtained by removing one part of size 1 and adding 1 to the smallest part greater than 1 provided its multiplicity is unity (i.e. $m_{d-1}=1$), but $R(\alpha)=$ **nil** if α has no part of size 1 or if $m_{d-1}>1$.

In § 4, $L(\alpha)$ and $R(\alpha)$ are defined formally in terms of $\beta\mu$-notation.

We have shown [5] that these two functions, L and R, determine a binary tree T_n, with root 1^n, which contains each of the partitions in $\mathscr{P}_n$, i.e. for each $\alpha\in\mathscr{P}_n$ there exists a unique sequence of functions $\sigma_1,\sigma_2,\cdots,\sigma_t$ such that each σ_i, $1\leqq i\leqq t$, is either L or R and

$$\sigma_1(\sigma_2(\cdots(\sigma_t(1^n))\cdots))=\alpha.$$

3. Traversal orders of T_n. We now consider the orderings of $\mathscr{P}_n$ induced by the traversal of T_n in the three principal orders, viz. preorder, inorder and postorder [9, p. 316]; the corresponding induced orderings of $\mathscr{P}_n$ are designated L-order, M-order and P-order, respectively, and are denoted by $<_L$, $<_M$ and $<_P$. Although preorder has been investigated in [5], we include these results in the present unified treatment for completeness.

For any $\alpha\in\mathscr{P}_n$, we define $[\alpha]$ to be the closure of $\{\alpha\}$ under L and R, excluding **nil**. Thus, from the remarks at the end of § 2, we observe that $[1^n]=\mathscr{P}_n$.

The following lemma from [5] characterizes $[\alpha]$.

LEMMA 1. *Let $\alpha=(\alpha_1,\alpha_2,\cdots,\alpha_k)$ and $\gamma=(\gamma_1,\gamma_2,\cdots,\gamma_s)$ be any partitions in $\mathscr{P}_n$, and let h be the number of parts of α greater than* 1, *i.e. $h=k-\mu$. Then $\gamma\in[\alpha]$ if and only if*

$$\gamma_i=\alpha_i \quad \text{for } 1\leqq i\leqq h-1, \quad \gamma_h\geqq\alpha_h.$$

From the definition of L and R, we then have:

COROLLARY 1. *$\gamma\in[L(\alpha)]$ if and only if*

$$\gamma_i=\alpha_i \quad \text{for } 1\leqq i\leqq h, \quad \gamma_{h+1}\geqq 2.$$

COROLLARY 2. *$\gamma\in[R(\alpha)]$ if and only if*

$$\gamma_i=\alpha_i \quad \text{for } 1\leqq i\leqq h-1, \quad \gamma_h\geqq\alpha_h+1.$$

We extend the relations $<_L$, $<_M$ and $<_P$ to subsets of $\mathscr{P}_n$ by defining, for any Π_1, $\Pi_2\subseteq\mathscr{P}_n$,

$$\Pi_1<\Pi_2\Leftrightarrow\pi_1<\pi_2 \quad \text{for all } \pi_1\in\Pi_1,\ \pi_2\in\Pi_2.$$

(We also write $\pi_1<\Pi_2$ if $\Pi_1=\{\pi_1\}$, etc.)

We first consider L-order. For any $\alpha\in\mathscr{P}_n$ for which $L(\alpha)\neq$ **nil** and $R(\alpha)\neq$ **nil**, by the definition of $<_L$, we have

$$\alpha<_L[L(\alpha)]<_L[R(\alpha)]. \tag{12}$$

If just one of $L(\alpha)$ and $R(\alpha)$ is different from **nil**, then (12) degenerates to either $\alpha<_L[L(\alpha)]$ or $\alpha<_L[R(\alpha)]$, respectively. (We observe that $<_L$ is characterized by (12) since, by induction on the number of nodes in the subtree rooted at α, it follows that there is a unique order on $\mathscr{P}_n$ satisfying (12) for all $\alpha\in\mathscr{P}_n$.)

We have shown in [5] that $<_L$ is in fact the *lexicographic order* on $\mathscr{P}_n$ (i.e., with α and γ defined as in Lemma 1, $\alpha<_L\gamma$ if and only if there is some l, $1\leqq l\leqq\min(k, s)$, such that $\alpha_i=\gamma_i$ for $1\leqq i<l$ and $\alpha_l<\gamma_l$); this result can be derived directly from (12) with the aid of Corollaries 1 and 2.

Turning now to M-order, corresponding to (12), we now have

$$[L(\alpha)]<_M\alpha<_M[R(\alpha)]. \tag{13}$$

If either $L(\alpha) = \mathbf{nil}$ or $R(\alpha) = \mathbf{nil}$ then either the left- or right-hand inequality disappears from (13). (As with L-order, we note that $<_M$ is characterized by (13).) Again with the aid of Corollaries 1 and 2, it is straightforward to show that (13) is satisfied if $<_M$ is a lexicographic order in which the underlying collating sequence utilized is $2, 3, 4, \cdots n, 1$ instead of the usual collating sequence $1, 2, 3, \cdots, n$. We thus obtain the following characterization:

Remark 1. If α and γ are defined as in Lemma 1 then $\alpha <_M \gamma$ if and only if there is some l, $1 \leqq l \leqq \min(k, s)$, such that

$$\alpha_i = \gamma_i \qquad \text{for } 1 \leqq i < l,$$

and

$$\text{either} \quad 1 < \alpha_l < \gamma_l \quad \text{or} \quad \alpha_l > \gamma_l = 1.$$

Remark 2. If we define a 1–1 mapping on $\mathscr{P}_n$ by $\alpha \to \alpha^M$, where

$$\alpha^M = (\alpha_1^M, \alpha_2^M, \cdots, \alpha_k^M), \qquad \alpha_i^M = \begin{cases} \alpha_i & \text{if } \alpha_i > 1 \\ n+1 & \text{if } \alpha_i = 1 \end{cases} \quad \text{for } 1 \leqq i \leqq k,$$

then $\alpha <_M \gamma$ if and only if $\alpha^M <_L \gamma^M$.

Finally, we consider P-order. In this case, corresponding to (12), we have

$$[L(\alpha)] <_P [R(\alpha)] <_P \alpha. \tag{14}$$

As expected, this degenerates to $[L(\alpha)] <_P \alpha$ or $[R(\alpha)] <_P \alpha$ if $R(\alpha)$ or $L(\alpha)$ is **nil**. (Again we note that $<_P$ is characterized by (14).) This case is rather more involved than the two preceding cases. However, a careful examination of the conditions required for (14) to hold yields the following characterization:

Remark 3. If α and γ are defined as in Lemma 1 then $\alpha <_P \gamma$ if and only if there is some l, $1 \leqq l \leqq \min(k, s)$, such that

$$\alpha_i = \gamma_i \quad \text{for } 1 \leqq i < l, \tag{15}$$

and either

$$1 < \alpha_{l+1} \leqq \alpha_l < \gamma_l \tag{16}$$

or

$$\alpha_l > \gamma_l \geqq \gamma_{l+1} = 1. \tag{17}$$

It is straightforward to show that the order characterized by Remark 3 is a total order on $\mathscr{P}_n$ and, by using Corollaries 1 and 2, that it satisfies (14). This proves Remark 3, since $<_P$ is characterized by (14).

The following lemma relates $<_P$ to $<_L$ in a similar manner to the relationship of $<_M$ to $<_L$ given by Remark 2.

LEMMA 2. *Let* $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_k)$ *and* $\gamma = (\gamma_1, \gamma_2, \cdots, \gamma_s)$ *be any partitions in* $\mathscr{P}_n$ *and let* $h'_\alpha = max$ (1, *the number of parts of* α *greater than* 1), *with* h'_γ *defined similarly. If a* 1–1 *mapping on* $\mathscr{P}_n$ *is defined by* $\alpha \to \alpha^P$, *where*

$$\alpha^P = (\alpha_1^P, \alpha_2^P, \cdots, \alpha_k^P), \qquad \alpha_i^P = \begin{cases} \alpha_i & \text{if } i \neq h'_\alpha \\ 2n - \alpha_i & \text{if } i = h'_\alpha \end{cases} \quad \text{for } 1 \leqq i \leqq k, \tag{18}$$

then $\alpha <_P \gamma$ *if and only if* $\alpha^P <_L \gamma^P$.

(We note that (18) is just one of several similar mappings for which this result holds.)

Proof. Let us assume that $\alpha <_P \gamma$. Then by Remark 3, there exists l, $1 \leqq l \leqq \min(k, s)$, such that (15) and either (16) or (17) hold. Suppose first that (16) holds. We then have $h'_\alpha \geqq l+1$ and $h'_\gamma \geqq l$. It thus follows that $\alpha_l^P = \alpha_l < \gamma_l \leqq \gamma_l^P$; this together with (15) implies that $\alpha^P <_L \gamma^P$. Next suppose instead that (17) holds. We now have that $h'_\alpha \geqq l$ and $h'_\gamma = l-1$ or l. If $h'_\gamma = l-1$ then

$$\alpha_{l-1}^P = \alpha_{l-1} = \gamma_{l-1} < 2n - \gamma_{l-1} = \gamma_{l-1}^P,$$

so $\alpha^P <_L \gamma^P$. Similarly, if $h'_\gamma = l$ then $\alpha_l^P \leqq 2n - \alpha_l < 2n - \gamma_l = \gamma_l^P$, so again $\alpha^P <_L \gamma^P$.

Conversely, if $\alpha^P <_L \gamma^P$ then $\alpha <_P \gamma$ since $<_P$ is a total order on $\mathscr{P}_n$ and $<_L$ is a total order on the image of $\mathscr{P}_n$ under the transformation $\alpha \to \alpha^P$. □

The following remark follows directly from Remarks 1 and 3.

Remark 4. If l, the least value of i for which $\alpha_i \neq \gamma_i$, is such that $\alpha_{l+1} > 1$ and $\gamma_{l+1} > 1$, then the relationship between α and γ is the same for each of the orders $<_L$, $<_M$ and $<_P$.

4. Algorithms for generating $\mathscr{P}_n$. We now describe algorithms for obtaining the successor of an arbitrary partition $\alpha \in \mathscr{P}_n$ for the various orders. $\mathscr{P}_n$ may then be generated by repeated application of any of these algorithms. For any order, we define the successor of the last partition to be **nil**. Moreover, for symmetry, it is convenient to define the successor of **nil** to be the first partition in the relevant order, although the algorithms do not check for this case.

4.1. *L*-order, *M*-order and *P*-order. In [6] concise algorithms were obtained for finding the preorder, inorder and postorder successors of a node in a triply linked binary tree (i.e. one having additional pointers from each node to its father). If $L(v)$, $R(v)$ and $F(v)$ denote the left son, right son and father of the node v, respectively, then these algorithms take the form

```
procedure Presucc(v): result is v
  if L(v) ≠ nil then v ← L(v)
                else while R(v) = nil do Ascend(v)
                       v ← R(v)
procedure Insucc(v): result is v
  if R(v) ≠ nil then Descend(v)
                else Ascend(v)
procedure Postsucc(v): result is v
  if isrightson(v) then v ← F(v)
                   else v ← F(v)
                        while R(v) ≠ nil do Descend(v),
```

where the procedures *Descend* and *Ascend* are defined by

```
procedure Descend(v)                 procedure Ascend(v)
  v ← R(v)                             while isrightson(v) do v ← F(v)
  while L(v) ≠ nil do v ← L(v)         v ← F(v)
```

and the predicate *isrightson* is defined by

$$isrightson(v) \equiv (v = R(F(v))).$$

(Algorithms for traversing triply linked binary trees presented in a rather different form from those above may be found in [1].)

We now turn to the case when the tree to be traversed is T_n. First we give formal definitions of L and R in terms of $\beta\mu$-notation. However, in order to reduce the

number of cases in the various algorithms derived in this section to manageable proportions, it is convenient to express the resulting partitions using a relaxed form of $\beta\mu$-notation in which (9) is replaced by

$$\beta_1 \geqq \beta_2 \geqq \cdots \geqq \beta_\Delta > 1, \qquad m_i \geqq 0 \quad \text{for } 1 \leqq i \leqq \Delta, \quad \mu \geqq 0. \tag{19}$$

We refer to this relaxed form as *nonstandard* $\beta\mu$-notation to distinguish it from the *standard* $\beta\mu$-notation defined in § 2. For the remainder of this section *we assume that* $n \geqq 4$, although some of the algorithms are also applicable for smaller values of n.

From (6) we immediately see that

$$L(\alpha) = \begin{cases} \Sigma_\Delta 2^1 1^{\mu-2} & \text{if } \mu \geqq 2, \\ \textbf{nil} & \text{otherwise,} \end{cases}$$

$$R(\alpha) = \begin{cases} \Sigma_{\Delta-1}(\beta_\Delta+1)^1 1^{\mu-1} & \text{if } \mu \geqq 1 \wedge m_\Delta = 1, \\ \textbf{nil} & \text{otherwise.} \end{cases}$$

Also, if $\alpha \neq 1^n$, *isrightson*(α) is true if and only if $\beta_\Delta > 2$, in which case

$$F(\alpha) = R^{-1}(\alpha) = \Sigma_{\Delta-1}\beta_\Delta^{m_\Delta-1}(\beta_\Delta-1)^1 1^{\mu+1}.$$

It is now straightforward to verify that the effects of the procedures *Ascend* and *Descend*, for $\alpha \in \mathcal{P}_n - \{1^n\}$, are

$$\textit{Ascend}(\alpha)\colon \alpha \leftarrow \Sigma_{\Delta-1}\beta_\Delta^{m_\Delta-1} 1^{\mu+\beta_\Delta}, \tag{20}$$

$$\textit{Descend}(\alpha)\colon \alpha \leftarrow \Sigma_{\Delta-1}(\beta_\Delta+1)^1 2^{\kappa-1} 1^\lambda, \tag{21}$$

where, for brevity, we introduce the notation $\kappa = \lfloor(\mu+1)/2\rfloor$, $\lambda = (\mu+1)$ **mod** 2.

Notes. (a) *Descend*(α) is only defined if $R(\alpha) \neq$ **nil**.

(b) In the three successor algorithms, *Ascend*(α) is only ever invoked when $R(\alpha) =$ **nil** and *Descend*(α) when $R(\alpha) \neq$ **nil**.

Using the above results, we now derive versions of *Presucc*, *Insucc* and *Postsucc* for $v = \alpha \in T_n$. We call these *L-succ*, *M-succ* and *P-succ*, respectively.

L-succ. When $\mu \geqq 2$ the result is immediate. If either $\mu = 1$ or if $\mu = 0$ and $m_\Delta > 1$, then *Ascend* is performed $m_\Delta - 1$ times. However, if $\mu = 0$ and $m_\Delta = 1$ then, in general, $\Delta > 1$ and *Ascend* is performed $m_{\Delta-1}$ times; when $\Delta = 1$ then α is n^1 which has no preorder successor. We thus obtain

L-succ$(\alpha) =$ **cond**

(L1) $\mu \geqq 2 \rightarrow \Sigma_\Delta 2^1 1^{\mu-2}$

(L2) $\mu = 1 \vee m_\Delta > 1 \rightarrow \Sigma_{\Delta-1}(\beta_\Delta+1)^1 1^{\mu+\beta_\Delta(m_\Delta-1)-1}$

(L3) $\Delta > 1 \rightarrow \Sigma_{\Delta-2}(\beta_{\Delta-1}+1)^1 1^{\beta_{\Delta-1}(m_{\Delta-1}-1)+\beta_{\Delta-1}}$

(L4) **true** $\rightarrow$ **nil**

end.

M-succ. On using (20) and (21) and the definition of *Insucc*, we obtain

M-succ$(\alpha) =$ **cond**

(M1) $\mu \geqq 1 \wedge m_\Delta = 1 \rightarrow \Sigma_{\Delta-1}(\beta_\Delta+1)^1 2^{\kappa-1} 1^\lambda$

(M2) $\Delta > 0 \rightarrow \Sigma_{\Delta-1}\beta_\Delta^{m_\Delta-1} 1^{\mu+\beta_\Delta}$

(M3) **true** $\rightarrow$ **nil**

end.

P-succ. If $\beta_\Delta > 2$, then the result is immediate. Otherwise, if $\beta_\Delta = 2$, *Descend* is performed zero or more times, beginning with the partition $F(\alpha) = \Sigma_{\Delta-1}2^{m_\Delta - 1}1^{\mu+2}$, until this partition is transformed into a partition α^* for which $R(\alpha^*) = \textbf{nil}$. If $m_\Delta > 2$ then $R(F(\alpha)) = \textbf{nil}$, so *Descend* is not performed; this is also the case if $m_\Delta = 1$ and either $\Delta = 1$ or $m_{\Delta-1} > 1$. Otherwise *Descend* is performed at least once and the first application transforms $F(\alpha)$ to α^+, where

$$\alpha^+ = \begin{cases} \Sigma_{\Delta-1}3^1 2^\kappa 1^\lambda & \text{if } m_\Delta = 2, \\ \Sigma_{\Delta-2}(\beta_{\Delta-1}+1)^1 2^\kappa 1^\lambda & \text{if } m_\Delta = 1. \end{cases}$$

Now $R(\alpha^+) = \textbf{nil}$ unless $\kappa \leqq 1$ and $\lambda = 1$, i.e. unless $\mu = 0$ or 2. For $R(\alpha^+)$ to be nonnull in the case $\mu = 0$, we also require $\beta_{\Delta-1} > 3$ if $m_\Delta = 2$ or $\beta_{\Delta-2} > \beta_{\Delta-1} + 1$ if $m_\Delta = 1$. In these cases *Descend* now yields a partition α^* with no parts of size 1, so $R(\alpha^*) = \textbf{nil}$. We therefore see that *Descend* is performed at most twice and thus we obtain

P-succ(α) = **cond**

(P1) $\Delta = 0$ → **nil**

(P2) $\beta_\Delta > 2$ → $\Sigma_{\Delta-1}\beta_\Delta^{m_\Delta - 1}(\beta_\Delta - 1)^1 1^{\mu+1}$

(P3) $m_\Delta > 2 \vee (m_\Delta = 1 \wedge m_{\Delta-1} > 1)$ → $\Sigma_{\Delta-1}2^{m_\Delta-1}1^{\mu+2}$

$m_\Delta = 2$ → **cond**

(P4) $\mu = 0 \wedge \beta_{\Delta-1} > 3$ → $\Sigma_{\Delta-1}4^1$

(P5) $\mu = 2$ → $\Sigma_{\Delta-1}3^2$

(P6) **true** → $\Sigma_{\Delta-1}3^1 2^\kappa 1^\lambda$

end

$m_\Delta = 1$ → **cond**

(P7) $\mu = 0 \wedge \beta_{\Delta-2} > \beta_{\Delta-1} + 1$ → $\Sigma_{\Delta-2}(\beta_{\Delta-1}+2)^1$

(P8) $\mu = 2$ → $\Sigma_{\Delta-2}(\beta_{\Delta-1}+1)^1 3^1$

(P9) **true** → $\Sigma_{\Delta-2}(\beta_{\Delta-1}+1)^1 2^\kappa 1^\lambda$

end

end.

4.2. The three reverse orders. The traversal algorithms for an arbitrary binary tree in the three reverse orders can be obtained immediately by interchanging the roles of left and right in *Presucc*, *Insucc* and *Postsucc*. This yields the following procedures.

```
procedure Prepred(v): result is v
  if isleftson(v) then v ← F(v)
                  else v ← F(v)
                       while L(v) ≠ nil do R-Descend(v)

procedure Inpred(v): result is v
  if L(v) ≠ nil then R-Descend(v)
                else R-Ascend(v)
```

procedure *Postpred*(v): **result is** v
  if $R(v) \neq$ **nil then** $v \leftarrow R(v)$
    else while $L(v) =$ **nil do** *R-Ascend*(v)
      $v \leftarrow L(v)$,

where the procedures *R-Descend* and *R-Ascend* are defined by

procedure *R-Descend*(v)
  $v \leftarrow L(v)$
  while $R(v) \neq$ **nil do** $v \leftarrow R(v)$

procedure *R-Ascend*(v)
  while *isleftson*(v) **do** $v \leftarrow F(v)$
  $v \leftarrow F(v)$

and the predicate *isleftson* is defined by

$$isleftson(v) \equiv (v = L(F(v))).$$

Note. *Prepred* is obtained by interchanging the roles of left and right in *Postsucc*, and similarly for *Postpred* and *Presucc*.

When $v = \alpha \in T_n$ we call these predecessor procedures *L-pred*, *M-pred* and *P-pred*, respectively.

It is easy to see that *isleftson*(α) is true if and only if $\beta_\Delta = 2$, in which case

$$F(\alpha) = L^{-1}(\alpha) = \Sigma_{\Delta-1} 2^{m_\Delta - 1} 1^{\mu+2}.$$

Thus the effects of the procedures *R-Ascend* and *R-Descend* for $\alpha \in \mathscr{P}_n$ are

$$(22) \quad R\text{-}Ascend(\alpha)\colon \alpha \leftarrow \begin{cases} \Sigma_{\Delta-2}\beta_{\Delta-1}^{m_\Delta - 1}(\beta_{\Delta-1}-1)^1 1^{\mu+1+2m_\Delta} & \text{if } \beta_\Delta = 2 \wedge \Delta > 1, \\ \Sigma_{\Delta-1}\beta_\Delta^{m_\Delta-1}(\beta_\Delta - 1)^1 1^{\mu+1} & \text{if } \beta_\Delta > 2 \wedge \Delta > 0, \end{cases}$$

$$(23) \quad R\text{-}Descend(\alpha)\colon \alpha \leftarrow \begin{cases} \Sigma_\Delta \mu^1 & \text{if } \beta_\Delta > \mu > 1, \\ \Sigma_{\Delta-1}\beta_\Delta^{m_\Delta+1} 1^{\mu-\beta_\Delta} & \text{if } \beta_\Delta \leqq \mu. \end{cases}$$

Notes. (a) *R-Descend*(α) is only defined if $L(\alpha) \neq$ **nil**.

(b) In the three predecessor algorithms, *R-Ascend*(α) is only ever invoked when $L(\alpha) =$ **nil** and *R-Descend*(α) when $L(\alpha) \neq$ **nil**.

(c) *R-Ascend*(α) is not defined when $\Delta = 0$ or when $\beta_1 = 2$, but in these cases $L(\alpha)$ is **nil** only when $\mu = 0$ or 1, i.e. when α is the first partition in both M-order and P-order.

L-pred. When $\beta_\Delta = 2$ the result is immediate. Otherwise, starting with $F(\alpha)$, *R-Descend* is initially performed $\lfloor (\mu+1)/(\beta_\Delta - 1) \rfloor$ times, and if $(\mu+1)$ **mod** $(\beta_\Delta - 1) > 1$ it is then performed one further time. We thus obtain

L-pred$(\alpha) =$ **cond**

(RL1) $\Delta = 0 \rightarrow$ **nil**

(RL2) $\beta_\Delta = 2 \rightarrow \Sigma_{\Delta-1} 2^{m_\Delta - 1} 1^{\mu+2}$

(RL3) $(\beta_\Delta - 1) | (\mu+1) \rightarrow \Sigma_{\Delta-1}\beta_\Delta^{m_\Delta-1}(\beta_\Delta-1)^{1+(\mu+1)/(\beta_\Delta-1)}$

(RL4) $(\beta_\Delta - 1) \nmid (\mu+1) \rightarrow \Sigma_{\Delta-1}\beta_\Delta^{m_\Delta-1}(\beta_\Delta-1)^{1+\lfloor(\mu+1)/(\beta_\Delta-1)\rfloor} ((\mu+1)$ **mod** $(\beta_\Delta-1))^1$

end.

M-pred. On using (22) and (23) and the definition of *Inpred*, we obtain

$M\text{-}pred(\alpha) =$ **cond**

$\mu \geqq 2 \rightarrow$ **cond**

(RM1) $\mu < \beta_\Delta \rightarrow \Sigma_\Delta \mu^1$

(RM2) $\mu \geqq \beta_\Delta \rightarrow \Sigma_{\Delta-1} \beta_\Delta^{m_\Delta+1} 1^{\mu-\beta_\Delta}$

end

$\mu < 2 \rightarrow$ **cond**

(RM3) $\beta_\Delta > 2 \rightarrow \Sigma_{\Delta-1} \beta_\Delta^{m_\Delta-1} (\beta_\Delta - 1)^1 1^{\mu+1}$

(RM4) $\Delta > 1 \rightarrow \Sigma_{\Delta-2} \beta_{\Delta-1}^{m_{\Delta-1}-1} (\beta_{\Delta-1} - 1)^1 1^{\mu+1+2m_\Delta}$

(RM5) **true** $\rightarrow$ **nil**

end

end.

P-pred. If $\mu \geqq 1$ and $m_\Delta = 1$ then the result is immediate. Similarly, if $\mu \geqq 2$ then *R-Ascend* is not performed and the result is immediate. Otherwise, since each *R-Ascend* increases the number of parts of size unity, after at most two invocations of *R-Ascend* the resulting partition will have a left son. Thus *R-Ascend* will be performed either just once if $\beta_\Delta = 2$ or $\mu = 1$, or twice if $\beta_\Delta > 2$ and $\mu = 0$. The result then follows on applying L to the resulting partition. We thus obtain

$P\text{-}pred(\alpha) =$ **cond**

(RP1) $\mu \geqq 1 \wedge m_\Delta = 1 \rightarrow \Sigma_{\Delta-1} (\beta_\Delta + 1)^1 1^{\mu-1}$

(RP2) $\mu \geqq 2 \rightarrow \Sigma_\Delta 2^1 1^{\mu-2}$

(RP3) $\beta_\Delta = 2 \wedge \Delta > 1 \rightarrow \Sigma_{\Delta-2} \beta_{\Delta-1}^{m_{\Delta-1}-1} (\beta_{\Delta-1} - 1)^1 2^1 1^{\mu-1+2m_\Delta}$

(RP4) $\beta_\Delta = 2 \rightarrow$ **nil**

(RP5) $\mu = 1 \rightarrow \Sigma_{\Delta-1} \beta_\Delta^{m_\Delta-1} (\beta_\Delta - 1)^1 2^1$

(RP6) $\beta_\Delta = 3 \wedge m_\Delta > 1 \rightarrow \Sigma_{\Delta-1} 3^{m_\Delta-2} 2^2 1^2$

(RP7) $\beta_\Delta = 3 \rightarrow \Sigma_{\Delta-2} \beta_{\Delta-1}^{m_{\Delta-1}-1} (\beta_{\Delta-1} - 1)^1 2^1 1^2$

(RP8) $\beta_\Delta > 3 \rightarrow \Sigma_{\Delta-1} \beta_\Delta^{m_\Delta-1} (\beta_\Delta - 2)^1 2^1$

end.

It is quite easy to subdivide certain of the cases in any of the above algorithms, by means of additional tests, so that they yield the resulting partitions in standard $\beta\mu$-notation. Similarly, it is straightforward to rewrite the algorithms for β-notation: in some cases this will necessitate further tests in order to separate the cases $\mu = 0$ and $\mu > 0$.

In order to generate $\mathscr{P}_n$ in any of the three orders, or the corresponding reverse orders, we can use a scheme of the form

$\alpha \leftarrow$ *firstpartition*
while $\alpha \neq$ **nil do** $Process(\alpha)$; $\alpha \leftarrow Succ(\alpha)$.

The value of *firstpartition* is 1^n for *L-succ* and n^1 for *L-pred*; for *M-succ* and *P-succ* the corresponding value of *firstpartition* is $2^{\lfloor n/2 \rfloor} 1^{n \bmod 2}$ and for *M-pred* and *P-pred* it is 1^n.

4.3. The relationship between the successor and predecessor algorithms. It is possible to show that, for any of the three orders, the corresponding successor and predecessor algorithms, as well as being functional inverses, are also, in a certain sense, structural inverses. We now explain briefly the nature of this relationship and illustrate this by stating the results for one case. Fuller details of how to derive such results may be found in § 4 of [7].

We first partition $\mathscr{P}_n$ on the outcome of the various tests in the algorithm under consideration. As an illustration we consider the algorithm *M-pred*: in this case we obtain five disjoint subsets, which we denote by [RM1], [RM2], [RM3], [RM4] and [RM5], using the numbering of the different cases in the algorithm. Thus, for example,

$$[\text{RM3}] = \mathscr{P}_n[0 \leqq \mu \leqq 1, \beta_\Delta > 2].$$

Considering similarly the subsets corresponding to *M-succ*, it is not difficult to show that

$$M\text{-}succ\ [\text{M1}] = [\text{RM3}] \cup [\text{RM4}],$$

$$M\text{-}succ\ [\text{M2}] = [\text{RM1}] \cup [\text{RM2}],$$

where the extension of the successor and predecessor functions to *sets* of partitions is defined in the usual way. If we now define

$$[\text{M1a}] = [\text{M1}] \cap \mathscr{P}_n[\mu \leqq 2], \qquad [\text{M1b}] = [\text{M1}] \cap \mathscr{P}_n[\mu > 2],$$

$$[\text{M2a}] = [\text{M2}] \cap \mathscr{P}_n[m_\Delta = 1], \qquad [\text{M2b}] = [\text{M2}] \cap \mathscr{P}_n[m_\Delta > 1],$$

the results in Table 1 then follow in a simlar fashion. Corresponding results can be obtained for the other two pairs of algorithms. Moreover, similar results also exist

TABLE 1

M-succ[**nil**] = [RM5]	*M-succ*[M2a] = [RM1]
M-succ[M1a] = [RM3]	*M-succ*[M2b] = [RM2]
M-succ[M1b] = [RM4]	*M-succ*[M3] = **nil**.

for each of the corresponding pairs of standard $\beta\mu$-notation algorithms as well as for the β-notation algorithms. (Details of the correspondence between *L-succ* and *L-pred* for the β-notation algorithms are given in § 4 of [7].)

5. Asymptotic analysis. As we shall see, the asymptotic behaviour (i.e. for large n) of each of the algorithms in § 4 is determined by the single case which, for large n, will be selected for *almost all* partitions $\alpha \in \mathscr{P}_n$. For example, if we consider the cases of *L-succ*, it is easy to show that $\#[\text{L1}] = P_{n-2}$; thus the proportion of the P_n possible partitions $\alpha \in \mathscr{P}_n$ for which (L1) will be selected is $P_{n-2}/P_n = 1 - O(n^{-1/2})$ by (11). If an algorithm has such a case, which is selected for $(1 - o(1))P_n$ of the partitions $\alpha \in \mathscr{P}_n$, we call this case the (asymptotically) *dominant* case. Accordingly, in order to investigate the asymptotic behaviour of the algorithms, it is sufficient to consider solely the dominant case provided one exists.

In employing the algorithms, one would obviously require that the result be expressed in either β-notation or standard $\beta\mu$-notation. We choose to use β-notation, although very similar corresponding results can be obtained in a similar fashion for

standard $\beta\mu$-notation. The reason for our choice is that the former notation is, in some sense, both more symmetric and natural than $\beta\mu$-notation and, moreover, almost all references in the literature to multiplicity notations for partitions use β-notation (see [5], [10] and [11]).

For the purposes of our asymptotic analysis, it is not necessary to express the whole of each algorithm in β-notation: it is sufficient to identify the dominant case (which will, in general, be a subcase of the dominant case of the corresponding nonstandard $\beta\mu$-notation algorithm) and express both the tests which define it as well as the result using β-notation. In order to identify the dominant case of any of the algorithms (for any of the notations), we observe that, for large n, almost all partitions in $\mathscr{P}_n$ have several parts of size 1 and several parts of size 2. In fact, by using the Hardy–Ramanujan asymptotic formulae for P_n (see [2] and [8]), it is possible to show that, for any $\theta < 1/2$, the number of partitions in $\mathscr{P}_n$ having at least n^θ parts of size 1 and at least n^θ parts of size 2 is $(1-o(1))P_n$.

In addition, we obtain counting formulae for the number of partitions $\alpha \in \mathscr{P}_n$ for which the dominant case is selected. In this context it is convenient to introduce the following notation: if Q is a set of partitions each of which contains the partition $\hat{\alpha}$ as a subpartition, then $Q/\hat{\alpha}$ denotes the set of partitions obtained by removing the subpartition $\hat{\alpha}$ from each partition in Q. Thus, obviously, $\#(Q/\hat{\alpha}) = \#Q$.

We now consider each of the algorithms in turn.

L-succ. By the remarks above, the dominant case of *L-succ* is (L1). In order to transform (L1) into the dominant case of the corresponding β-notation algorithm, the test $\mu \geqq 2$ must be replaced by $\beta_d = 1 \wedge m_d \geqq 2$; in addition, we need to test whether $m_d = 2$ and, when $d > 1$, whether $\beta_{d-1} = 2$. It is clear that the dominant case will satisfy $m_d > 2$ and $\beta_{d-1} = 2$. (We note that $\beta_{d-1} \neq 2$ if $d = 1$; moreover, if $\beta_{d-1} = 2$, then we must have $\beta_d = 1$). The tests which determine the dominant case and the successor partition for this case can therefore be expressed in the form

$$\text{(L1*)} \qquad m_d > 2 \wedge \beta_{d-1} = 2 \rightarrow \Sigma_{d-2} 2^{m_{d-1}+1} 1^{m_d-2}.$$

Thus

$$[\text{L1*}] = \mathscr{P}_n[\beta_d = 1, m_d > 2, d > 1, \beta_{d-1} = 2].$$

It is now easy to verify that $[\text{L1*}]/2^1 1^3 = \mathscr{P}_{n-5}$, so $\#[\text{L1*}] = P_{n-5}$.

For the remaining algorithms, we abbreviate the derivation of the dominant cases of the corresponding β-notation algorithms, since they are obtained in a similar manner to that for *L-succ*.

M-succ. The dominant case of *M-succ* is (M2) and, corresponding to (L1*), the dominant case for the β-notation algorithm is

$$\text{(M2*)} \qquad \beta_d = 1 \wedge d > 1 \wedge m_{d-1} > 1 \rightarrow \Sigma_{d-2} \beta_{d-1}^{m_{d-1}-1} 1^{m_d+\beta_{d-1}}.$$

We now have

$$\begin{aligned}[\text{M2*}] &= \mathscr{P}_n[\beta_d = 1, d > 1, m_{d-1} > 1] \\ &= \mathscr{P}_n[\beta_d = 1] - \mathscr{P}_n[\beta_d = 1, d > 1, m_{d-1} = 1] - \{1^n\}.\end{aligned}$$

Consider adding 1 to the single part of size β_{d-1} for each partition in $\mathscr{P}_n[\beta_d = 1, d > 1, m_{d-1} = 1]$; this gives a bijection between this set and $\mathscr{P}_{n+1}[\beta_d = 1, d > 1, \beta_{d-1} > 2]$. Hence, since

$$\mathscr{P}_{n+1}[\beta_d = 1, d > 1, \beta_{d-1} > 2] = \mathscr{P}_{n+1}[\beta_d = 1] - \mathscr{P}_{n+1}[\beta_d = 1, d > 1, \beta_{d-1} = 2] - \{1^{n+1}\},$$

we see that

$$\#[\mathrm{M2}^*] = P_n[\beta_d = 1] - P_{n+1}[\beta_d = 1] + P_{n+1}[\beta_d = 1, d > 1, \beta_{d-1} = 2].$$

As it is easy to see that

$$\mathscr{P}_n[\beta_d = 1]/1^1 = \mathscr{P}_{n-1}, \qquad \mathscr{P}_{n+1}[\beta_d = 1]/1^1 = \mathscr{P}_n,$$
$$\mathscr{P}_{n+1}[\beta_d = 1, d > 1, \beta_{d-1} = 2]/2^1 1^1 = \mathscr{P}_{n-2},$$

it follows that $\#[\mathrm{M2}^*] = P_{n-2} + P_{n-1} - P_n$.

P-succ. The dominant case of *P-succ* is (P3) and for the β-notation algorithm the corresponding dominant case is

$$\beta_{d-1} = 2 \wedge m_{d-1} > 2 \to \Sigma_{d-2} 2^{m_{d-1}-1} 1^{m_d+2}. \tag{P3*}$$

Thus

$$[\mathrm{P3}^*] = \mathscr{P}_n[\beta_d = 1, d > 1, \beta_{d-1} = 2, m_{d-1} > 2],$$

so $\#[\mathrm{P3}^*] = P_{n-7}$ since $[\mathrm{P3}^*]/2^3 1^1 = \mathscr{P}_{n-7}$.

L-pred. The dominant case of *L-pred* is (RL2) and, correspondingly, for the β-notation algorithm we obtain

$$\beta_{d-1} = 2 \wedge m_{d-1} > 1 \to \Sigma_{d-2} 2^{m_{d-1}-1} 1^{m_d+2}. \tag{RL2*}$$

Thus

$$[\mathrm{RL2}^*] = \mathscr{P}_n[\beta_d = 1, d > 1, \beta_{d-1} = 2, m_{d-1} > 1],$$

and consequently, since $[\mathrm{RL2}^*]/2^2 1^1 = \mathscr{P}_{n-5}$, it follows that $\#[\mathrm{RL2}^*] = P_{n-5}$.

M-pred. The dominant case of *M-pred* is (RM2) and, correspondingly, for the β-notation algorithm we obtain

$$\beta_d = 1 \wedge m_d > \beta_{d-1} \to \Sigma_{d-2} \beta_{d-1}^{m_{d-1}+1} 1^{m_d - \beta_{d-1}}. \tag{RM2*}$$

We thus have

$$[\mathrm{RM2}^*] = \mathscr{P}_n[\beta_d = 1, d > 1, m_d > \beta_{d-1}].$$

Removing β_{d-1} parts of size 1 and replacing them by one part of size β_{d-1} for each partition in $[\mathrm{RM2}^*]$ yields a bijection between this set and $\mathscr{P}_n[\beta_d = 1, d > 1, m_{d-1} > 1]$. This latter set is $[\mathrm{M2}^*]$, hence $\#[\mathrm{RM2}^*] = P_{n-2} + P_{n-1} - P_n$.

P-pred. The dominant case of *P-pred* is (RP2) and, correspondingly, for the β-notation algorithm we obtain

$$m_d > 2 \wedge \beta_{d-1} = 2 \wedge m_{d-1} > 1 \to \Sigma_{d-2} 2^{m_{d-1}+1} 1^{m_d-2}. \tag{RP2*}$$

Thus

$$[\mathrm{RP2}^*] = \mathscr{P}_n[\beta_d = 1, m_d > 2, d > 1, \beta_{d-1} = 2, m_{d-1} > 1],$$

and it is easy to see that $[\mathrm{RP2}^*]/2^2 1^3 = \mathscr{P}_{n-7}$, so $\#[\mathrm{RP2}^*] = P_{n-7}$.

It is interesting to observe that the dominant cases above for the β-notation versions of the successor and predecessor algorithms are functional inverses in the sense of § 4.3, viz.

$$L\text{-}succ[\mathrm{L1}^*] = [\mathrm{RL2}^*],\ M\text{-}succ[\mathrm{M2}^*] = [\mathrm{RM2}^*],\ P\text{-}succ[\mathrm{P3}^*] = [\mathrm{RP2}^*].$$

We also see that

$$[\text{RP2*}] \subseteq [\text{L1*}] \subseteq [\text{RM2*}],$$

$$[\text{P3*}] \subseteq [\text{RL2*}] \subseteq [\text{M2*}].$$

Moreover, (RP2*) is a subcase of (L1*) and (RM2*) since, when $\beta_{d-1} = 2$, the right-hand sides of these three cases are all identical. Furthermore, since (RP2*) is a *dominant* case, it follows that for *almost all* partitions $\alpha \in \mathcal{P}_n$,

$$L\text{-}succ(\alpha) = M\text{-}pred(\alpha) = P\text{-}pred(\alpha).$$

Similar comments hold for the other three algorithms.

6. Discussion. In order to accurately compare (for the purpose of generating $\mathcal{P}_n$) the asymptotic performance of the six successor and predecessor algorithms, it would be necessary to know the computational costs of the basic computer operations involved. However, by examining the dominant cases of each of the algorithms (viz. (L1*), (M2*), (P3*), (RL2*), (RM2*), (RP2*)), it appears that, under any reasonable assumptions for the computational costs, the asymptotic costs will be fairly similar for all six algorithms; moreover, the ratio of the cost of the most expensive algorithm to that of the least expensive is unlikely to exceed 1.5.

It should be borne in mind, however, that even for quite large values of n the proportion of partitions for which the dominant case is selected will not be particularly large, e.g., for $n = 200$, $P_n \sim 4 \times 10^{12}$ but the corresponding proportions for the *L*-, *M*- and *P*-orders are 0.65, 0.76 and 0.55, respectively. The reason for this is that (11) tends to 1 rather slowly as n increases. In order to examine the behaviour of the algorithms for moderate values of n, it would be necessary to obtain counting formulae for each of the nondominant cases of the algorithms. Although such formulae have been derived for *L-succ* and *L-pred* in [7], and similar techniques can be used to obtain corresponding formulae for the other algorithms, we refrain from pursuing this further here because of the large number of cases involved, e.g. the β-notation version of *P-pred* has over two dozen cases. As well as using such counting formulae to compare the performance of the different algorithms, they could also be used to optimize the ordering of the tests in each of the algorithms in order to reduce the average number of tests.

We note that all six algorithms of § 4 are loop-free [4]. However, if the algorithms are rewritten for partitions represented in α-notation (i.e. using a k-tuple instead of a multiplicity representation), then none of the resulting algorithms are loop-free. Moreover, it is straightforward to show that no α-notation algorithm for generating successors or predecessors in any of the three orders can be loop-free. In [5] we observed that if a mixed representation is used, with parts greater than 2 represented using α-notation together with the (possibly zero) multiplicities of parts of size 2 and 1, e.g. the partition $6^3 4^1 3^2 2^4 1^5$ is represented as the triple $\langle (6, 6, 6, 4, 3, 3), 4, 5 \rangle$, then the corresponding *M-succ* algorithm is loop-free. It is easy to see that this is also the case for the corresponding *P-succ*, *M-pred* and *P-pred* algorithms, although the *L-succ* and *L-pred* algorithms cannot be loop-free for this representation.

Another possible representation is to use α-notation for parts greater than 1 together with just the multiplicity of parts of size 1, e.g. the partition $6^3 4^1 3^2 2^4 1^5$ is represented as the pair $\langle (6, 6, 6, 4, 3, 3, 2, 2, 2, 2), 5 \rangle$. Again, with this representation, the *L-succ* and *L-pred* algorithms cannot be loop-free and, at first sight, the other four algorithms are also not loop-free. However, under suitable assumptions, these algorithms can be rewritten in a loop-free form. For the *M-succ* and *P-succ* algorithms,

if the parts greater than 1 are stored in an array a_i, $1 \leqq i \leqq \lfloor n/2 \rfloor$, and the index h specifies the number of parts greater than 1, then we require that, at any time, the unused elements of the array, viz. $a_{h+1}, a_{h+2}, \cdots, a_{\lfloor n/2 \rfloor}$, all contain the value 2 and that these elements are not changed between successive invocations of *M-succ* or *P-succ*. For *M-pred* and *P-pred*, however, we only require that an additional count be kept of the number of parts of size 2. It is then straightforward to obtain loop-free implementations of the four algorithms under these assumptions.

In addition to the algorithms of § 4, it is interesting to note that none of the algorithms in the literature (see the references in § 1 of [5] and [7]) succumb to a loop-free implementation for partitions represented in α-notation, i.e. the standard k-tuple representation. It is at present an interesting open question whether or not there exists some loop-free algorithm for generating $\mathscr{P}_n$ in α-notation.

Note added in proof. We thank one of the referees for pointing out that the application of Algorithm NEXT to Family 6 [10, 2nd ed. 1978, p. 105] can be viewed as a binary tree traversal algorithm. Unlike T_n, in this binary tree (which is not explicitly defined but is implicit in the description of the algorithm) only the *terminal* nodes represent partitions, and these are scattered among many levels of the tree (in fact, approximately between levels $2\sqrt{n}$ and n). The algorithm, however, is not loop-free and, moreover, it can be shown that the *average* number of steps per partition in generating $\mathscr{P}_n$ is $\Omega(\phi(n))$ for any function ϕ for which $\phi(n) = o(\sqrt{n})$.

REFERENCES

[1] E. N. ADAMS III, *Another representation of binary tree traversal*, Inform. Process. Lett., 2 (1973), pp. 52–54.
[2] G. E. ANDREWS, *The Theory of Partitions*, Addison-Wesley, Reading, MA, 1976.
[3] C. BERGE, *Principles of Combinatorics*, Academic Press, New York, 1971.
[4] G. EHRLICH, *Loopless algorithms for generating permutations, combinations, and other combinatorial configurations*, J. Assoc. Comput. Mach., 20 (1973), pp. 500–513.
[5] T. I. FENNER AND G. LOIZOU. *A binary tree representation and related algorithms for generating integer partitions*, Comput. J., 23 (1980), pp. 332–337.
[6] ———, *A note on traversal algorithms for triply linked binary trees*, BIT, 21 (1981), pp. 153–156.
[7] ———, *An analysis of two related loop-free algorithms for generating integer partitions*, Acta Inform., 16 (1981), pp. 237–252.
[8] G. H. HARDY AND S. RAMANUJAN, *Asymptotic formulae in combinatory analysis*, Proc. London Math. Soc., 17 (1918), pp. 75–115.
[9] D. E. KNUTH, *The Art of Computer Programming, Vol.* 1, Addison-Wesley, Reading, MA, 1973.
[10] A. NIJENHUIS AND H. S. WILF, *Combinatorial Algorithms*, Academic Press, New York, 1975.
[11] E. M. REINGOLD, J. NIEVERGELT AND N. DEO, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0011 $01.25/0

POSITIVE RELATIVIZATIONS OF COMPLEXITY CLASSES*

ALAN L. SELMAN,† XU MEI-RUI‡ AND RONALD V. BOOK§

Abstract. Due to the work of Baker, Gill and Solovay [SIAM J. Comput., 4(1975), pp. 431–442] and others, it has become a paradigm that important open questions about complexity classes do not relativize. This paper develops restrictions of the standard oracle machine model for which, in contrast, positive inclusion relationships do relativize. Our results are obtained by uniform simulation techniques. As a consequence, the new oracle machine models exhibit the same computational power as do the corresponding nonrelativized devices.

Key words. complexity classes, polynomial time, polynomial space, determinism vs. nondeterminism, positive relativizations, oracle Turing machines

1. Introduction.

Baker, Gill and Solovay [2] argue on the basis of their relativization results that traditional methods, diagonalization in particular, will not settle questions such as P = ? NP and P = ? PSPACE. They say,

> It seems unlikely that ordinary diagonalization methods are adequate for producing an example of a language in NP but not in P; such diagonalizations, we would expect, would apply equally well to the relativized classes, implying a negative answer to all relativized P = ? NP questions, a contradiction. On the other hand, we do not feel that one can give a general method for simulating nondeterministic machine by deterministic machines in polynomial time, since such a method should apply as well to relativized machines and therefore imply affirmative answers to all relativized P = ? NP questions, also a contradiction.

However, oracle computations can be so powerful that the combinatorial difficulties of resource-bounded machines are effectively overwhelmed. Therefore, one must be very careful about the conclusions drawn from relativization results. To quote Baker, Gill and Solovay again, "resolving the original question requires careful analysis of the computational power of machines." This paper is a contribution to such an analysis.

We consider refinements of PSPACE (A) and NP (A) so that with respect to these refinements positive inclusion relationships do relativize. One principal result here settles affirmatively a conjecture raised by Book in [5]. Let PQUERY (A) be the class of languages accepted relative to set A by deterministic oracle machines that operate in polynomial space and that permit in any computation at most a polynomial number of queries. We prove that P = PSPACE if and only if for every set A, P (A) = PQUERY (A). Also, a new relativization of NP is introduced here in order to obtain a positive relativization of the P = ? NP question.

Book and Wrathall [5, 7] introduced the nondeterministic analogue of PQUERY $(\cdot)$, called NPQUERY $(\cdot)$, and showed in [5] that NP = PSPACE if and only if for every set A, NP (A) = NPQUERY (A). However, the proofs in [5], [7]

* Received by the editors October 20, 1981, and in final revised form September 13, 1982. This research was supported in part by the National Science Foundation under grants MCS77-23493 and MCS80-11979. The research reported here was performed while the first author visited the Department of Computer Science, Technion, Haifa, Israel, with funds provided by the United States–Israel Educational Foundation (Fulbright Award), and while the second author visited the University of California at Santa Barbara.

† Department of Computer Science, Iowa State University, Ames, Iowa 50011. Research done at: Department of Computer Science, Technion, Haifa, Israel, supported by funds provided by the United States–Israel Educational Foundation (Fulbright Award).

‡ Harbin University of Science and Technology, Harbin, Heilongjiang, People's Republic of China. Research done at: University of California at Santa Barbara.

§ Department of Mathematics, University of California at Santa Barbara, Santa Barbara, California 93106.

involved heavy use of the "algebraic" techniques of formal language theory and applied only to classes specified by nondeterministic machines.

All the proofs in this paper will be via effective machine simulations. We depend on techniques established in [13] that relate efficient acceptors with efficient computation of functions. The techniques are described in § 2. As a consequence of our algorithmic methods, the oracle machines considered here exhibit the same computational power as do the corresponding nonrelativized devices.

A theme that has recently emerged in related work recurs here. The power of oracle Turing machines can be attributed to the number of queries that may be made, and under the restriction that the number of queries to an oracle is polynomially bounded some fundamental inclusion problems about complexity classes have equal answers for the standard case and for arbitrary relativizations. Investigations [6], [8], [18] made subsequent to [5], [7] have shown that the bound on the number of queries allowed is the factor that allows many of the diagonalization arguments used in [1]–[3], [15] to separate classes by time-bounded machines to be applied to classes specified by space-bounded machines.

Our main relativization results are found in § 3. Of course, the kind of restrictions studied here are not entirely new. A characterization of polynomial time-bounded truth-table reducibility involves placing a polynomial bound on the number of queries permitted in a computation [11]. Thus, relativizations based on truth-table reducibilities are also studied (§ 4). Generalizations and extensions to other models and other sets of bounds are summarized without proof in § 5.

Kozen [9] and Kozen and Machtey [10] also contain analyses of the Baker, Gill, and Solovay phenomena, but they lie in completely different directions.

We conclude this introduction with a few remarks concerning notation. Namely, notation is standard and consistent with the prior papers. Unless specified otherwise, all sets are languages over the finite alphabet $\Sigma = \{0, 1\}$. The length of a string x in Σ^* is denoted $|x|$.

An oracle Turing machine is a multitape Turing machine with a distinguished query tape and three distinguished states QUERY, YES, and NO. We will be interested in oracle Turing acceptors and in oracle Turing transducers. Transducers will be described in the next section. Given an oracle Turing acceptor M and an oracle set A, $L(M, A)$ will denote the set of input strings accepted by M with A as its oracle. Oracle machines may operate within time bounds or space bounds. If an oracle machine operates within a space bound S, then we require that the query tape, as well as the ordinary work tapes, be bounded in length by S.

Given a complexity class $\mathscr{C}$ and a reducibility $\leqq_r$, a set L is *$\leqq_r$-hard for $\mathscr{C}$* if for every set $A \in \mathscr{C}$, $A \leqq_r L$. If a set L is $\leqq_T^P$-hard for $\mathscr{C}$, then L is said to be *$\mathscr{C}$-hard*. (This notation is consistent with the generally accepted usage of "NP-hard" for "$\leqq_T^P$-hard for NP." Garey and Johnson [19] contains a useful discussion and terminological history.)

2. Computation of functions. The proofs of our main results employ techniques that relate efficient set acceptors with efficient computation of functions by transducers. These techniques will be described here, and they have independent interest.

A nondeterministic oracle Turing transducer is a nondeterministic oracle Turing machine with distinguished accepting states and a distinguished output tape. A transducer M computes a value y on an input string x relative to an oracle set A if there is an accepting computation of M on x with oracle A for which y is the final contents of M's output tape. In general, a nondeterministic oracle Turing machine transducer

M computes a partial multi-valued function. We will be especially interested in the case that M computes a partial single-valued function; i.e., every accepting computation of M on an input string x relative to an oracle A produces the same contents of the output tape.

Space bounds for oracle Turing machine transducers apply to all work tapes including the query tape and the output tape.

For every set A, consider the following three classes of partial single-valued functions: let PF (A) be the class of functions computed relative to A by deterministic polynomial time-bounded oracle transducers; let NPF (A) be the class of functions computed relative to A by nondeterministic polynomial time-bounded oracle transducers; and let PSPACEF (A) be the class of functions computed relative to A by deterministic polynomial space-bounded oracle transducers.

Let M be an arbitrary oracle Turing machine. Define the partial multi-valued function NEXT-CALL$_M$ on configurations of M by: J is a value of NEXT-CALL$_M$ (I) if some computation of M beginning in configuration I reaches configuration J, in configuration J M is in the QUERY state, and in that computation no configuration between I and J is in the QUERY state.

It is clear that if M is deterministic, then NEXT-CALL$_M$ is single-valued. If M is deterministic and operates in polynomial time, then NEXT-CALL$_M$ is in PF; if M is deterministic and operates in polynomial space, then NEXT-CALL$_M$ is in PSPACEF.

The functions NEXT-CALL$_M$ will be used throughout the next section as well as in the following definition.

Define an oracle machine M to be *confluent* if NEXT-CALL$_M$ is single-valued.

If M is confluent and NEXT-CALL$_M$ $(I) = J$, then there can be only two types of computation paths branching out from I—those that lead to the query configuration J and those that do not lead to queries at all. It is clear that if M is nondeterministic and confluent, and M operates in polynomial time, then NEXT-CALL$_M$ is in NPF.

One can also define the class of functions computed by nondeterministic oracle transducers that operate within polynomial space. However, using the techniques of [12], [14], it can be shown that for every set A, only functions in PSPACEF (A) are obtained.

A function $f: \Sigma^* \to \Sigma^*$ is *polynomial-bounded* if there is a polynomial p such that for all x in the domain of f, $|f(x)| \leqq p(|x|)$.

Clearly, for every set A, every function in PF (A), NPF (A), and PSPACEF (A) is polynomial-bounded.

For a partial function $f: \Sigma^* \to \Sigma^*$, let graph $(f) = \{\langle x, y\rangle | f(x) = y\}$.

The proof of the following fact is left to the reader.

LEMMA 2.1. *Let $f: \Sigma^* \to \Sigma^*$ be a function.*

(i) *If f is polynomial-bounded, then $f \in$* NPF (graph (f)).

(ii) *If f is polynomial-bounded and for some set A,* graph $(f) \in$ NP (A) (PSPACE (A)), *then f is in* NPF (A) *(resp.,* PSPACEF (A)).

(iii) *If $f \in$* PF (A) (NPF (A), PSPACEF (A)), *then* graph $(f) \in$ P (A) *(resp.,* NP (A), PSPACE (A)).

For each partial function $f: \Sigma^* \to \Sigma^*$, define code (f) to be the set of triples $\langle \sigma, x, k\rangle \in \{0, 1\} \times \Sigma^* \times \Sigma^*$, where k is the binary representation of a number $n(k)$, defined as follows:

$\langle 0, x, k\rangle \in$ code (f) if and only if $f(x)$ has an $n(k)$th bit;

$\langle 1, x, k\rangle \in$ code (f) if and only if the $n(k)$th bit of $f(x)$ is 1.

LEMMA 2.2. *Let* $f: \Sigma^* \to \Sigma^*$ *be a function.*

(i) *If* f *is polynomial-bounded, then* $f \in \mathrm{PF}(\mathrm{code}(f))$.

(ii) *If* f *is polynomial-bounded and for some set* A, $\mathrm{code}(f) \in \mathrm{P}(A)$ ($\mathrm{PSPACE}(A)$), *then* f *is in* $\mathrm{PF}(A)$ (*resp.*, $\mathrm{PSPACEF}(A)$).

(iii) *If* $f \in \mathrm{PF}(A)$ ($\mathrm{NPF}(A)$, $\mathrm{PSPACEF}(A)$), *then* $\mathrm{code}(f) \in \mathrm{P}(A)$ (*resp.*, $\mathrm{NP}(A)$, $\mathrm{PSPACE}(A)$).

This notion will be used repeatedly. Because clause (i) of Lemma 2.2 is stronger than that of Lemma 2.1, it is frequently more useful than the usual graph encoding. Selman [13] has explored in depth the notion of a function being polynomial-bounded and the usefulness of code (f) for such functions f.

Relationships between complexity classes of sets and of functions are given by the following theorem.

THEOREM 2.3.

(i) $\mathrm{P} = \mathrm{NP}$ *if and only if* $\mathrm{PF} = \mathrm{NPF}$.

(ii) $\mathrm{P} = \mathrm{PSPACE}$ *if and only if* $\mathrm{PF} = \mathrm{PSPACEF}$.

(iii) $\mathrm{NP} = \mathrm{PSPACE}$ *if and only if* $\mathrm{NPF} = \mathrm{PSPACEF}$.

Proof. In each case, the proof from right to left is immediate. To prove (i) from left to right, assume $\mathrm{P} = \mathrm{NP}$ and let $f \in \mathrm{NPF}$. By Lemma 2.2 (iii), $\mathrm{code}(f) \in \mathrm{NP}$. Therefore, $\mathrm{code}(f) \in \mathrm{P}$. Thus, Lemma 2.2 (ii) applies, and $f \in \mathrm{PF}$. Clause (ii) is proved the same way.

To prove (iii) from left to right, assume $\mathrm{NP} = \mathrm{PSPACE}$, let $f \in \mathrm{PSPACEF}$, and apply Lemma 2.1. Then, $\mathrm{graph}(f) \in \mathrm{PSPACE}$. Hence, $\mathrm{graph}(f) \in \mathrm{NP}$, from which it follows that $f \in \mathrm{NPF}$. □

The following lemma collects some trivial observations that will be needed in the next section. (Perhaps this lemma is worth a moment of reflection. Despite the intense effort these past ten years to discover NP-complete and NP-hard problems, the more challenging task is to discover some set that is not NP-hard.)

LEMMA 2.4.

(i) $\mathrm{P} = \mathrm{NP}$ *if and only if every set* A *is* NP-*hard.*

(ii) $\mathrm{P} = \mathrm{PSPACE}$ *if and only if every set* A *is* PSPACE-*hard.*

(iii) $\mathrm{NP} = \mathrm{PSPACE}$ *if and only if every set* A *is* $\leqq_{\mathrm{T}}^{\mathrm{NP}}$-*hard for* PSPACE.

3. Principal results. For every set A, let $\mathrm{PQUERY}(A)$ ($\mathrm{NPQUERY}(A)$) be the class of languages accepted relative to A by deterministic (resp., nondeterministic) oracle machines that operate in polynomial space and that make at most a polynomial number of oracle queries in any computation.

Clearly, $\mathrm{PQUERY}(A) \subseteq \mathrm{NPQUERY}(A) \subseteq \mathrm{PSPACE}(A)$, for every A. It is well known that there exist sets A and B such that $\mathrm{P}(A) = \mathrm{PSPACE}(A)$ and $\mathrm{P}(B) \neq \mathrm{PSPACE}(B)$ [1]–[3], [15]. In contrast, we will prove (see Corollary 3.2 below) that $\mathrm{P} = \mathrm{PSPACE}$ if and only if for every set A, $\mathrm{P}(A) = \mathrm{PQUERY}(A)$. In addition, our results will be organized so as to display an important role that hard sets play in this context.

Classes of the form $\mathrm{PQUERY}(A)$ and $\mathrm{NPQUERY}(A)$ were introduced by Book and Wrathall in [5], [7] where it was shown that there exist sets C and D such that $\mathrm{PQUERY}(C) \neq \mathrm{NPQUERY}(C)$ and $\mathrm{NPQUERY}(D) \neq \mathrm{PSPACE}(D)$.

For every set A, let $\mathrm{PQUERYF}(A)$ ($\mathrm{NPQUERYF}(A)$) be the class of functions computed relative to A by deterministic (resp., nondeterministic) oracle transducers that operate in polynomial space and that make at most a polynomial number of oracle queries in any computation.

Clearly, for every A, $\mathrm{PQUERYF}(A) \subseteq \mathrm{NPQUERYF}(A) \subseteq \mathrm{PSPACEF}(A)$.

THEOREM 3.1. *For every set A, the following are equivalent*:
(a) *A is* PSPACE-*hard*;
(b) $P(A) = PQUERY(A)$;
(c) $PF(A) = PQUERYF(A)$.

Compare the equivalence of (a) and (b) with the observation (essentially) made in [2] that if A is PSPACE-complete, then $P(A) = PSPACE(A)$.

Proof. It is obvious that (c) implies (b). To prove (b) implies (a), assume $P(A) = PQUERY(A)$. Note that $PSPACE \subseteq PQUERY(A) = P(A)$. Thus, A is PSPACE-hard.

It suffices to prove that (a) implies (c). Let A be a PSPACE-hard set. Let M be an oracle transducer that operates in polynomial work space and that makes at most a polynomial number of queries in any computation. Let f be the function in $PQUERYF(A)$ computed by M relative to A. We will show that $f \in PF(A)$.

Recall that NEXT-CALL$_M$ is a partial function in PSPACEF. Therefore, code (NEXT-CALL$_M$) belongs to PSPACE. Since A is PSPACE-hard, code (NEXT-CALL$_M$) is in $P(A)$ and, by Lemma 2.2, NEXT-CALL$_M$ is in $PF(A)$.

Define a partial function FINAL$_M$ on configurations of M as follows:

$$\text{FINAL}_M(I) = \begin{cases} \text{the final configuration of } M \text{ following } I, \\ \qquad\qquad \text{if NEXT-CALL}_M(I) \text{ is defined;} \\ \text{undefined} \quad \text{otherwise.} \end{cases}$$

Notice that NEXT-CALL$_M(I)$ is undefined if and only if $\langle 0, I, 1\rangle \notin$ code(NEXT-CALL$_M$). Also, since M operates in polynomial space, code (NEXT-CALL$_M$) is in PSPACE. Thus, it follows that FINAL$_M$ is in PSPACEF. Therefore, as above, FINAL$_M$ is in $PF(A)$.

The following procedure determines for any input string w, the value (if any) of $f(w)$. The basic idea is to simulate M on w by iterating instances of NEXT-CALL$_M$ until a configuration I is reached so that NEXT-CALL$_M(I)$ is undefined. Then, FINAL$_M(I)$ is computed and checked. The simulation is successful because the number of oracle calls is polynomial-bounded.

input w;
$I := I_0$; {M's initial configuration on input w}
while $\langle 0, I, 1\rangle \in$ code (NEXT-CALL$_M$)
{i.e., NEXT-CALL$_M(I)$ is defined}
  do begin
    $I :=$ NEXT-CALL$_M(I)$;
    Use I and the oracle A to determine the next configuration J;
    $I := J$
    end;
$I :=$ FINAL$_M(I)$;
if I is an accepting configuration
  then output the string on the output tape encoded in I
  else $f(w)$ is undefined.

The loop is executed at most a polynomial number of times, because M makes at most a polynomial number of queries in any computation. Both NEXT-CALL$_M$ and FINAL$_M$ are in $PF(A)$ and code (NEXT-CALL$_M$) is in $P(A)$. Hence, this procedure operates in polynomial time relative to A. Thus, $f \in PF(A)$. □

COROLLARY 3.2. *The following are equivalent*:
(a) $P = PSPACE$;
(b) $PF = PSPACEF$;
(c) *for every set* A, $P(A) = PQUERY(A)$;
(d) *for every set* A, $PF(A) = PQUERYF(A)$.

The corollary is obtained from Theorem 3.1 by means of Lemma 2.4. However, the proof of Theorem 3.1 also yields a direct proof of the corollary. Since the simulation shows $PQUERYF(A) \subseteq PF(A)$ under the hypothesis $PSPACE \subseteq P(A)$, it also shows $\forall A\ PQUERYF(A) \subseteq PF(A)$ under the more general hypothesis $PSPACE \subseteq P$.

A proof that $P \neq PSPACE$ must entail a proof that $P(A) \neq PQUERY(A)$, for some set A. By the theorem, this would be equivalent to the discovery of a set A that is not PSPACE-hard.

The classes just compared, $P(\cdot)$ and $PQUERY(\cdot)$, are both deterministic. Therefore, for oracle machines M that recognize sets in these classes, NEXT-$CALL_M$ is single-valued, and a simulation of M based on successive applications of NEXT-$CALL_M$ is already deterministic. In the proof of Theorem 3.1 the assumption $PSPACE \subseteq P(A)$ yielded NEXT-$CALL_M \in PF(A)$, and it was only necessary that M make at most a fixed polynomial number of queries in order to insure a polynomial bound on the number of applications of NEXT-$CALL_M$.

Now we will develop a positive relativization of the $P = ?\ NP$ problem, and to do so we will use the notion of confluent machines. This time we will compare a nondeterministic class with a deterministic class. The basic idea once again is to exploit single-valuedness of NEXT-$CALL_M$. If M is a nondeterministic confluent polynomial time-bounded oracle Turing machine, then a nondeterministic simulation based on iterative applications of NEXT-$CALL_M$ can be given. If $P = NP$ is assumed, then NEXT-$CALL_M$ becomes computable deterministically in polynomial time and all other instances of nondeterminism in the simulation of M can be eliminated as well.

For every set A, let $NPC(A)$ be the class of languages accepted relative to A by nondeterministic polynomial time-bounded confluent oracle machines.

There exist sets A and B such that $P(A) = NP(A)$ and $P(B) \neq NP(B)$ [2]. We will prove (Corollary 3.5) that $P = NP$ if and only if for every set A, $P(A) = NPC(A)$. This time we will see the significance of NP-hard sets. First, let us note the following fact.

THEOREM 3.3. *For every set* A, $NPC(A) \subseteq PQUERY(A)$.

Therefore, a nondeterministic confluent machine can be simulated by a deterministic machine with a polynomial limit to the number of queries. In contrast, it is shown in [5] that there exists a set A such that $NP(A) \not\subseteq PQUERY(A)$.

Let $NPCF(A)$ be the class of functions computed relative to A by nondeterministic polynomial time-bounded confluent oracle transducers.

THEOREM 3.4. *For any set* A, *the following are equivalent*:
(a) A *is* NP-*hard*;
(b) $P(A) = NPC(A)$;
(c) $PF(A) = NPCF(A)$.

Proof. It is obvious that (c) implies (b). To prove (b) implies (a), assume $P(A) = NPC(A)$. Then, $NP \subseteq NPC(A) = P(A)$; thus, A is NP-hard.

Now we show that (a) implies (c). Let A be an NP-hard set. Let f be a function in $NPCF(A)$ and let M be a confluent oracle transducer that witnesses this fact.

Using Lemma 2.2 and the hypothesis that A is NP-hard, we note that NEXT-$CALL_M$ is in $PF(A)$ and code (NEXT-$CALL_M$) is in $P(A)$.

Define a unary relation ACCEPT_M on configurations of M by $\text{ACCEPT}_M(I) =$ true if some computation of M starting in configuration I enters an accepting configuration without going through a query configuration. Clearly, $\text{ACCEPT}_M \in \text{NP}$. Hence, $\text{ACCEPT}_M \in \text{P}(A)$.

Assume (without loss of generality) that M has a unique accepting configuration except for output values, e.g., a unique accepting state and blank query tape and work tapes. Define a partial function FINAL_M on configurations of M as follows:

$$\text{FINAL}_M(I) = \begin{cases} \text{an accepting configuration of } M \text{ reachable from} \\ \text{configuration } I \text{ by means of a computation of } M \text{ that} \\ \text{starts with } I \text{ and does not pass through a query configuration} \\ \qquad\qquad \text{if such a configuration exists,} \\ \text{undefined} \quad \text{otherwise.} \end{cases}$$

Clearly, FINAL_M is in NPF and so $\text{FINAL}_M \in \text{PF}(A)$.

We give an iterative procedure for computing $f(w)$ that is similar to the one used in the proof of Theorem 3.1. The important difference is that prior to each execution of $\text{NEXT-CALL}_M(I)$, we first determine whether I leads to an accepting configuration without going through a query configuration (by an application of $\text{ACCEPT}_M(I)$).

input w;
$I := I_0$; {M's initial configuration on input w}
while $\neg$ $\text{ACCEPT}_M(I)$ and $\langle 0, I, 1\rangle \in$ code $(\text{NEXT-CALL}_M(I))$
  {i.e., $\text{NEXT-CALL}_M(I)$ is defined}
  do begin
    $I := \text{NEXT-CALL}_M(I)$;
    Use I and the oracle A to determine the next configuration J;
    $I := J$
    end;
{if $\text{NEXT-CALL}_M(I)$ is defined, then I leads to an accepting configuration without going through a query configuration}
if $\text{ACCEPT}_M(I)$
  then begin
    $I := \text{FINAL}_M(I)$;
    output the string on the output tape encoded in I
    end
  else $f(w)$ is undefined.

Since M operates in polynomial time, the loop is executed a polynomial number of times. Each step takes at most polynomial time relative to A, because ACCEPT_M and code (NEXT-CALL_M) are in $\text{P}(A)$ and both FINAL_M and also NEXT-CALL_M are in $\text{PF}(A)$. Thus, $f \in \text{PF}(A)$. □

COROLLARY 3.5. *The following are equivalent*:

(a) $\text{P} = \text{NP}$;

(b) $\text{PF} = \text{NPF}$;

(c) *for every set* A, $\text{P}(A) = \text{NPC}(A)$;

(d) *for every set* A, $\text{PF}(A) = \text{NPCF}(A)$.

Once again, the simulation given in the proof of the theorem will yield the corollary directly. But the route taken yields the observation that a set A is NP-hard if and only if $\text{P}(A) = \text{NPC}(A)$.

The proof of Theorem 3.4 can be altered so as to strengthen Theorem 3.3. Consider confluent oracle machines and define classes of the form NPQUERYC (A) in the obvious way. Since nondeterministic polynomial space is identical to deterministic polynomial space, we have the following fact.

COROLLARY 3.6. *For every set* A, PQUERY (A) = NPQUERYC (A).

In contrast, it is shown in [5] that there exists a set A such that PQUERY $(A) \neq$ NPQUERY (A). In this sense PQUERY (A) and NPQUERY (A) behave more like classes specified by time-bounded machines than like classes specified by space-bounded machines.

Confluent machines are also useful in studying classes specified by space-bounded machines.

For each space bound S and set A, let NSPACEC (S, A) be the class of languages L such that $L \in$ NSPACE (S, A) is witnessed by a nondeterministic confluent oracle machine that operates within space S.

It is easy to show that the following is true.

THEOREM 3.7. *For every space bound* S, *the following are equivalent*:

(a) DSPACE (S) = NSPACE (S);

(b) *for every set* A, DSPACE (S, A) = NSPACEC (S, A).

The proof of Theorem 3.4 also yields a positive relativization of NP = ? co-NP which is due to Timothy Long.

THEOREM 3.8. NP = co-NP *if and only if for every set* A, NPC (A) = co-NPC (A).

Proof. The proof from right to left is immediate. For the proof in the other direction, assume NP = co-NP. Let A be an arbitrary set and let $\bar{L} \in$ NPC (A) be witnessed by a confluent oracle machine M.

Next we rewrite the canonical procedure for simulating M, with a slight alteration, so that this time we have an acceptor for L.

input w;
$I := I_0$; {M's initial configuration on input w}
(1) **while** $\neg$ ACCEPT$_M$ (I) and $\langle 0, I, 1\rangle \in$ code(NEXT-CALL$_M$ (I))
  do begin
(2) $I :=$ NEXT-CALL$_M$ (I);
  Use I and the oracle A to determine the next configuration J;
  $I := J$
  end;
(3) **if** ACCEPT$_M$ (I) {i.e., $w \in \bar{L}$}
(4) **then** halt without accepting
(5) **else** accept.

Recall that NEXT-CALL$_M \in$ NPF and that code (NEXT-CALL$_M$) and ACCEPT$_M$ are in NP. Therefore, since we are assuming NP = co-NP, $\neg$code (NEXT-CALL$_M$) and $\neg$ACCEPT$_M$ are in NP. The test on line 1 is executed by simultaneous executions of NP-machines for ACCEPT$_M$ (I), $\neg$ACCEPT$_M$ (I), $\langle 0, I, 1\rangle \in$ code (NEXT-CALL$_M$ (I)), and $\langle 0, I, 1\rangle \notin$ code (NEXT-CALL$_M$ (I)). Call these machines N_1, N_2, N_3, and N_4, respectively. Line 1 becomes true if N_2 and N_3 both accept their inputs, and becomes false if either N_1 or N_4 accepts. In all other cases the procedure terminates. The test on line 3 is implemented similarly. Line 2 is implemented by a polynomial number of simultaneous calls to code (NEXT-CALL$_M$ (I)) and to $\neg$ code (NEXT-CALL$_M$ (I)). In each instance, continuation occurs only if one of the two calls accepts. Therefore, for every string w there is a computation that leads either to line 4 or to line 5. Thus, there is an accepting computation of the procedure on input w if and only if w is in L.

We conclude that the procedure describes a nondeterministic polynomial time-bounded acceptor for L relative to the oracle A; i.e., $L \in \mathrm{NPC}(A)$. Thus, $\mathrm{NPC}(A) = \text{co-NPC}(A)$ for all A. □

How does one relativize PSPACE in order to study NP = ? PSPACE? This question was considered in [5] where the equivalence of parts (a) and (c) of Corollary 3.10 are shown. The proof of Theorem 3.9 is left to the reader.

THEOREM 3.9. *For any set A, the following are equivalent*:

(a) *A is $\leqq_T^{NP}$-hard for* PSPACE;

(b) $\mathrm{NP}(A) = \mathrm{NPQUERY}(A)$;

(c) $\mathrm{NPF}(A) = \mathrm{NPQUERYF}(A)$.

COROLLARY 3.10. *The following are equivalent*:

(a) NP = PSPACE;

(b) NPF = PSPACEF;

(c) *for every set A*, $\mathrm{NP}(A) = \mathrm{NPQUERY}(A)$;

(d) *for every set A*, $\mathrm{NPF}(A) = \mathrm{NPQUERYF}(A)$.

If NP ≠ PSPACE, then it is still possible that PSPACE is equal to the union of the polynomial-time hierarchy. We can relativize this question by considering NP $(\cdot)$ and NPQUERY $(\cdot)$.

First we review some definitions.

For every set A, Let $\Sigma_1^P(A) = \mathrm{NP}(A)$, $\Sigma_{i+1}^P(A) = \cup\{\mathrm{NP}(B) | B \in \Sigma_i^P(A)\}$ for $i > 0$, and $\mathrm{NP}^*(A) = \cup_{i>0}\Sigma_i^P(A)$. For every set A, let $\Sigma_1^{PQ}(A) = \mathrm{NPQUERY}(A)$, $\Sigma_{i+1}^{PQ}(A) = \cup\{\mathrm{NPQUERY}(B) | B \in \Sigma_i^{PQ}(A)\}$ for $i > 0$, and $\mathrm{NPQUERY}^*(A) = \cup_{i>0}\Sigma_i^{PQ}(A)$.

For every set A, the structure $\Sigma_1^P(A) \subseteq \Sigma_2^P(A) \subseteq \cdots$ is the *polynomial-time hierarchy relative to A*, and the structure $\Sigma_1^{PQ}(A) \subseteq \Sigma_2^{PQ}(A) \subseteq \cdots$ is the *polynomial-query hierarchy relative to A*.

Write Σ_i^P for $\Sigma_i^P(\phi)$ and NP* for NP*(ϕ).

The polynomial-time hierarchy and its relativized versions have been well studied [1], [3], [16], [17]. The relativized versions of the polynomial-query hierarchy were introduced in [7]. The polynomial-query hierarchy collapses to PSPACE when relativized to ϕ, i.e., $\Sigma_i^{PQ}(\phi) = \mathrm{NPQUERY}^*(\phi) = \mathrm{PSPACE}$.

It is known [3] that there is a set A such that $\Sigma_1^P(A) \not\subseteq \Sigma_2^P(A) \not\subseteq \Sigma_3^P(A)$ and it is noted in [7] that the proof of this fact can be used to show that there is a set B such that $\Sigma_1^{PQ}(B) \not\subseteq \Sigma_2^{PQ}(B) \not\subseteq \Sigma_3^{PQ}(B)$.

For every set A the class of *polynomial-time hierarchy functions computed relative to A* is defined as follows: $\Sigma_1^P\mathrm{F}(A) = \mathrm{NPF}(A)$, $\Sigma_{i+1}^P\mathrm{F}(A) = \cup\{\mathrm{NPF}(B) | B \in \Sigma_i^P(A)\}$ for $i > 0$, and $\mathrm{NP}^*\mathrm{F}(A) = \cup_{i>0}\Sigma_i^P\mathrm{F}(A)$. For every set A the class of *polynomial-query functions computed relative to A* is defined as follows: $\Sigma_1^{PQ}\mathrm{F}(A) = \mathrm{NPQUERYF}(A)$, $\Sigma_{i+1}^{PQ}\mathrm{F}(A) = \cup\{\mathrm{NPQUERYF}(B) | B \in \Sigma_i^{PQ}(A)\}$ for $i > 0$, and $\mathrm{NPQUERY}^*\mathrm{F}(A) = \cup_{i>0}\Sigma_i^{PQ}\mathrm{F}(A)$.

LEMMA 3.11. NP* = PSPACE *if and only if for all sets A*, $\mathrm{PSPACE} \subseteq \mathrm{NP}^*(A)$.

THEOREM 3.12. *For any set A, the following are equivalent*:

(a) $\mathrm{PSPACE} \subseteq \mathrm{NP}^*(A)$;

(b) $\mathrm{NP}^*(A) = \mathrm{NPQUERY}^*(A)$;

(c) $\mathrm{NP}^*\mathrm{F}(A) = \mathrm{NPQUERY}^*\mathrm{F}(A)$.

Therefore, we have the following corollary.

COROLLARY 3.13. *The following are equivalent*:

(a) NP* = PSPACE;

(b) NP*F = PSPACEF;

(c) *for every set A*, $\mathrm{NP}^*(A) = \mathrm{NPQUERY}^*(A)$;

(d) *for every set A*, $\mathrm{NP}^*\mathrm{F}(A) = \mathrm{NPQUERY}^*\mathrm{F}(A)$.

The equivalence of (a) and (c) of this corollary was established in [7].

Proof of Theorem 3.12. It is clear that (c) implies (b) and that (b) implies (a). We will show that (a) implies (c).

Suppose that $\mathrm{PSPACE} \subseteq \mathrm{NP}^*(A)$ for some set A. Let M be a nondeterministic polynomial space-bounded oracle Turing machine transducer that operates in such a way that in every computation only a polynomial number of oracle calls are made. Let f be the function in $\Sigma_1^{\mathrm{PQ}}\mathrm{F}(A)$ computed by M relative to A.

Let NEXT_M be the relation on configurations such that $\mathrm{NEXT}_M(I, J)$ holds if and only if there is a computation of M beginning with configuration I that eventually reaches configuration J, configuration J is either a final (halting) configuration or is a query configuration, and during this computation no configuration I' occurring strictly between I and J is a final configuration or a query configuration. Notice that $\mathrm{graph}(\text{NEXT-CALL}_M) \subseteq \mathrm{NEXT}_M$. Since M operates in polynomial space, it is clear that NEXT_M is in PSPACE and hence is in $\mathrm{NP}^*(A)$ by hypothesis. Let $t > 0$ be the least integer such that NEXT_M is in $\Sigma_t^{\mathrm{P}}(A)$.

Determine for any input string w the value (if any) of $f(w)$ by using the following routine.

input w;
$I := I_0$; {M's initial configuration on w}
while $\exists J$ such that $\mathrm{NEXT}_M(I, J)$ holds
  do begin
    nondeterministically guess a configuration J
    if $\mathrm{NEXT}_M(I, J)$ holds
      then if the configuration J is in the QUERY state
            then begin
                 use J and the oracle for A to determine the next configuration I';
                 $I := I'$;
                 end
            else if the configuration J is an accepting configuration
                 then output the value encoded in J and halt
                 else halt
      else halt
    end.

Since M makes at most a polynomial number of oracle calls in any computation, the loop is executed at most a polynomial number of times. Thus, relative to an oracle for the relation NEXT_M, this nondeterministic routine operates in polynomial time. Since NEXT_M is in $\Sigma_t^{\mathrm{P}}(A)$, the function f computed by this routine (and by M) relative to A is in $\Sigma_{t+1}^{\mathrm{P}}\mathrm{F}(A)$.

The argument so far yields the fact that $\mathrm{PSPACE} \subseteq \mathrm{NP}^*(A)$ implies that $\Sigma_1^{\mathrm{PQ}}\mathrm{F}(A) \subseteq \mathrm{NP}^*\mathrm{F}(A)$.

Suppose for some i that $\Sigma_i^{\mathrm{PQ}}\mathrm{F}(A) \subseteq \mathrm{NP}^*\mathrm{F}(A)$. Let f be a function in $\Sigma_{i+1}^{\mathrm{PQ}}\mathrm{F}(A)$. Then there is a set $B \in \Sigma_i^{\mathrm{PQ}}(A)$ such that $f \in \Sigma_1^{\mathrm{PQ}}(B)$. By the induction hypothesis $\Sigma_i^{\mathrm{PQ}}\mathrm{F}(A) \subseteq \mathrm{NP}^*\mathrm{F}(A)$ it follows that $\Sigma_i^{\mathrm{PQ}}(A) = \mathrm{NP}^*(A)$. Thus, $B \in \mathrm{NP}^*(A)$. By the first part of the proof, $f \in \mathrm{NP}^*\mathrm{F}(B)$. It is now easy to see that $f \in \mathrm{NP}^*\mathrm{F}(A)$.

Hence, by induction, $\Sigma_i^{\mathrm{PQ}}\mathrm{F}(A)$ for every i. Thus, $\mathrm{NPQUERY}^*\mathrm{F} \subseteq \mathrm{NP}^*\mathrm{F}(A)$ as desired. □

4. Reducibilities. In this section we consider restrictions of $\mathrm{P}(\cdot)$ and $\mathrm{PQUERY}(\cdot)$. To begin, we recall the following characterization of polynomial time-bounded truth-table reducibility.

For a deterministic oracle machine M, let $Q(M, A, w)$ denote the set of all strings queried by M on input w with oracle set A. Let $Q(M, w) = \bigcup_A Q(M, A, w)$. Let $\#Q(M, w)$ be the cardinality of $Q(M, w)$.

For every set A, let $PTT(A)$ be the class of languages accepted relative to A by deterministic oracle machines M that operate in polynomial time and have the property that there is a function $f \in PF$ such that for all input strings w of M, $f(w) = z_1 \,\%\, z_2 \,\%\, \cdots \,\%\, z_t$, where $Q(M, w) = \{z_1, \cdots, z_t\}$.

From [11] we see that $PTT(A)$ is the class of sets that are truth-table reducible to A in polynomial time.

The following question arises. If $B \leqq_T^P A$ is witnessed by a deterministic oracle machine M and if there is a polynomial p such that for all input strings w of M, $\#Q(M, w) \leqq p(|w|)$, then is $B \in PTT(A)$? We are interested also in the analogous question about space-bounded oracle machines.

For every set A, let $PB(A)$ ($PBQUERY(A)$) be the class of languages accepted relative to A by deterministic oracle machines M that operate in polynomial time (resp. space) and have the property that there is a polynomial p such that for all input strings w of M, $\#Q(M, w) \leqq p(|w|)$.

LEMMA 4.1. *Let M be a deterministic oracle machine that operates in polynomial work space and has the property that there is a polynomial p such that for all input strings w of M, $\#Q(M, w) \leqq p(|w|)$. Then there is a function $f \in PSPACEF$ such that for all input strings w of M, $f(w) = z_1 \,\%\, z_2 \,\%\, \cdots \,\%\, z_t$, where $Q(M, w) = \{z_1, \cdots, z_t\}$.*

Proof. Let M satisfy the hypothesis. A deterministic machine (without an oracle) can simulate all computations of M on input w and record on some work tape the strings z in $Q(M, w)$. When all computations have been simulated, $Q(M, w)$ can be copied onto the output tape. Thus, this new machine computes a function f in PSPACEF that for every w has value $Q(M, w)$. □

Thus, if $A \in PBQUERY(B)$ via oracle machine M, then $Q(M, w)$ can be computed for each w by a function $f \in PSPACEF$. Can $Q(M, w)$ be computed in polynomial time? That is, is the function f in PF?

Define $PTTQUERY(A)$ to be the class of languages accepted relative to A by deterministic oracle machines M that operate in polynomial space and have the property that there is a function $f \in PF$ such that for all input strings w of M, $f(w) = z_1 \,\%\, z_2 \,\%\, \cdots \,\%\, z_t$, where $Q(M, w) = \{z_1, z_2, \cdots, z_t\}$.

THEOREM 4.2.

(i) *For every set A, $PTT(A) \subseteq PB(A) \subseteq P(A)$ and $PTTQUERY(A) \subseteq PBQUERY(A) \subseteq PQUERY(A)$.*

(ii) *There exists a set A such that $PTT(A) \neq P(A)$ and $PB(A) \neq P(A)$.*

(iii) *There exists a set A such that $PTTQUERY(A) \neq PQUERY(A)$ and $PBQUERY(A) \neq PQUERY(A)$.*

The proof in [11] of the existence of A such that $PTT(A) \neq P(A)$ shows that $PB(A) \neq P(A)$, and this proof can be trivially modified to yield (iii).

The proof of Theorem 3.1 can be trivially modified to show the following fact.

THEOREM 4.3. *The following are equivalent*:

(a) $P = PSPACE$;

(b) *for every set A*, $PTT(A) = PTTQUERY(A)$;

(c) *for every set A*, $PB(A) = PBQUERY(A)$;

(d) *for every set A*, $PTTQUERY(A) \subseteq P(A)$;

(e) *for every set A*, $PBQUERY(A) \subseteq P(A)$.

The questions of whether for all A, $PB(A) = PTT(A)$ and whether for all A, $PTTQUERY(A) = PBQUERY(A)$, are difficult, as shown by the next result. These are the only inclusions listed in Theorem 4.2 that are not specified as proper.

THEOREM 4.4. *If* P = PSPACE, *then for every set* A, PB(A) = PTT(A) = PTTQUERY(A) = PBQUERY(A).

Proof. Using Theorem 4.3, it is sufficient to show that P = PSPACE implies that for every set A, PBQUERY$(A) \subseteq$ PTTQUERY(A). Using the equivalent hypothesis PF = PSPACEF, this result follows from Lemma 4.1. □

For completeness' sake, we also set down the following modification of Corollary 3.10.

THEOREM 4.5. *The following are equivalent*:

(a) NP $\neq$ PSPACE;

(b) *for every set* A, PSPACE $\subseteq$ NP(A);

(c) *for every set* A, PQUERY$(A) \subseteq$ NP(A);

(d) *for every set* A, PBQUERY$(A) \subseteq$ NP(A);

(e) *for every set* A, PTTQUERY$(A) \subseteq$ NP(A).

5. Extensions. In this section we are interested in resource bounds other than just the set of polynomials and in classes specified by simultaneous bounds on time and space. Consider a complexity class that is specified by machines that operate simultaneously within time bounds from a set $\mathcal{T}$ and space bounds from a set $\mathcal{S}$. How large must the space bounds be in order to obtain all of the languages specified by machines that operate within time bounds from $\mathcal{T}$? How large must the time bounds be in order to obtain all of the languages specified by machines that operate within space bounds from $\mathcal{S}$? To study such questions the following classes are defined.

(i) For every set A, let DTISP$(\mathcal{T}, \mathcal{S}, A)$ be the class of languages accepted relative to A by deterministic oracle machines that operate within time bound T for some $T \in \mathcal{T}$ and, simultaneously, within space bound S for some $S \in \mathcal{S}$.

(ii) For every set A, let DQUSP$(\mathcal{T}, \mathcal{S}, A)$ be the class of languages accepted relative to A by deterministic oracle machines that operate within space bound S for some $S \in \mathcal{S}$ and, simultaneously, can make a number of oracle queries bounded by some $T \in \mathcal{T}$ in any computation.

(iii) For every set A, let DTIQS$(\mathcal{T}, \mathcal{S}, A)$ be the class of languages accepted relative to A by deterministic oracle machines that operate within time bound T for some $T \in \mathcal{T}$ and, simultaneously, every configuration in state QUERY has every work tape (including the query tape) bounded by some $S \in \mathcal{S}$ in any computation.

Notice that for any $\mathcal{T}$ and $\mathcal{S}$, DQUSP$(\mathcal{T}, \mathcal{S}, \phi)$ = DSPACE$(\mathcal{S})$ and DTIQS$(\mathcal{T}, \mathcal{S}, \phi)$ = DTIME$(\mathcal{T})$. Classes NTISP$(\mathcal{T}, \mathcal{S}, A)$, NQUSP$(\mathcal{T}, \mathcal{S}, A)$, and NTIQS$(\mathcal{T}, \mathcal{S}, A)$ are studied in [6] where the notions of "QUSP" and "TIQS" are introduced.

A function S is *constructible* if there is a deterministic Turing machine M such that on every input string w, M's computation on w halts having used work space $S(|w|)$ and exactly $S(|w|)$ tape squares are marked on some work tape.

A function T is a *running time* if there is a deterministic Turing machine M such that on every input string w, M's computation on w halts in exactly $T(|w|)$ steps.

If T is a running time and S is constructible, then the pair (T, S) is *compatible* if there is a deterministic machine that makes no oracle queries and that simultaneously witnesses the fact that T is a running time and S is constructible.

If a pair (T, S) is compatible, then it is the case that for some $c > 0$ and all $n \geqq 0$, $S(n) \leqq T(n) \leqq 2^{cS(n)}$. Obviously, DSPACE$(S) \subseteq$ DTISP$(2^{cS(n)}, S(n))$. Therefore, we are interested in situations where $T(n) = O(2^{cS(n)})$ or $S(n) = O(T(n))$. For example, let $T(n) = n^k$ for some interer k and let $S(n) = n$.

Let $\mathcal{F}$ be a set of functions. A function g *is* $O(\mathcal{F})$ if there is some $f \in \mathcal{F}$ such that g is $O(f)$.

For our results we require that the sets $\mathcal{T}$ and $\mathcal{S}$ of bounds satisfy the following conditions.

Condition 5.1. Let $\mathcal{T}$ be a set of running times and let $\mathcal{S}$ be a set of constructible space bounds satisfying the following conditions:

(i) for every $T \in \mathcal{T}$ and $S \in \mathcal{S}$, the pair (T, S) is compatible;

(ii) for every $T \in \mathcal{T}(S \in \mathcal{S})$ and integer $c > 0$, the function $T'(n) = cT(n)$ $(S'(n) = cS(n))$ is in $\mathcal{T}$ (resp., $\mathcal{S}$);

(iii) if $T_1, T_2 \in \mathcal{T}$, then the function $T(n) = T_1(n)T_2(n)$ is $O(\mathcal{T})$;

(iv) if $T \in \mathcal{T}$ and $S \in \mathcal{S}$, then the function $T'(n) = T(S(n))$ is $O(\mathcal{T})$;

(v) if $S_1, S_2 \in \mathcal{S}$, then the function $S(n) = S_1(S_2(n))$ is $O(\mathcal{S})$.

Using the techniques of § 3, including the introduction of classes of functions specified by these controlled relativizations, one can prove the following theorems.

THEOREM 5.2. *Let* $(\mathcal{T}, \mathcal{S})$ *be a pair satisfying Condition* 5.1. *The following are equivalent*:

(a) DTISP $(\mathcal{T}, \mathcal{S})$ = DTIME $(\mathcal{T})$;

(b) *for every set* A, DTISP $(\mathcal{T}, \mathcal{S}, A)$ = DTIQS $(\mathcal{T}, \mathcal{S}, A)$.

THEOREM 5.3. *Let* $(\mathcal{T}, \mathcal{S})$ *be a pair satisfying Condition* 5.1. *The following are equivalent*:

(a) DTISP $(\mathcal{T}, \mathcal{S})$ = DSPACE $(\mathcal{S})$;

(b) *for every set* A, DTISP $(\mathcal{T}, \mathcal{S}, A)$ = DQUSP $(\mathcal{T}, \mathcal{S}, A)$.

Let poly $= \{n^k | k > 0$ an integer$\}$ and lin $= \{kn | k > 0$ an integer$\}$. Is DTISP (poly, lin) = P? Is DTISP (poly, lin) = DSPACE (lin)? By Theorem 5.2, DTISP (poly, lin) = P if and only if for every oracle set A, every language accepted relative to A by a deterministic oracle machine that runs in polynomial time and uses only linear work space in those configurations that query the oracle is also accepted relative to A by a deterministic oracle machine that simultaneously runs in polynomial time and uses linear work space. Similarly, Theorem 5.3 tells us that DTISP (poly, lin) = DSPACE (lin) if and only if for every oracle A, every language accepted relative to A by a deterministic oracle machine that uses linear work space and can make only a polynomial number of oracle calls in any computation is also accepted relative to A by a deterministic oracle machine that simultaneously runs in polynomial time and uses linear work space.

Since it is known that P $\neq$ DSPACE (lin) [4], either DTISP (poly, lin) $\neq$ P or DTISP (poly, lin) $\neq$ DSPACE (lin). Therefore, there is a set A such that DTIQS (poly, lin, A) $\neq$ DQUSP (poly, lin, A) and for this set A, at least one of these two classes is different from DTISP (poly, lin, A).

Consider confluent oracle machines and define classes of the form NTISPC $(\mathcal{T}, \mathcal{S}, A)$ in the obvious way.

THEOREM 5.4. *Let* $(\mathcal{T}, \mathcal{S})$ *be a pair satisfying Condition* 5.1. *The following are equivalent*:

(a) DTISP $(\mathcal{T}, \mathcal{S})$ = NTISP $(\mathcal{T}, \mathcal{S})$;

(b) *for every set* A, DTISP $(\mathcal{T}, \mathcal{S}, A)$ = NTISPC $(\mathcal{T}, \mathcal{S}, A)$.

THEOREM 5.5. *Let* $\mathcal{T}$ *be a set of functions such that* $(\mathcal{T}, \mathcal{T})$ *satisfies Condition* 5.1. *The following are equivalent*:

(a) DTIME $(\mathcal{T})$ = NTIME $(\mathcal{T})$;

(b) for every set A, DTIME $(\mathcal{T}, A)$ = NTIMEC $(\mathcal{T}, A)$.

Examples of pairs $(\mathcal{T}, \mathcal{S})$ such that Theorems 5.1–5.5 hold are the following:

(a) $\mathcal{T} = \{n^k | k > 0 \text{ an integer}\}$ and
$\mathcal{S} = \{kn | k > 0 \text{ an integer}\}$;

(b) $\mathcal{T} = \{n^k | k > 0 \text{ an integer}\}$ and
$\mathcal{S} = \{n(\log n)^k | k > 0 \text{ an integer}\}$;

(c) $\mathcal{T} = \{n^{k \log n} | k > 0 \text{ an integer}\}$ and
$\mathcal{S} = \{kn | k > 0 \text{ an integer}\}$;

(d) $\mathcal{T} = \{n^{k \log n} | k > 0 \text{ an integer}\}$ and
$\mathcal{S} = \{n^k | k > 0 \text{ an integer}\}$;

(e) $\mathcal{T} = \{n^{k \log n} | k > 0 \text{ an integer}\}$ and
$\mathcal{S} = \{n(\log n)^k | k > 0 \text{ an integer}\}$;

(f) $\mathcal{T} = \{2^{(\log n)^k} | k > 0 \text{ an integer}\}$ and
$\mathcal{S} = \{kn | k > 0 \text{ an integer}\}$;

(g) $\mathcal{T} = \{2^{(\log n)^k} | k > 0 \text{ an integer}\}$ and
$\mathcal{S} = \{n^k | k > 0 \text{ an integer}\}$;

(h) $\mathcal{T} = \{2^{(\log n)^k} | k > 0 \text{ an integer}\}$ and
$\mathcal{S} = \{n(\log n)^k | k > 0 \text{ an integer}\}$.

6. Discussion. There exist sets A, B, C, and D such that $\mathrm{P}(A) = \mathrm{NP}(A)$, $\mathrm{P}(B) \neq \mathrm{NP}(B)$, $\mathrm{P}(C) = \mathrm{PSPACE}(C)$, and $\mathrm{P}(D) \neq \mathrm{PSPACE}(D)$ [1]–[3], [15]. Since these results were obtained, it has become a paradigm that important open questions about complexity classes do not relativize. Furthermore, on the basis of these results (cf. the Introduction) one argues that traditional techniques will not settle these open questions.

In this paper we developed "positive relativizations" of these questions by placing restrictions on the standard oracle machine model. Are the resulting relativizations natural? It seems that naturalness for an oracle machine model is not an intrinsic notion, but depends on the purpose for which it is designed. Consider PQUERY $(\cdot)$. Since our results are proved by general simulations, PQUERY $(\cdot)$-machines have as much computational power as do the nonrelativized PSPACE-machines. If an efficient simulation were to prove $\mathrm{P} = \mathrm{PSPACE}$ it would prove $\mathrm{P}(D) = \mathrm{PQUERY}(D)$ for every D, as well. Moreover, one approach to proving $\mathrm{P} \neq \mathrm{PSPACE}$ is to construct, by diagonalization perhaps, a set C such that $\mathrm{P}(C) \neq \mathrm{PQUERY}(C)$. We have seen that the latter task is equivalent to the construction of a set C that is not PSPACE-hard.

On the other hand, let us consider the standard oracle machine model PSPACE $(\cdot)$ together with the well-known construction of Baker, Gill, and Solovay [2] as it applies to the construction of a set C such that $\mathrm{P}(C) \neq \mathrm{PSPACE}(C)$. They apply the oracle property "$x \in L(C)$" if and only if "C contains a string of length $|x|$." A polynomial space-bounded machine can accept $L(C)$ relative to C because it can search a set of size $2^{|x|}$. Intuition says that no deterministic oracle machine can perform this search in polynomial time, and of course the proofs bear this out. It is commonly believed that $\mathrm{P} \neq \mathrm{PSPACE}$ precisely for the reason that PSPACE-machines can perform certain kinds of complete searches that polynomial time-bounded machines apparently cannot perform. Nevertheless, the "standard" relativized machine model for polynomial space is explicitly given the ability to search an oracle set of size $2^{|x|}$, and this same ability is explicitly withheld from the "standard" relativized machine model for polynomial time. When this ability is withheld from PSPACE $(\cdot)$-machines, the result is PQUERY $(\cdot)$.

For other results on PQUERY $(\cdot)$ and NPQUERY $(\cdot)$ and related classes, see [5], [6], [7], [18]. New positive relativizations of complexity classes will appear in [20]

and [21]. In [20] it is shown that P = NP if and only if for every set A, every nondeterministic polynomial time-bounded oracle Turing machine M for which $\neq Q(M, A, x)$ is bounded by a fixed polynomial has a deterministic polynomial time-bounded simulation. Notable about [21] is a positive relativization of the P = ? NP $\cap$ co-NP problem.

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, 4 (1975), pp. 431–442.

[3] T. BAKER AND A. SELMAN, *A second step towards the polynomial-time hierarchy*, Theoret. Comput. Sci., 8 (1979), pp. 177–187.

[4] R. BOOK, *On languages accepted in polynomial time*, this Journal, 1 (1972), pp. 281–287.

[5] ———, *Bounded query machines: on* NP *and* PSPACE, Theoret. Comput. Sci., 15 (1981), pp. 27–39.

[6] R. BOOK, C. WILSON, AND XU MEI-RUI, *Relativizing time, space, and time-space*, this Journal, 11 (1982), pp. 571–581.

[7] R. BOOK AND C. WRATHALL, *Bounded query machines: on* NP( ) *and* NPQUERY( ), Theoret. Comput. Sci., 15 (1981), pp. 41–50.

[8] J. DONER, *Relativized complexity classes*, submitted for publication.

[9] D. KOZEN, *Indexings of subrecursive classes*, Theoret. Comput. Sci., 11 (1980), pp. 277–301.

[10] D. KOZEN AND M. MACHTEY, *On relative diagonals*, unpublished manuscript, 1981.

[11] R. LADNER, N. LYNCH AND A. SELMAN, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–123.

[12] W. SAVITCH, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. Syst. Sci., 4 (1970), pp. 177–192.

[13] A. SELMAN, *Polynomial time enumeration reducibility*, this Journal, 7 (1978), pp. 440–457.

[14] I. SIMON, *On some subrecursive reducibilities*, Ph.D. dissertation, Stanford University, Stanford, 1977.

[15] I. SIMON AND J. GILL, *Polynomial reducibilities and upwards diagonalizations*, Proc. 9th ACM Symposium Theory of Computing, 1977, pp. 186–194.

[16] L. STOCKMEYER, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 1–22.

[17] C. WRATHALL, *Complete sets and the polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 23–33.

[18] XU MEI-RUI, J. DONER AND R. BOOK, *Refining nondeterminism in relativizations of complexity classes*, J. Assoc. Comput. Mach., to appear.

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

[20] R. BOOK, T. LONG AND A. SELMAN, *Quantitative relativizations of complexity classes*, manuscript, 1982.

[21] ———, *Qualitative relativizations of complexity classes*, manuscript, 1983.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0012 $01.25/0

TWO RESULTS ON POLYNOMIAL TIME TRUTH-TABLE REDUCTIONS TO SPARSE SETS*

ESKO UKKONEN†

Abstract. Inspired by the recent solution of the Berman–Hartmanis conjecture [SIAM J. Comp., 6 (1977), pp. 305–322] that NP cannot have a sparse complete set for many-one reductions unless P = NP, we analyze the implications of NP and PSPACE having sparse, hard or complete sets for reduction types more general than the many-one reductions. Three special cases of truth-table reductions are considered. First we show that if NP (PSPACE) has a tally $\leqq^P_{k-T}$-hard set, then P = NP (P = PSPACE). Here $\leqq^P_{k-T}$ denotes a subcase of polynomial time Turing reducibility in which, for some constant k, the reducing Turing machine is allowed to make at most k queries to the oracle. We also show that if co-NP (PSPACE) has a sparse hard set for conjunctive polynomial time reductions, then P = NP (P = PSPACE).

Key words. truth-table reduction, Turing reduction, polynomial time, hard set, sparse set

1. Introduction. This paper is one in a series initially stimulated by a conjecture of L. Berman and J. Hartmanis [1] that all sets $\leqq^P_m$-complete for NP are polynomial time isomorphic; i.e., that between any two such sets there is a polynomial time bijective reduction with polynomial time inverse. Because the conjecture implies that P $\neq$ NP, it understandably is still open.

Another implication of the conjecture is that all $\leqq^P_m$-complete sets have similar density. Since all the known complete sets have exponential density [1], Berman and Hartmanis additionally conjectured that no sparse set (a set with polynomial density) could be $\leqq^P_m$-complete for NP unless P = NP. Recently, this conjecture was proved by Mahaney [8]. However, the first significant step towards the solution was made already by P. Berman [2], who proved that if NP has an $\leqq^P_m$-complete set in 1* (a *tally* complete set), then P = NP. Fortune [3], and later Meyer and Paterson [9], improved Berman's result by showing that if NP has a co-sparse $\leqq^P_m$-complete set, then P = NP.

With the Berman–Hartmanis conjecture solved for $\leqq^P_m$-complete sets, it is natural to consider implications of NP having sparse, hard or complete sets with respect to reductions more general than $\leqq^P_m$. Particularly interesting is the polynomial time Turing reduction $\leqq^P_T$, since, as noted by Meyer (see [1]), NP has polynomial size circuits if and only if NP has a sparse $\leqq^P_T$-hard set. For Turing reductions, it is known [5] that if NP has a sparse $\leqq^P_T$-hard set (or equivalently, polynomial size circuits), then the polynomial hierarchy collapses to $\Sigma^P_2 \cap \Pi^P_2$. Mahaney [8] establishes a related result by showing that if the sparse $\leqq^P_T$-hard set for NP is actually in NP, then the polynomial hierarchy collapses to Δ^P_2. Finally, Long [7] proves a companion result that the polynomial hierarchy equals Δ^P_2 if NP has a co-sparse $\leqq^P_T$-complete set.

Thus the question of whether the existence of a sparse (or co-sparse) $\leqq^P_T$-hard set for NP implies P = NP, is still open. Instead of the $\leqq^P_T$-reducibility, which is the most general form of polynomial time reducibilities, we consider in this paper some more restricted reductions which still are properly more general than $\leqq^P_m$. First we consider polynomial time Turing reductions restricted to machines which may make at most a constant number of queries to the oracle. Such reducibility is a special case

* Received by the editors October 16, 1981, and in revised form September 10, 1982. This work was supported by the Academy of Finland and by the Finnish Cultural Foundation. Partial support was provided by the National Science Foundation under grant MCS 79-15763 at the University of California, Berkeley.

† Department of Computer Science, University of Helsinki, Tukholmankatu, SF-00250 Helsinki 25, Finland.

of polynomial time truth-table reducibility and is denoted by $\leqq^P_{k-T}$, where k is the integer constant limiting the number of queries. We show, generalizing an original result of P. Berman [2], that if NP has a $\leqq^P_{k-T}$-hard tally set (i.e., a set over a one-symbol alphabet), then P = NP. We also consider two other subcases of truth-table reductions—the polynomial time conjunctive and disjunctive reductions $\leqq^P_c$ and $\leqq^P_d$. It is shown, generalizing a result of Fortune [3], that if co-NP has an $\leqq^P_c$-hard sparse set (or, equivalently, NP has an $\leqq^P_d$-hard co-sparse set), then P = NP. These proofs are easily modified to show that if PSPACE has a tally $\leqq^P_{k-T}$-hard set or sparse $\leqq^P_c$-hard set (or co-sparse $\leqq^P_d$-hard set), then P = PSPACE.

Results closely related to ours have been recently derived by Yesha [12] (this issue, pp. 411–425).

2. Polynomial time reducibilities. We assume familiarity with classes P, NP, co-NP and PSPACE, and with the concepts of hardness and completeness for a class of languages with respect to a given form of reductions between languages [4]. Unless specified otherwise, all sets are languages over a fixed finite alphabet Σ of at least two elements. In particular, sets in 1^*, i.e., sets over a one symbol alphabet, are called *tally languages*. For $A \subset \Sigma^*$, $\bar{A}$ denotes the complement of A in Σ^*. For a string x, $|x|$ denotes the length of x.

An oracle Turing machine is a deterministic multitape Turing machine acceptor with a distinguished oracle tape and three special states Q, YES, NO. By an acceptor we mean a machine where final states are divided into accept and reject states. When the oracle machine enters state Q, the next state is YES or NO depending on whether or not the word currently on the oracle tape belongs to the oracle set. In this way, the machine with an oracle set B receives an answer to a test of the form "$x \in B$?" in one step. The set accepted by the oracle Turing machine M with B as its oracle set will be denoted by $L(M, B)$.

A set A is Turing reducible to a set B in polynomial time ($A \leqq^P_T B$), if $A = L(M, B)$ for an oracle Turing machine M that runs in polynomial time.

Set A is many-one reducible to set B in polynomial time ($A \leqq^P_m B$), if there is a function $f: \Sigma^* \to \Sigma^*$, computable in polynomial time, such that for all $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$.

The computations of an oracle Turing machine M, which operates with input x, can be described by a binary computation tree $T = T(M, x)$. The root of T is labeled x. Every leaf is labeled either "accept" or "reject". Every internal node is labeled with a query. Every right branch is labeled "yes", and every left branch is labeled "no", corresponding to whether the state following the query state is YES or NO. A path from the root to a leaf is called an *accept path* (*reject path*) if the leaf is labeled "accept" ("reject").

In what follows, we analyze three restricted forms of the Turing reductions. The first one is simply the $\leqq^P_T$-reduction, limited to oracle machines that, independently of the input and the oracle, may ask only a constant number of questions. In more detail, let $A \leqq^P_T B$ via a machine M, so that, for some fixed integer k, the height of the computation tree $T(M, x)$ is at most k for every input x. Then we write $A \leqq^P_{k-T} B$ and say that A is *k-question Turing reducible to B in polynomial time*. Thus, independently of the current oracle, any computation of M contains at most k queries.

To give the two remaining reductions, we say that a set A is *disjunctive reducible to a set B in polynomial time*, $A \leqq^P_d B$, if $A \leqq^P_T B$ via an oracle Turing machine whose accept states contain only the YES state. Furthermore, a set A is *conjunctive reducible to a set B in polynomial time*, $A \leqq^P_c B$, if $A \leqq^P_T B$ via a machine whose reject states contain only the NO state. These definitions are more convenient to work with than

the equivalent definitions in the literature (e.g. [6]). Note that if $A \leqq_d^P B$ via a machine M, then in every computation tree of M, the path leading to the leftmost leaf is the only reject path. Similarly, if $A \leqq_c^P B$ via M then the only accept path is the path leading to the rightmost leaf.

For completeness, we will recall from the literature some additional forms of reducibility between languages and compare them with those given above. First, an oracle Turing machine is called *positive*, if whenever an oracle set B is a subset of another oracle set B', then $L(M, B) \subseteq L(M, B')$. Set A is *positive reducible to a set B in polynomial time*, $A \leqq_p^P B$, if $A \leqq_T^P B$ via a positive Turing machine [11]. Clearly, both $\leqq_c^P$ and $\leqq_d^P$ are positive reductions.

Next recall that, according to the original definition [10], a set A is *truth-table reducible* to a set B ($A \leqq_{tt} B$) if there is a recursive function f that on input x computes a list of queries $q_1, \cdots, q_k$, and a boolean function α such that $x \in A$ if and only if $\alpha(C_B(q_1), \cdots, C_B(q_k)) = 1$, where C_B is the characteristic function of the set B.

The following characterization of $\leqq_{tt}$ is useful from the complexity theoretic point of view [11]. Let c be a symbol not in $\{0, 1\}$. Then $A \leqq_{tt} B$ if and only if there is an oracle Turing machine M such that $A \leqq_T B$ via M, and a recursive function $f: \{0, 1\}^* \to (c\{0, 1\}^*)^*$ such that, for each input x to M, M makes queries to B from only the list $f(x)$. If here M operates in polynomial time and f is polynomial time computable, then we say that A is *truth-table reducible to B in polynomial time* ($A \leqq_{tt}^P B$). In addition, if M is positive, then A *is positive truth-table reducible to B in polynomial time* ($A \leqq_{ptt}^P B$), and if $f: \{0, 1\}^* \to (c\{0, 1\}^*)^k$ for some fixed integer k (thus the length of $f(x)$ is always k), then A is *k-question truth-table reducible to B in polynomial time* ($A \leqq_{k-tt}^P B$).

The basic relations between these reducibilities are now outlined. Note that the reductions $\leqq_{k-T}^P$, $\leqq_c^P$ and $\leqq_d^P$ we will consider in the sequel are instances of truth-table reducibility $\leqq_{tt}^P$. From [6] we first quote:

THEOREM 1 [6].

$$A \leqq_m^P B \begin{matrix} \nearrow \\ \searrow \end{matrix} \begin{matrix} A \leqq_c^P B \\ A \leqq_d^P B \end{matrix} \begin{matrix} \searrow \\ \nearrow \end{matrix} A \leqq_{ptt}^P B \Rightarrow A \leqq_{tt}^P B.$$

We also have:

THEOREM 2. $A \leqq_m^P B \Rightarrow A \leqq_{k-tt}^P B \Rightarrow A \leqq_{k-T}^P B \Rightarrow A \leqq_{(2^k-1)-tt}^P B \Rightarrow A \leqq_{tt}^P B \Rightarrow A \leqq_T^P B$.

Proof. All the implications in the theorem are almost trivial. We show here only the second and the third one.

Let $A \leqq_{k-tt}^P B$ via a machine M. For each input the list of allowed queries is of length k. Hence M may ask the oracle only k different questions. This does not necessarily mean that the number of queries in every computation is $\leqq k$, because the same question may occur several times. However, by providing M with an extra tape for bookkeeping queries and answers, we obtain a polynomial time machine which needs to ask each different query at most once. This shows that $A \leqq_{k-T}^P B$.

If $A \leqq_{k-T}^P B$ via a machine M, then every computation tree $T(M, x)$ contains at most $2^k - 1$ queries. Let $f(x)$ denote a list of such queries. Then we can compute $f(x)$ in polynomial time $O(2^k p(|x|))$ by simulating each of the, at most, 2^k paths from the root to a leaf in $T(M, x)$. Hence $A \leqq_{(2^k-1)-tt}^P B$. □

3. *k*-question Turing reducibility and tally oracles. Now we are ready to generalize the original result of P. Berman [2] to the $\leqq_{k-T}^P$-reducibility. Technically, we follow, when appropriate, the exposition of [8]. The proof is based, besides on properties of $\leqq_{k-T}^P$, on the following *self-reducibility structure* of satisfiable boolean

formulas: the problem of deciding the satisfiability of a formula F reduces to problems of whether either of F_t and F_f is satisfiable, where $F_t(F_f)$ is the result of setting the first variable in F to true (false) and simplifying. It is important that then $|F_t| \leqq |F|$ and $|F_f| \leqq |F|$.

THEOREM 3. *If* NP *has a tally* $\leqq^P_{k-T}$*-hard set, then* P = NP.

Proof. Let SAT be the set of satisfiable boolean formulas. Since SAT is $\leqq^P_m$-complete for NP, it suffices to prove that if $\text{SAT} \leqq^P_{k-T} B$ for some $B \subseteq 1^*$, then $\text{SAT} \in \text{P}$.

Suppose that $B \subseteq 1^*$ and $\text{SAT} \leqq^P_{k-T} B$ via a machine M. Let F be a boolean formula whose satisfiability is to be decided. The self-reductions of F form a binary tree with F as the root, and with F_t and F_f, as defined above, as the left and right sons of the root, and so on. The leaves will simply be true or false. If F has m variables, then the tree will have $2^{m+1}-1$ nodes.

We perform a depth-first search on this tree and use properties of M and B to prune the search so much that the time requirement is only polynomial in $|F|$. The search will either find a satisfying assignment or determine that none exists.

At every node G encountered during the search we simulate all the computations presented by the tree $T(M, G)$. During the simulation we form a list $l = (l_1, l_2, \cdots, l_p)$ such that the list has an element l_i for each rejecting computation found in $T(M, G)$. Since $T(M, G)$ contains at most 2^k computation paths from the root to acceptance or rejection, we have $p \leqq 2^k$. Each l_i is of the form $l_i = ((q_1, a_1), (q_2, a_2), \cdots, (q_r, a_r))$. Here $q_j \in 1^*$ is the jth query in the computation represented by l_i, and $a_j = 1$ or 0 depending on whether the computation takes after the jth query the YES branch or the NO branch. Since M is a k-query machine, we have $r \leqq k$. We label node G with $l = l(G)$. All this can be accomplished in time $O(2^k p(|G|))$, where p is the polynomial bounding the time requirement of M. Since $|G| \leqq |F|$ for every formula G in the self-reducibility tree of F, we obtain $O(2^k p(|G|)) \leqq O(2^k p(|F|))$. Thus the search needs a polynomial time at each node.

For a node G, its label $l(G)$ is called a *reject label*, if we know that G is not in SAT. During the search we can infer that certain labels are reject labels as follows:

(1) l(false) is clearly a reject label.

(2) If $l(G_f)$ and $l(G_t)$ are reject labels, then $l(G)$ is also a reject label, since then G_f as well as G_t are not in SAT, which means that G cannot be in SAT.

The search is pruned by not searching below a node whose label is already known to be a reject label. Observe that if $l(G)$ is a reject label by rule (1) or (2), then all G' such that $l(G') = l(G)$ have a reject label, of course. This conclusion is correct only if we can now infer that every such G' is not in SAT. Because G is not in SAT, machine M with oracle B must follow some reject path of $T(M, G)$. Let $((q_1, a_1), (q_2, a_2), \cdots, (q_r, a_r))$ be the encoding of this path in the label $l(G)$. Thus each $q_i \in B$ if and only if $a_i = 1$. Since $l(G) = l(G')$, a reject path with the same encoding occurs in $T(M, G')$. With oracle B, machine M must follow this path and will reject G'. Thus G' is not in SAT.

The search stops when either a leaf with formula "true" is found or when $l(F)$ is found to be a reject label. In the former case the path from the root to the "true" leaf indicates a satisfying assignment. In the latter case F cannot be satisfiable.

To complete the proof, we must show that the outlined searching algorithm runs in polynomial time. The following lemma establishes this. □

LEMMA 4. *Let F be a formula with m variables, and let p be a polynomial bound of the running time of M. Then the algorithm above visits at most* $m + m \times (2(p(|F|)+1))^{k \cdot 2^k}$ *interior nodes of the self-reducibility tree for F and therefore runs in polynomial time.*

Proof. If G and G' are two unsatisfiable formulas with the same label (i.e., $l(G) = l(G')$) occurring in the interior of the pruned search tree, then they must be on the same branch from the root. Otherwise, one of the formulas, say G, would be searched first, and its label $l(G)$ would be determined to be a reject label. But then the depth-first search would not go below G', contradicting the assumption that G' is not a leaf.

Thus the number of distinct paths from the root to unsatisfiable interior nodes is bounded by the number of distinct reject labels. This number surely is at most equal to the number of all possible labels. This in turn equals $(2(p(|F|)+1))^{k \cdot 2^k}$, because each label $l(G)$ is of the form $(l_1, \cdots, l_p)$, where $p \leqq 2^k$, and each l_i is of the form $((q_1, a_1), (q_2, a_2), \cdots, (q_r, a_r))$, where $r \leqq k$, and each $q_j \in 1^*$ is of length at most $p(|G|) \leqq p(|F|)$, and each a_j equals 0 or 1. Since the tree has height m, there are at most $m \times (2(p(|F|)+1))^{k \cdot 2^k}$ interior nodes with reject labels. A satisfying assignment visits at most another m nodes. □

The method presented in the proof of Theorem 3 above for deciding the satisfiability of a boolean formula can easily be adopted for deciding in polynomial time the validity of a closed quantified boolean formula; cf. [3]. Since the set of valid such formulas is $\leqq^P_m$-complete for PSPACE, we obtain:

THEOREM 5. *If* PSPACE *has a tally* $\leqq^P_{k-T}$*-hard set, then* P = PSPACE.

Proof. We sketch the changes needed in the proof of Theorem 3. Let QFB denote the set of closed valid quantified boolean formulas. Suppose that QFB $\leqq^P_{k-T} B$ via a machine M for some $B \subseteq 1^*$. We show that then QFB is in P.

Let $F = Q_1 x_1 Q_2 x_2 \cdots Q_v x_v H(x_1, \cdots, x_v)$, where each Q_i is $\forall$ or $\exists$, be a closed quantified boolean formula whose validity is to be decided. Denote by F_t and F_f the formulas obtained by setting x_1 in F to true and false and simplifying. The problem of deciding the validity of F reduces to the problem of whether F_t and F_f are valid when Q_1 is $\forall$, and to the problem of whether F_t or F_f is valid when Q_1 is $\exists$. Since $|F_t| \leqq |F|$ and $|F_f| \leqq |F|$, we again have a self-reducibility structure.

The validity of F can be decided in polynomial time during a depth-first search over the tree describing the self-reducibility structure of F. At every node G encountered during the search, we simulate all the computations presented by $T(M, G)$ and form two lists, l and l'. List l is as in the proof of Theorem 3, and list $l' = (l'_1, l'_2, \cdots, l'_{p'})$ is as list l, but l' contains an element l'_i for each accepting computation found in $T(M, G)$. We label node G with two labels, $l = l(G)$ and $l' = l'(G)$.

Label $l(G)$ is the *proper label* of G if we know that G is in $\overline{\text{QFB}}$, and $l'(G)$ is the proper label of G if we know that G is in QFB. Thus at most one of $l(G)$ and $l'(G)$ may be proper. During the search we can infer the proper label for certain nodes as follows:

(1) l'(true) and l(false) are clearly proper labels.

(2) If $l'(G_f)$ and $l'(G_t)$ are proper labels, then $l'(G)$ is the proper label of G (independently of the leading quantifier of G); if $l'(G_f)$ or $l'(G_t)$ is a proper label and the leading quantifier of G is $\exists$, then $l'(G)$ is the proper label of G; if $l(G_f)$ or $l(G_t)$ is a proper label and the leading quantifier of G is $\forall$, then $l(G)$ is the proper label of G.

(3) If $l(G')$ equals the proper label of another node G, then $l(G')$ is the proper label of G', and if $l'(G')$ equals the proper label of G, then $l'(G')$ is the proper label of G'.

Correctness of rules (1) and (2) should be obvious. As for rule (3), suppose that the proper label L for G has been chosen correctly. This means that the path in $T(M, G)$, which M follows with oracle B and input G, must be in L. If now $l(G') = L$ for some node G' whose proper label is still unknown, a path with the same encoding

must occur among the reject paths of $T(M, G')$. With oracle B, machine M must follow this path and will reject G'. Hence $l(G') = L$ is the correct proper label for G'. The case where $l'(G') = L$, and therefore an accept path is followed in $T(M, G')$, is similar. It is also easy to show by contradiction that rule (3) cannot give two proper labels for G'.

The search is pruned by not searching below a node whose proper label is already known. The search stops when the proper label for the root F is found. Formula F is in QFB if and only if $l'(F)$ is proper.

To estimate the time requirement of the search, let F be a formula with m variables, and let p be a polynomial bounding the running time of M. Then the search visits at most $m \times (2(p(|F|)+1))^{k \cdot 2^k}$ interior nodes of the self-reducibility tree for F and hence takes a polynomial time in $|F|$. That is, if G and G' are formulas with the same proper label occurring in the interior of the pruned search tree, then, exactly as in the proof of Lemma 4, they must be on the same path from the root. The number of different proper labels is again $\leqq (2(p(|F|)+1))^{k \cdot 2^k}$. Since the tree is of height $\leqq m$, the upper bound follows. □

Moreover, the generalization of Fortune's [3] results given by Meyer and Paterson [9] also applies to our proof of Theorem 3. Thus every language which has a self-reducibility property in the precise sense defined in [9], and which $\leqq^P_{k\text{-}T}$-reduces to a tally language, can be recognized in polynomial time.

4. Conjunctive reducibility and sparse oracles. In conjunctive and disjunctive reductions, a Turing machine uses its oracle in a very limited way. Therefore for these reductions, we may obtain a result similar to Theorem 3 without a restriction to k-question machines. Also, the restriction to tally oracles can be relaxed. It suffices to assume that the oracle is *sparse*, that is, there is a polynomial q such that the number of elements in the oracle of length at most n is at most $q(n)$.

THEOREM 6. *If* co-NP *has a sparse* $\leqq^P_c$*-hard set* (*or equivalently*, NP *has a co-sparse* $\leqq^P_d$*-hard set*), *then* P = NP.

Proof. We first note that because $A \leqq^P_c B$ if and only if $\bar{A} \leqq^P_d \bar{B}$, the two alternative premises of the theorem really are equivalent.

The set of nontautological boolean formulas, $\overline{\text{SAT}}$, is $\leqq^P_m$-complete for co-NP. Hence it suffices to prove that if $\overline{\text{SAT}} \leqq^P_c B$, where $B \subset \Sigma^*$ is sparse, then $\overline{\text{SAT}} \in \text{P}$. Then also $\text{SAT} \in \text{P}$, which means P = NP, as required.

Let $\overline{\text{SAT}} \leqq^P_c B$ via a Turing machine M whose only reject state is the NO state, and whose running time is bounded by a polynomial p. Let F be a boolean formula whose satisfiability is to be decided. We again perform a similar depth-first search on the tree of self-reductions of F as in the proof of Theorem 3. The labeling function l must be modified as follows: The label of a node corresponding to a formula G, $l(G)$, equals the list of all queries occurring in the tree $T(M, G)$. The list is easy to form because it is the sequence of queries on the (rightmost) path on which each query to the oracle is answered "yes". So $l(G)$ is of the form $l(G) = (q_1, \cdots, q_r)$, where each $q_i \in \Sigma^{p(|F|)}$ and $r \leqq p(|G|) \leqq p(|F|)$. Hence, $l(G)$ can be computed in time $O(p(|F|))$.

As in the proof of Theorem 3, a label $l(G)$ is called a reject label if we know that $G \leqq \overline{\text{SAT}}$. (Actually, machine M will accept such a formula G, but we prefer to stick to the old terminology.) Now we have three possibilities (1)–(3) below to infer that certain labels are reject labels ((1) and (2) are as in the proof of Theorem 3):

(1) l(false) is a reject label.

(2) If $l(G_f)$ and $l(G_t)$ are reject labels, then $l(G)$ is also a reject label.

(3) If each element q_i in the label $l(G) = (q_1, \cdots, q_r)$ occurs in some reject label, then $l(G)$ is also a reject label.

If $l(G')$ is a reject label, then M accepts G', which is possible only if M obtains the answer "yes" to every query it makes for input G'. So every element of $l(G')$ must be in the oracle set B. This also implies that the new rule (3) is correct. Suppose, namely, that $l(G)$ is to be a reject label by rule (3), i.e. that each element of $l(G)$ occurs in some other reject label found so far. Then we know (formally, by the induction hypothesis) that every element of $l(G)$ must be in B. This implies that M must accept G, and hence $G \in \overline{\text{SAT}}$.

Now the search procedure can be completed as in the proof of Theorem 3: The search is pruned by not searching below a node whose label is already known to be a reject label. In this way, either a satisfying assignment is found, or $l(F)$ is found to be a reject label, in which case $F \in \overline{\text{SAT}}$.

In the following lemma we show that the running time of the pruned search is polynomially bounded. □

LEMMA 7. *Let F be a formula with m variables. Let p be a polynomial bounding the running time of M, and q a polynomial bounding the density of the sparse oracle set B. Then the algorithm above visits at most $m + m \times q(p(|F|))$ interior nodes of the self-reducibility tree for F and therefore runs in polynomial time.*

Proof. We will again show that the number of distinct paths from the root of the pruned search tree to an interior node is polynomially bounded. Because now the number of different labels of nodes does not necessarily have a polynomial bound, the proof of Lemma 5 must be modified.

Let t be a path in the pruned tree from the root to a leaf corresponding to an unsatisfiable formula. Suppose moreover that the interior nodes of t are not properly contained in the interior nodes of any other path. Let G be the last interior node of t. Then all sons of G must be leaves.

Consider the moment when the search reaches a node G. Denote by A the set of those strings in Σ^* which occur in some label known to be a reject label. Since G is an interior node, at least one element q of $l(G)$ must be outside A. Otherwise we could infer by rule (3) that $l(G)$ is a reject label, and the depth-first search would not go below G, contradicting the assumption that G is not a leaf.

Hence the search goes below G. When returning back to G, $l(G)$ is determined to be a reject label. The current set A therefore will also contain q. So each path t must increase the size of A by at least one. On the other hand, A must always be a subset of the sparse oracle B. The length of each element of A is at most $p(|F|)$. There are at most $q(p(|F|))$ such elements in B. Thus $q(p(|F|))$ is an upper bound for the number of distinct paths t. Then, since the tree has height m, there can be at most $m \times q(p(|F|))$ interior nodes with reject labels. A satisfying assignment again visits at most another m nodes. □

As for Theorem 3, we also have in this case the following related result:

THEOREM 8. *If* PSPACE *has a sparse $\leqq_c^P$-hard set (or equivalently, a co-sparse $\leqq_d^P$-hard set), then* P = PSPACE.

Proof. The two alternative premisses are equivalent, because—as already mentioned—$A \leqq_c^P B$ if and only if $\bar{A} \leqq_d^P \bar{B}$, and because PSPACE is closed for complementation.

Let $B \subset \Sigma^*$ be a sparse $\leqq_c^P$-hard set for PSPACE. Then $\overline{\text{QFB}} \leqq_c^P B$ via a machine M, and $\text{QFB} \leqq_c^P B$ via a machine M', since both QFB and $\overline{\text{QFB}}$ are in PSPACE. Let F be the quantified boolean formula whose validity is to be decided. As in the proof of Theorem 5, we perform a depth-first search on the tree of the self-reductions

of F. At every node G encountered during the search, we form two lists, $l(G)$ and $l'(G)$. List $l(G)$ equals the list of all queries occurring in the tree $T(M, G)$, and list $l'(G)$ equals the list of all queries occurring in the tree $T(M', G)$. Both lists can be formed as described in the proof of Theorem 6.

The definition of the proper label for G is as in the proof of Theorem 5. Also rules (1) and (2) for inferring the proper label remain unchanged. Rule (3) should be replaced by the following:

(3′) If each element q_i in $l(G) = (q_1, \cdots, q_r)$ occurs in some proper label, then $l(G)$ is the proper label of G; if each element q_i in $l'(G)$ occurs in some proper label, then $l'(G)$ is the proper label of G.

Rule (3′) is correct, since all elements in proper labels must be members of B.

The search over the tree of self-reductions of F proceeds and stops as described in the proof of Theorem 5. The time requirement is polynomial in $|F|$, since the search visits at most $m \times q(p(|F|))$ interior nodes of the tree, where m is the number of variables in F, p is a polynomial bounding the running time of M and M', and q is a polynomial bounding the density of B. The proof is similar to the proof of Lemma 7, if "reject label" is replaced by "proper label." □

Moreover, the generalization in [9] applies to the proof of Theorem 6. Thus every language which has the self-reducibility property of [9], and the complement of which $\leqq_c^P$-reduces to a sparse language, can be recognized in polynomial time.

REFERENCES

[1] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of* NP *and other complete sets*, this Journal, 6 (1977), pp. 305–322.

[2] P. BERMAN, *Relationship between density and deterministic complexity of* NP*-complete languages*, Automata, Languages and Programming (Fifth Int. Coll., Udine, Italy, July 1978), Lecture Notes in Computer Science 62, Springer-Verlag, Berlin-Heidelberg-New York, 1978, pp. 63–71.

[3] S. FORTUNE, *A note on sparse complete sets*, this Journal, 8 (1979), pp. 431–433.

[4] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.

[5] R. M. KARP AND R. J. LIPTON, *Some connections between non-uniform and uniform complexity classes*, Proc. 12th ACM Symposium on Theory of Computing, 1980, pp. 302–309.

[6] R. E. LADNER, N. A. LYNCH AND A. L. SELMAN, *A comparison of polynomial time reducibilities*, Theoret. Comput. Sci., 1 (1975), pp. 103–123.

[7] T. J. LONG, *A note on co-sparse polynomial time Turing complete sets for* NP, Manuscript, Dept. Computer and Information Science, Ohio State University, Columbus, 1980.

[8] S. R. MAHANEY, *Sparse complete sets for* NP*: Solution of a conjecture of Berman and Hartmanis*, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science, 1980, pp. 54–60.

[9] A. R. MEYER AND M. S. PATERSON, *With what frequency are apparently intractable problems difficult*? MIT Technical Report MIT/LCS/TM-126, Massachusetts Institute of Technology, Cambridge, MA, 1979.

[10] E. L. POST, *Recursively enumerable sets of integers and their decision problems*, Bull. Amer. Math. Soc., 50 (1944), pp. 284–316.

[11] A. L. SELMAN, *Analogues of semirecursive sets and effective reducibilities to the study of* NP *complexity*, Manuscript, Computer Science Dept., Iowa State University, Ames, 1981.

[12] Y. YESHA, *On certain polynomial-time truth-table reducibilities of complete sets to sparse sets*, this Journal, this issue, pp. 411–425.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0013 $01.25/0

A SHORTEST-PATH ALGORITHM WITH EXPECTED TIME $O(n^2 \log n \log^* n)$

PETER A. BLONIARZ*†

Abstract. We present an algorithm which determines the shortest distances between all pairs of points in a nonnegatively weighted directed graph in average time $O(n^2 \log n \log^* n)$. This algorithm, which uses a search strategy similar to Dantzig [Management Sci., 2 (1960), pp. 187–190; *Linear Programming and Extensions*, Princeton Univ. Press, 1963] and Spira [SIAM J. Comput., 2 (1973), pp. 28–32] executes in the stated time over quite general classes of probability distributions on such graphs.

Key words. graph, algorithm, shortest path, computational complexity, expected time, average time

1. Introduction. We consider the problem of finding the shortest distance between all pairs of nodes in a nonnegatively weighted directed graph. For the worst-case measure of complexity, the fastest known algorithm for solving this problem is due to Fredman [10]. His algorithm has a running time[1] of $O(n^3 (\log \log n)^{1/3}/(\log n)^{1/3})$, a slight improvement over the $O(n^3)$ algorithms of Dijkstra [6] and Floyd [8].

In this paper, we examine the expected time to solve this problem, where we assume the existence of a probability distribution on the set of nonnegatively weighted directed graphs, and evaluate the average running time under this distribution.

Spira [15] first posed this problem under the assumption that the edge weights in the graph were independent identically distributed random variables from an arbitrary distribution. He proposed an algorithm which has an average running time[2] of $O(n^2 \log^2 n)$. The main result of this paper is an algorithm for this problem, which for a class of probability distributions which includes that of Spira, has expected running time $O(n^2 \log n \log^* n)$.

The strategy of this algorithm and those of Spira [15], Dijkstra [6] and Dantzig [5] is basically the same. A single source node is specified, and the algorithm computes the shortest distance from the source node to all other nodes by searching out from the source node along paths of increasing length. The improvement in average running time of this and Spira's algorithms over the basic paradigm of Dantzig results from not unnecessarily considering long paths in the graph. Our improvement over Spira's algorithm is to avoid considering short paths which lead to nodes to which we already know the distance.

---

* Received by the editors September 10, 1980, and in revised form August 5, 1982. This research was supported by the National Science Foundation under grant MCS78-04346. A preliminary version of this material appeared in the Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April, 1980, pp. 378–384.

† Department of Computer Science, State University of New York at Albany, Albany, New York 12222.

[1] We use the following notation in this paper. All logarithms are to the base e, and

$$\log^* n = \min \{i \in \mathbb{N} | \overbrace{\log \log \cdots \log}^{i \text{ times}} n \leqq 1\}.$$

For any finite set S, $|S|$ is the number of elements in S, and for any real number r, $\lceil r \rceil$ is the least integer greater than or equal to r.

[2] Spira's analysis of his algorithm neglected the possibility of equal-length paths and edges, which may lead to slow implementation in certain instances [3]. A corrected version of his algorithm, together with a proof of the stated average time for the class of probability distributions described in this paper, is contained in [2].

Using this general strategy, the new algorithm obtains the shortest distance from a single source node to all other nodes in expected time $O(n \log n \log^* n)$. However, the algorithm requires that the graph be presented as lists of edges from each node in sorted order, and the conversion to this representation, if necessary, requires $\theta(n^2 \log n)$ time on the average to sort. Since Dijkstra's algorithm computes the shortest distance from a single source node to all others in worst-case time $O(n^2)$, our algorithm is practical only if one desires the shortest distance between all pairs of nodes in the graph.[3]

The class of probability distributions for which the results hold is quite general. Informally, we require that the cost of an edge in the graph be independent of the node to which it points (though it may depend on the node from which it points). The formal definition is contained in § 4.

Empirical studies of this algorithm are reported in § 6. In [4], it was shown that, for randomly chosen graphs, Yen's implementation of Dijkstra's algorithm [16] is superior to Spira's algorithm for graphs of fewer than 100 nodes after which Spira's is superior. Our studies indicate that the average running time of this algorithm is superior to that of Yen for graphs with 25 nodes or more, and that Spira's algorithm need not be as inefficient as was previously reported.

Algorithms which solve related graph connectivity problems with small expected time may be found in [3], [12], [13].

2. Graph definitions. We assume that the reader is familiar with standard graph terminology as contained in [1]. In particular, a *graph* will be a labelled directed graph with nonnegative edge weights; that is, a triple $G = (V, E, c)$, where $V = \{1, 2, \cdots, n\}$ is a set of *nodes*, $E \subset V \times V$ is a set of *edges*, and $c: E \to \{r \in \mathbb{R} | r \geqq 0\} \cup \{\infty\}$ is a function giving the *cost* or *weight* of each edge in the graph. For simplicity, we assume that $E = V \times V$, and represent the absence of an edge by an edge with cost ∞.

A graph may be represented in several standard ways. One is to present the cost function c as the $n \times n$ *adjacency matrix* of edge costs. Another is the *adjacency list* representation, in which for each node $i \in V$ we have a list L_i of the edges emanating from node i. For this algorithm, we will use the *sorted adjacency list* representation in which each adjacency list L_i is ordered by increasing cost. Although many structures for storing lists could be used, we assume in our description that the sorted adjacency lists are stored as a pair of $n \times n$ matrices TAIL and COST, where

TAIL (i, k) is the node index of the kth most expensive edge emanating from node i,

and

COST (i, k) is the cost of this edge, namely $c(i, \text{TAIL } (i, k))$.

A graph presented as either an adjacency matrix or (unsorted) adjacency list may be converted into a sorted adjacency list representation in time $O(n^2 \log n)$ using a sorting algorithm such as heapsort [1].

The possibility of equal-weight edges on an edge list is a complicating factor. For definiteness, we assume that the endpoints of equal-weight edges on an edge list are in random order. This restriction is actually unnecessary, as we will discuss in § 5.

A path in a graph from node i to node j is a finite sequence of edges (v_0, v_1), (v_1, v_2), (v_2, v_3), $\cdots$, (v_{k-1}, v_k) such that $v_0 = i$ and $v_k = j$. The cost of a path is the sum of the costs of the edges which comprise it; in particular, in any graph there is

[3] The initial sort required in this algorithm may be eliminated to solve the single-source shortest path problem in $O(n^2)$ average time, but $\theta(n^2 \log n)$ worst-case time.

a path of no edges from any node to itself of cost 0. We define the *minimum cost* (or *shortest-path*) matrix MIN of a graph G by

$$\text{MIN}(i, j) = \text{minimum}\{\text{cost}(P) \mid P \text{ is a path from } i \text{ to } j \text{ in } G\}.$$

Since all edge costs are nonnegative, all entries in MIN are well-defined.

The problem of constructing the shortest-path matrix MIN for a graph is the *all-pairs shortest-path problem.* The *single-source shortest-path problem* is that of computing one row of this matrix, namely computing the shortest distance from a single *source* node to all other nodes in the graph.

3. The algorithm. To solve the single-source shortest-path problem, the algorithms of Dantzig [5], Spira [15] and this paper all follow the same general strategy. In these algorithms, successively longer paths from the source are investigated until the shortest paths to all nodes are discovered. We begin by describing Dantzig's algorithm.

Let G be a graph and SOURCE be a node in G. To find the shortest distances from SOURCE, the algorithm maintains a set NEAR of modes which satisfies two properties: If $J \in$ NEAR, then the distance MIN (SOURCE, J) has been determined by the algorithm. Furthermore, the nodes in NEAR are those closest to SOURCE; i.e. for all $J \in$ NEAR and $K \notin$ NEAR, MIN (SOURCE, J) $\leqq$ MIN (SOURCE, K). An element of NEAR will be termed a NEAR node and all others will be called FAR nodes.

Dantzig's algorithm initializes NEAR to {SOURCE} and iteratively adds nodes to NEAR until all nodes are included. To do this, it considers certain paths leading to FAR nodes. For each NEAR node J, the algorithm maintains a path defined as follows:

END (J) is the FAR node closest to J by a single edge; i.e., END (J) $\notin$ NEAR and $c(J, \text{END}(J)) = \text{minimum}\{c(J, K) \mid K \notin \text{NEAR}\}$.

PATH (J) is the path from SOURCE to END (J) consisting of a minimal-length path from SOURCE to J followed by the edge from J to END (J).

COSTPATH (J) is the cost of PATH (J), i.e.,

$$\text{MIN}(\text{SOURCE}, J) + c(J, \text{END}(J)).$$

The algorithm is based on the following observation:

LEMMA 1 [5]. *If J_0 is a* NEAR *node such that*

$$\text{COSTPATH}(J_0) = \text{minimum}\{\text{COSTPATH}(J) \mid J \in \text{NEAR}\},$$

then:

a) PATH (J_0) *is a path of minimum cost from* SOURCE *to* END (J_0); i.e.,

$$\text{MIN}(\text{SOURCE}, \text{END}(J_0)) = \text{COSTPATH}(J_0).$$

b) MIN (SOURCE, END (J_0)) $\leqq$ MIN (SOURCE, K) for all $K \notin$ NEAR.

Thus, if node J_0 satisfies the hypotheses of Lemma 1, then the node END (J_0) may be added to NEAR while preserving the properties we require of NEAR. Dantzig's algorithm repeatedly uses Lemma 1 to add nodes to NEAR until all nodes are included, at which point the algorithm has determined the minimum distance from SOURCE to all other nodes.

In implementing Dantzig's algorithm, one must maintain the condition that, for all NEAR nodes J, END (J) be a FAR node. When a node END (J_0) is added to NEAR, all NEAR nodes J whose PATH terminates at END (J_0) must have their END and COSTPATH values changed since END (J) is now a NEAR node. This

requires finding the next-cheapest edge from J to a FAR node, and may be performed efficiently if the graph is represented by a sorted adjacency list. If a pointer POINTER (J) to the edge from J to END (J) is maintained for each edge list, then the least-cost edge from J to a FAR node is obtained by advancing POINTER (J) along the Jth row of TAIL until an edge to a FAR node is encountered.

A PASCAL-like encoding of Dantzig's algorithm is contained in Fig. 1. UPDATE maintains the correctness of the END and COSTPATH vectors.

```
DANTZIG (SOURCE):
  NEAR ← {SOURCE};
  For all J ∈ V do POINTER (J) ← 1;
  MIN (SOURCE, SOURCE) ← 0;
  UPDATE (SOURCE);
LOOP: While NEAR ≠ V do begin
        J ← MINPATH;
        E ← END (J);
        NEAR ← NEAR ∪ {E};
        MIN (SOURCE, E) ← COSTPATH (J);
        UPDATE (E);
        For all K such that END (K) = E do UPDATE (K) end;
  End DANTZIG.

MINPATH: Returns node J0 such that COSTPATH (J0) = minimum {COSTPATH (J)|J ∈ NEAR}.

UPDATE (K):
  If NEAR ≠ V then begin
        While TAIL (K, POINTER (K)) ∈ NEAR do
              POINTER (K) ← POINTER (K) + 1;
        END (K) ← TAIL (K, POINTER (K));
        COSTPATH (K) ← MIN (SOURCE, K) + COST (K, POINTER (K)) end;
  End UPDATE.
```

FIG. 1. *Dantzig's algorithm.*

The dominant factor in the running time of Dantzig's algorithm is the cost of searching adjacency lists in UPDATE. In the worst case, all NEAR nodes may need to be UPDATED in each iteration of LOOP resulting in a $\theta(n^2)$ running time. The average running time has the same asymptotic behavior.

Spira improved the time on the average by reducing this searching of adjacency lists. This was accomplished by relaxing the requirement that each path PATH (J) terminate in a FAR node. In this algorithm, NEAR, PATH and COSTPATH are defined as in Dantzig's algorithm, but END satisfies the following weakened constraint:

END (J) is a (NEAR or FAR) node such that

$$c(J, \text{END}(J)) \leqq \text{minimum}\{c(J, K) | K \notin \text{NEAR}\}.$$

Spira's algorithm is based on the following analogue to Lemma 1.

LEMMA 2 [15]. *Suppose* J_0 *is a* NEAR *node such that*

$$\text{COSTPATH}(J_0) = \text{minimum}\{\text{COSTPATH}(J) | J \in \text{NEAR}\}.$$

If END (J_0) $\notin$ NEAR *then*:

a) PATH (J_0) *is a path of minimum cost from* SOURCE *to* END (J_0); *i.e.*,

$$\text{MIN}(\text{SOURCE}, \text{END}(J_0)) = \text{COSTPATH}(J_0).$$

b) MIN (SOURCE, END (J_0)) $\leqq$ MIN (SOURCE, K) *for all* $K \notin$ NEAR.

Spira's algorithm uses Lemma 2 to expand NEAR by repeatedly selecting the NEAR node J_0 with minimum COSTPATH and, if END $(J_0) \notin$ NEAR, adding END (J_0) to NEAR. The weakened assumption about the END values has two implications. Upon expansion of NEAR, it is no longer necessary to update all END values so that all PATHs terminate at FAR nodes. However, application of Lemma 2 will not add a new node to NEAR in the event that END (J_0) is already a member of NEAR.

To avoid the latter problem, when the NEAR node J_0 with minimum COSTPATH is selected, its PATH is updated. To avoid searching J_0's edgelist, POINTER (J_0) is merely advanced by one to the next-most-costly edge from node J_0, and END (J_0) and COSTPATH (J_0) updated to refer to this edge. When a node E is newly added to NEAR, its POINTER is merely set to the first edge on its edge list. A straightforward argument demonstrates that, ultimately, repeated applications of Lemma 2 and this UPDATE rule will enlarge NEAR, and the correctness of the algorithm follows. A detailed encoding of the algorithm is in Fig. 2.

```
SPIRA (SOURCE):
  NEAR ← {SOURCE};
  For all J ∈ V do POINTER (J) ← 0;
  MIN (SOURCE, SOURCE) ← 0;
  UPDATE (SOURCE):
LOOP: While NEAR ≠ V do begin
        J ← MINPATH;
        E ← END (J);
        If E ∉ NEAR do begin
          NEAR ← NEAR ∪ {E};
          MIN (SOURCE, E) ← COSTPATH (J);
          UPDATE (E) end;
        UPDATE (J) end;
  End SPIRA.

MINPATH: Returns node J0 such that COSTPATH (J0) = minimum {COSTPATH (J)|J ∈ NEAR}.

UPDATE (K):
  if NEAR ≠ V then begin
        POINTER (K) ← POINTER (K) + 1;
        END (K) ← TAIL (K, POINTER (K));
        COSTPATH (K) ← MIN (SOURCE, K) + COST (K, POINTER (K)) end;
  End UPDATE.
```

FIG. 2. *Spira's algorithm.*

In the worst case, Spira's algorithm may execute LOOP $\theta(n^2)$ times, but its average case performance is better than Dantzig's algorithm. Under the assumptions on the probability distribution on graphs cited in § 4, Spira's algorithm executes LOOP $O(n \log n)$ times on the average [2], [15]. If the nodes in NEAR are maintained in a heap [1] by their COSTPATH values, then each iteration of the loop takes $O(\log n)$ time, yielding a total average time of $O(n \log^2 n)$.

The algorithm presented in this paper is a synthesis of Dantzig's and Spira's algorithms. We preserve Spira's idea of allowing PATHs to terminate at NEAR nodes (and hence avoid searching all edge lists for FAR nodes). Spira's limit on the search of edgelists in UPDATE to the next edge on the list (which may be in NEAR), is overly conservative, however. This results in a larger number of assignments of PATHS that terminate at NEAR nodes than is necessary. Each assignment of COSTPATH involves a heap operation which takes $\theta(\log n)$ time, and these unnecessary heap operations can be avoided.

In the new algorithm, when Spira's algorithm would update a node K's PATH value, we do a bounded search of K's edge list attempting to find the next-cheapest edge to a FAR node. This search terminates when either $\lceil \log n \rceil$ edges have been examined or when an edge from K to a FAR node has been found. END (K) is set to either the FAR node discovered or the endpoint of the last edge encountered. The algorithm SHORT is identical to that of SPIRA with the modified UPDATE procedure of Fig. 3. The algorithm may be easily proven correct using Lemma 2.

```
UPDATE (K):
  If NEAR ≠ V then begin
    POINTER (K) ← POINTER (K) + 1;
    COUNT ← 1;
    While COUNT < ⌈log n⌉ and TAIL (K, POINTER (K)) ∈ NEAR do begin
      POINTER (K) ← POINTER (K) + 1;
      COUNT ← COUNT + 1 end;
    END (K) ← TAIL (K, POINTER (K));
    COSTPATH (K) ← MIN (SOURCE, K) + COST (K, POINTER (K)) end;
  End UPDATE.
```

FIG. 3. *Revised* UPDATE *for* SHORT.

Having UPDATE search only $\lceil \log n \rceil$ edges guarantees that each iteration of LOOP still executes in total time $O(\log n)$. In the next section we obtain bounds on the expected number of times LOOP is executed in SHORT under suitable probability distributions and implementation restrictions.

4. Analysis of the algorithm. Algorithm SHORT can easily be shown to take $\Omega(n^2 \log n)$ time in the worst case; the average time it takes depends both on the probability distribution we place on the set of graphs and on the fairness of the subroutine MINPATH. If several NEAR nodes have equal and minimal COSTPATH values, biased tie-breaking by MINPATH can lead to poor execution time of this and similar algorithms (see [2], [3]). Accordingly, we impose the following "fairness" constraint on MINPATH:

P1) In the case of nodes with equal COSTPATH value, the value J_0 returned by MINPATH is independent of the value of END (J_0) and depends only on the sequence of previous queue operations.

In the proof below, we assume that, in the case of ties, the value J_0 returned is the node of minimal index with minimal COSTPATH. A similar proof would work with any tiebreaking rule satisfying P1. More will be said about MINPATH in § 5.

We characterize the class of probability measures for which our average-time analysis holds as follows. Let $\mathcal{G}_n$ be the set of n node graphs, and suppose P is a probability measure on $\mathcal{G}_n$. Through the representation of adjacency matrices, we may identify $\mathcal{G}_n$ with the set of all $n \times n$ matrices with entries in $\mathcal{R} = \{r \in \mathbb{R} | r \geqq 0\} \cup \{\infty\}$. P is uniquely characterized by its distribution function $F_P : \mathcal{G}_n \to [0, 1]$ defined by $F_P(G) = P(\{G' \in \mathcal{G}_n | c_{G'}(i, j) \leqq c_G(i, j) \text{ for } 1 \leqq i, j \leqq n\})$ for any $G \in \mathcal{G}_n$.

Suppose i, j, and k are arbitrary nodes of V. We define the mapping $\langle i, j, k \rangle : \mathcal{G}_n \to \mathcal{G}_n$ by defining the graph $\langle i, j, k \rangle(G)$ to be identical to G with the exception that the costs of the edges (i, j) and (i, k) are interchanged. The set of functions $\{\langle i, j, k \rangle | i, j\, k \in V\}$ generates a group $\mathcal{T}_n$ of transformations on $\mathcal{G}_n$ under the operation of composition.

We call a probability measure *endpoint-independent* if every $\tau \in \mathcal{T}_n$ is a measure-preserving transformation on the probability space $(\mathcal{G}_n, P)$; that is, if $F_P(G) =$

$F_P(\langle i, j, k\rangle(G))$ for all $G \in \mathcal{G}_n$ and all $i, j, k \in V$. Intuitively this means that exchanging endpoints of edges from any fixed node doesn't affect the probability.

Many natural probability measures on $\mathcal{G}_n$ are endpoint-independent. For example, for each $i \in \{1, 2, \cdots, n\}$, if P_i is an arbitrary probability measure on $\mathcal{R}$, define P to be the measure on $\mathcal{G}_n$ obtained by selecting each entry of the adjacency matrix $c(i, j)$ independently according to distribution P_i. Then P, being the product measure generated by the P_i, is endpoint-independent. This class includes those distributions defined by Spira [15].

As another general example, if R is an arbitrary probability measure on a set S, and $f: \mathcal{G}_n \to S$ is a function such that $f \circ \tau = f$ for each $\tau \in \mathcal{T}_n$, then the induced probability measure $R \circ f$ on $\mathcal{G}_n$ is endpoint-independent. Specific such examples include having a distribution on the weights of edges from each node, a distribution on the sum of the weights of the edges leaving each node, or a distribution on the maximum weight of any edge in the graph, with the property that each arrangement of endpoints be equally likely.

The primary property of endpoint-independent measures which we exploit is the following. Suppose a particular edge is selected from the sorted edge list L_i by virtue of either its position on the list or the value of its cost. If P is endpoint-independent, then the endpoint of this edge is independent of the edge's selection; every endpoint is equally likely. Note that this is true even if the graph has edges of equal weight by our requirement that equal-weight edges be stored with endpoints in random order.

For any real-valued function f on the probability space $(\mathcal{G}_n, P)$, let $E_P(f)$ be the expected value of f. If SOURCE is a fixed node, for any $G \in \mathcal{G}_n$, we define $T(G)$ to be the time taken by algorithm SHORT (SOURCE) on G. The main result of this paper is the following.

THEOREM. *If P is an endpoint-independent probability measure on $\mathcal{G}_n$, and* SOURCE *is a fixed node, then*

$$E_P(T) = O(n \log n \log^* n).$$

Proof. In the remainder of this section, we assume that $n > 1$, that P is an endpoint-independent probability measure and that SOURCE is a fixed node. The theorem is established by obtaining a bound on the expected number of calls to MINPATH in SHORT. We partition these calls to MINPATH into two classes and establish bounds on each class separately. Define a call to MINPATH which returns node J *useful* if, at the previous time node J's PATH had been established in UPDATE, END $(J) \notin$ NEAR *at the time of assignment* of END (J). This corresponds to a search in UPDATE which discovers an edge from J to a FAR node. All other calls to MINPATH will be termed *useless*.

To bound the number of useless calls to MINPATH, we first show that the portion of the sorted adjacency list examined in algorithm SHORT is not much larger than that examined in SPIRA.

Suppose $G \in \mathcal{G}_n$. Define t_i to be the point in the execution of SHORT on G at which MINPATH is called for the ith time. We define the following parameters of the execution of SHORT:

J_i is the node returned by MINPATH at time t_i

E_i is the value of END (J_i) at t_i

NEAR_i is the value of NEAR at t_i

STATUS_i is the triple $(J_i, E_i, \text{NEAR}_i)$

OLDNEAR_i is the value of NEAR at the most recent call to UPDATE (J_i) prior to t_i

$\text{POINTER}_i(K)$ is, for any node K, the value of $\text{POINTER}(K)$ at t_i

$\text{PATHCOST}_i(K)$ is, for any node K, the value of $\text{PATHCOST}(K)$ at t_i.

Define $\text{TRACE}(G)$ to be the ordered sequence $(\text{STATUS}_i \mid 1 \leqq i \leqq \text{TL}(G))$, where $\text{TL}(G)$ is the number of times LOOP is executed in running algorithm SHORT on G.

In a similar fashion, we define symbols t_i^{SPIRA}, $\text{TRACE}^{\text{SPIRA}}(G)$, etc. for the computation of SPIRA on G.

LEMMA 3. $\text{TRACE}(G)$ *is a subsequence of* $\text{TRACE}^{\text{SPIRA}}(G)$. *Moreover, if* $\phi : \{1, 2, \cdots, \text{TL}(G)\} \to \{1, 2, \cdots, \text{TL}^{\text{SPIRA}}(G)\}$ *is the correspondence, then*

$$\text{POINTER}_{\phi(i)}^{\text{SPIRA}}(K) \leqq \text{POINTER}_i(K) < \text{POINTER}_{\phi(i)}^{\text{SPIRA}}(K) + \log n$$

for all $K \in V$.

Proof. By induction on t_i. The initial call to UPDATE (SOURCE) in SHORT sets POINTER (SOURCE) to either 1 or 2 depending on whether the edge from SOURCE to itself is cheapest among all edges from SOURCE. The first call to MINPATH in SHORT yields $J_1 = \text{SOURCE}$, $\text{NEAR}_1 = \{\text{SOURCE}\}$, and $E_1 =$ the endpoint of the shortest nonreflexive edge from SOURCE. This is either $\text{STATUS}_1^{\text{SPIRA}}$ or $\text{STATUS}_2^{\text{SPIRA}}$. The bounds on POINTERs are easily verified in the basis case.

Inductively, suppose the correspondence between STATUS_{i-1} and $\text{STATUS}_{\phi(i-1)}^{\text{SPIRA}}$ have been established. Since $\text{NEAR}_{i-1} = \text{NEAR}_{\phi(i-1)}^{\text{SPIRA}}$, either both algorithms terminate or both continue computing after this point. If computation continues, define

$$\phi(i) = \text{least } t > \phi(i-1) \text{ such that } \text{POINTER}_t^{\text{SPIRA}}(J_t^{\text{SPIRA}}) = \text{POINTER}_i(J_t^{\text{SPIRA}}).$$

We know that such a t exists due to three facts:

1) $\text{POINTER}_{\phi(i-1)}^{\text{SPIRA}}(K) \leqq \text{POINTER}_{i-1}(K)$ for all $K \in V$ (by induction),
2) $\text{TAIL}(K, p) \in \text{NEAR}$ for $\text{POINTER}_{\phi(i-1)}^{\text{SPIRA}}(K) \leqq p < \text{POINTER}_i(K)$ for all $K \in V$ (by the operation of SHORT),
3) SPIRA terminates only when all nodes have been included in NEAR.

If we let $J_0 = J_{\phi(i)}^{\text{SPIRA}}$, then since SPIRA increments POINTERS by 1 at each UPDATE we know

$$\text{POINTER}_{\phi(i)}^{\text{SPIRA}}(K) \leqq \text{POINTER}_i(K) \text{ for all } K \neq J_0.$$

Hence

$$\text{PATHCOST}_{\phi(i)}^{\text{SPIRA}}(K) \leq \text{PATHCOST}_i(K) \text{ for all } K \neq J_0,$$

and

$$\text{PATHCOST}_{\phi(i)}^{\text{SPIRA}}(J_0) = \text{PATHCOST}_i(J_0).$$

Thus J_0 is also the node of minimum PATHCOST and minimum index in SHORT, and is returned by MINPATH at t_i in SHORT. Thus $\text{STATUS}_i = \text{STATUS}_{\phi(i)}^{\text{SPIRA}}$.

The bounds on POINTERs follow from the operation of UPDATE in both algorithms. □

We can use Lemma 3 to bound the number of edges on edge lists examined by SHORT. For each node K, let $\text{LEN}(K, G)$ denote the value of $\text{POINTER}_{\text{TL}(G)}(K)$ upon termination of SHORT. Let

$$\text{LEN}(G) = \sum_{K=1}^{n} \text{LEN}(K, G).$$

Similarly define $\text{LEN}^{\text{SPIRA}}$.

In [3], [15] is proven the following bound on the average behavior of SPIRA.

LEMMA 4. $E_P(\text{LEN}^{\text{SPIRA}}) \leqq n \log n + O(n)$.

COROLLARY 5. $E_P(\text{LEN}) \leqq 2n \log n + O(n)$.

Proof. Upon termination at time $\tau = \text{TL}(G)$, by Lemma 3 we know that

$$\text{POINTER}_\tau(K) < \text{POINTER}^{\text{SPIRA}}_{\Phi(\tau)}(K) + \log n$$

for all $K \in V$. Hence

$$\text{LEN}(K, G) < \text{LEN}^{\text{SPIRA}}(K, G) + \log n$$

for all $K \in V$, and summing over all nodes we obtain

$$\text{LEN}(G) < \text{LEN}^{\text{SPIRA}}(G) + n \log n.$$

Taking expectations of both sides establishes the result. □

We can proceed to establish bounds on $E_P(\text{TL})$. Given $G \in \mathcal{G}_n$, let

$A(G)$ = the number of useless calls to MINPATH in TRACE (G),

$B(G)$ = the number of useful calls to MINPATH in TRACE (G).

LEMMA 6. $E_P(A) \leqq 2n + o(n)$.

Proof. Each useless call of MINPATH which returns J incremented POINTER (J) by $\log n$ in the immediately previous call to UPDATE (J). Hence

$$A(G) \log n \leqq \text{LEN}(G)$$

and Corollary 5 establishes the bound on the expectation. □

We use the following fact from [7, p. 224].

LEMMA 7. *Suppose we have a sequence of independent trials with possible outcomes in* {SUCCESS, FAIL}, *and the probability of* SUCCESS *is p. Then the expected number of trials until the first* SUCCESS *is* p^{-1}.

We now bound the number of useful calls to MINPATH. While a useful call to MINPATH which returns J guarantees that END $(J) \notin$ NEAR at the time of assignment of PATH (J), there is no guarantee that when J is returned by MINPATH a new node will be added to NEAR. However, this happens with sufficiently high probability as demonstrated by the following.

LEMMA 8. $E_P(B) \leqq 2n \log^* n + O(n)$.

Proof. Let

$$f(i) = e^{e^{\cdot^{\cdot^{\cdot^{e}}}}\Big\}i},$$

and define

$$n_0 = n - 1,$$

$$n_i = \left\lfloor \frac{n}{f(i)} \right\rfloor \text{ for } i = 1, 2, \cdots, \log^* n.$$

Suppose $G \in \mathcal{G}_n$. For $0 \leqq i \leqq \log^* n - 1$, define $\Psi(i)$ to be the least t such that, in TRACE (G), $|\text{NEAR}_t| = n - n_i$. Define $\Psi(\log^* n) = \text{TL}(G) + 1$. If $B_i(G)$ denotes the number of useful calls to MINPATH during the period of time $\mathcal{T}_i = \{\Psi(i), \Psi(i)+1, \cdots, \Psi(i+1)-1\}$, then

$$B(G) = \sum_{i=0}^{\log^* n - 1} B_i(G). \tag{1}$$

To bound $E_P(B_i)$, suppose i is fixed. We will say a time t in $\mathcal{T}_i$ is *favorable* if the call to MINPATH at time t is useful and if node J_t was returned by MINPATH at an earlier time *in* $\mathcal{T}_i$. Let $\mathcal{S}_i = (s_1, s_2, \cdots, s_l)$ denote the ordered sequence of favorable times in $\mathcal{T}_i$. Observe that

$$B_i(G) \leqq l + n, \tag{2}$$

since the first time a node is returned by MINPATH in $\mathcal{T}_i$ is not a favorable time.

For any $t \in \mathcal{S}_i$, the fact that J_t is returned by MINPATH at time t is independent of E_t due to our fairness constraint on MINPATH. Since P is endpoint-independent, when the assignment of E_t to END (J_t) was made, every node not in OLDNEAR_t had an equal probability of being assigned to END (J_t). Hence the probability that, at time t, a new element is added to NEAR is at least

$$\frac{n - |\text{NEAR}_t|}{n - |\text{OLDNEAR}_t|}.$$

Since t is favorable, we know that $\text{NEAR}_{\Psi(i)} \subseteq \text{OLDNEAR}_t \subseteq \text{NEAR}_t$, so this probability is at least

$$\frac{n - |\text{NEAR}_t|}{n - |\text{NEAR}_{\psi(i)}|} = \frac{n - |\text{NEAR}_t|}{n_i}.$$

Thus, the expected number of favorable times in $\mathcal{S}_i$ until a new node is added to NEAR is no greater than $n_i/(n - |\text{NEAR}_t|)$ by Lemma 7. Thus,

$$E_P(l) \leqq \frac{n_i}{n_i} + \frac{n_i}{n_i - 1} + \frac{n_i}{n_i - 2} + \cdots + \frac{n_i}{n_{i+1} + 1} = n_i \left(\sum_{k = n_{i+1}+1}^{n_i} \frac{1}{k} \right).$$

If $i = \log^* n - 1$, then $n_i \leqq n/\log n$, so

$$E_P(l) \leqq n_i \, (\log (n_i) + 1) \leqq (n/\log n)(\log (n/\log n) + 1) < 2n.$$

Otherwise,

$$E_P(l) \leqq n_i \, (\log (n_i) - \log (n_{i+1})).$$

If $i > 0$, then

$$E_P(l) \leqq \frac{n}{f(i)} (\log (f(i+1)) - \log (f(i)) + 1)$$

$$= \frac{n}{f(i)} (f(i) - f(i-1) + 1) \leqq n.$$

If $i = 0$, then

$$E_P(l) \leqq (n-1) \left(\log (n-1) - \log \left(\left\lfloor \frac{n}{e} \right\rfloor \right) \right)$$

$$\leqq (n-1)(\log (n-1) - \log n + 2) < 2n.$$

Equations (1) and (2) imply that

$$E_P(B) < 2n(\log^* n - 1) + 6n,$$

thus establishing the lemma. □

Proof of main theorem. Suppose $G \in \mathcal{G}_n$. There is a constant c such that

$$T(G) \leqq c \log n TL(G) + O(n)$$

$$= c \log n (A(G) + B(G)) + O(n),$$

where the $\log n$ factor accounts for the heap operation and edge list searches invoked in each iteration of LOOP. Lemmas 6 and 8 establish the result. □

COROLLARY 9. *The shortest distance matrix MIN may be computed for a graph in any of the indicated formats in expected time* $O(n^2 \log n \log^* n)$ *for any endpoint independent probability measure on* G_n.

Proof. Algorithm ALLPATHS works as follows. We first convert to the required sorted adjacency list format, which can be done using heapsort [1] in $O(n^2 \log n)$ time on the average. Then SHORT (SOURCE) is executed for each SOURCE $\in V$ for a total expected time of $O(n^2 \log n \log^* n)$ by the main theorem. □

5. Implementation details. Algorithm ALLPATHS may easily be modified to compute an actual path of shortest cost between any pair of nodes. To do this, an $n \times n$ matrix PREV of node indices is added with initial value PREV $(I, J) = I$. When node $E =$ END (J) is added to NEAR, PREV (SOURCE, E) is set to J. Following the execution of ALLPATHS, if we define

$K_0 = K$,

$K_{i+1} =$ PREV (SOURCE, K_i) for $i \geqq 0$,

$l =$ least i such that $K_i =$ SOURCE,

then a simple inductive argument demonstrates that $K_l, K_{l-1}, \cdots, K_0$ is a path of least cost from SOURCE to K.

As specified earlier, we required that edges of equal weight on an adjacency list L_i be in random order. In fact, the algorithm computes correctly without this assumption, and the randomness was only necessary for the proofs of timing in § 4. The running time of search-type algorithms can vary tremendously if this requirement is relaxed. For example, in [3] it is shown that, if the edges of equal weight are stored in increasing order of the indices of their endpoints, then quite natural implementations of Spira's algorithm can have $\Omega(n^3 \log n)$ average running time rather than the $O(n^2 \log^2 n)$ time originally claimed. A similar phenomenon will hold for this algorithm.

To ensure that algorithm SHORT will operate in the stated average time when the edges of equal weight are not presented in random order, we make one additional requirement on the priority queue of COSTPATH values, namely

P2) if J is returned by MINPATH, and the immediately subsequent call to UPDATE (J) does not change the value of COSTPATH (J), then the next call to MINPATH also returns J as value.

By implementing the priority queue as a heap, both P1 and P2 may be satisfied. A straightforward extension of the proofs of § 4, similar to [2], [13], show that under these criteria the main theorem and its corollary hold regardless of the order of edges of equal weight.

6. Empirical studies. Earlier studies [4] have reported that Yen's implementation of Dijkstra's algorithm [16] has the least average time of published algorithms for solving the all-pairs shortest-path problem on nonnegatively weighted directed graphs, for graphs with fewer than 100 nodes. Spira's algorithm was reported superior on the average to Yen's on larger graphs. An empirical study of the efficiency of ALLPATHS was undertaken. In this study, each graph was presented as an adjacency matrix, and the shortest distance matrix MIN was computed for each graph. For Spira's algorithm and ALLPATHS the sorted adjacency matrix was obtained by using a hybrid of *quicksort* and *straight insertion sort* [14]. The priority queue was implemented as a heap with external leaves. All procedure calls with the exception of

quicksort were coded in-line. The algorithms were written in Algol-60 and executed on the UNIVAC 1100/82 computer using the NUALGOL compiler.

Graphs of varying size were randomly generated with varying percentages of missing edges. Edge weights were assigned randomly, uniformly chosen from the integers from 0 to 100. Table 1 gives some results of this comparison with Spira's and Yen's[4] algorithms on graphs in which all edges are present, with 100 graphs of each size being tested.

This data suggests that the earlier data on Spira's algorithm may be overly pessimistic as, in this study, its expected time is superior to Yen's algorithm on graphs with 35 or more nodes, rather than the crossover point of 100 nodes previously reported. The most likely factor in this discrepancy is the fact that most subroutines were coded in-line, avoiding the overhead of stack manipulation.

As can be seen from Table 1, the average time of ALLPATHS is superior to Spira's and Yen's algorithms for graphs with at least 25 nodes. Further studies were undertaken to study the effect of changing the density of the graph. As the density of the graphs decreases, the point at which ALLPATHS is superior to Yen's gradually increases; with only 10% of the possible edges being present, the new algorithm is superior on the average to Yen's for graphs with 35 or more nodes.

TABLE 1
Comparison of three algorithms.
CPU *time in msec, all edges present, uniform distribution of edge weights.*

Number of nodes	YEN			SPIRA			ALLPATHS		
	Ave.	Min.	Max.	Ave.	Min.	Max.	Ave.	Min.	Max.
5	2	2	2	3	3	4	4	3	5
10	14	13	15	21	17	27	21	18	25
15	42	41	46	59	46	87	55	45	73
20	101	99	104	139	102	245	122	100	188
25	187	177	198	222	160	380	185	153	266
30	306	301	326	320	238	481	265	226	341
35	483	473	518	468	366	677	380	330	524
40	706	695	762	671	515	1098	535	450	755

7. Conclusion. We have presented an algorithm for finding all shortest paths in a nonnegatively weighted directed graph which has average time $O(n^2 \log n \log^* n)$ over a wide class of probability distributions on graphs. Empirical studies indicate that for the all-pairs shortest-path problem ALLPATHS is faster on the average than previously published algorithms for this problem for graphs of at least 25 nodes.

Fredman [9] has shown that a different modification of Spira's algorithm solves the problem with only $O(n^2 \log n)$ *comparisons* on the average, but at the expense of increasing the total running time to $\Omega(n^3)$ on the average. An interesting open question is whether there exists an $O(n^2 \log n)$ average-time algorithm for this problem. Frieze [11] has recently announced such an algorithm if the edge costs are drawn from a uniform [0, 1] distribution.

Acknowledgments. I would like to thank Albert Meyer, Richard Goldstein, Andrew Yao, and the referees for their helpful comments. David Goldhirsch and Richard Farina assisted in the empirical studies of this algorithm.

[4] Yen's algorithm was modified slightly to take advantage of missing edges in the graph.

REFERENCES

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

[2] P. BLONIARZ, A. MEYER AND M. FISCHER, *Some observations on Spira's shortest path algorithm*, Computer Science Dept. Tech. Report 79-6, State University of New York at Albany, 1979.

[3] P. BLONIARZ, M. FISCHER AND A. MEYER, *A note on the average time to compute transitive closures*, Third International Colloquium on Automata, Language and Programming, S. Michaelson and R. Milner, eds., Edinburgh University Press, Edinburgh, 1976, pp. 425–434.

[4] J. CARSON AND A. 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., 2 (1960), pp. 187–190. The method also appears in G. B. Dantzig, Linear Programming and Extensions, Princeton University Press, Princeton, NJ, 1963.

[6] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 260–271.

[7] W. FELLER, *An Introduction to Probability Theory and Its Applications*, Vol. 1, 3rd edition, John Wiley, New York, 1968.

[8] R. W. FLOYD, *Algorithm 97: Shortest path*, Comm. ACM, 5 (1962), p. 345.

[9] ———, *On the decision tree complexity of the shortest path problems*, 16th IEEE Symposium on Foundations of Computer Science, 1975, pp. 98–99.

[10] M. FREDMAN, *New bounds on the complexity of the shortest path problem*, this Journal, 5 (1976), pp. 83–89.

[11] A. M. FRIEZE, *On random shortest path problems*, Dept. Computer Science and Statistics Report, Queen Mary College, London, 1982.

[12] R. KARP AND R. E. TARJAN, *Linear expected-time algorithms for connectivity problems*, J. Algorithms, 1 (1980), pp. 374–393.

[13] C. P. SCHNORR, *An algorithm for transitive closure with linear expected time*, this Journal, 7 (1978), pp. 127–133.

[14] R. SEDGEWICK, *Implementing quicksort programs*, Comm. ACM, 21 (1978), pp. 847–857.

[15] P. SPIRA, *A new algorithm for finding all shortest paths in a graph of positive edges in average time* $O(n^2 \log^2 n)$, this Journal, 2 (1973), pp. 28–32.

[16] J. YEN, *Finding the lengths of all shortest paths in N-node nonnegative-distance complete networks using* $\frac{1}{2}N^3$ *additions and* N^3 *comparisons*, J. Assoc. Comput. Mach., 19 (1972), pp. 423–424.

SIAM J. COMPUT.
Vol. 12, No. 3, August 1983

0097-5397/83/1203-0014 $01.25/0

ON THE COMPLEXITY OF GENERAL GRAPH FACTOR PROBLEMS*

D. G. KIRKPATRICK† AND P. HELL‡

Abstract. For arbitrary graphs G and H, a G-factor of H is a spanning subgraph of H composed of disjoint copies of G. G-factors are natural generalizations of 1-factors (or perfect matchings), in which G replaces the complete graph on two vertices. Our results show that the perfect matching problem is essentially the only instance of the G-factor problem that is likely to admit a polynomial time bounded solution. Specifically, if G has any component with three or more vertices, then the existence question for G-factors is NP-complete. (In all other cases the question can be resolved in polynomial time.)

The notion of a G-factor suggests a natural generalization where G is replaced by an arbitrary family of graphs. This generalization gives rise not only to further NP-completeness results but also to new polynomial algorithms and duality theorems extending results of the traditional theory of matching. An indication of the nature and scope of these new results is presented.

Key words. algorithms, complexity, factor, graph, matching, NP-completeness

1. Introduction.

Let H denote an arbitrary graph with vertex set $V(H)$ and edge set $E(H)$. A *matching* in H is any subset M of $E(H)$ such that no two elements of M have a vertex in common. A matching M is *perfect* (also called a 1-factor) if exactly one element of M is incident with each vertex in $V(H)$. If H is a weighted graph, then the weight of a matching M is just the sum $\sum_{e\in M}$ weight (e).

The notion of a matching in a graph has numerous applications in such diverse areas as transversal theory, assignment problems, network flows, multiprocessor scheduling, shortest path algorithms, and the Chinese postman and traveling salesman problems [3], [10], [12], [13], [16], [17], [22], [31], [33], [36]. The existence of polynomial time bounded algorithms for the construction of matchings of maximum cardinality (and hence determining the existence of perfect matchings) or maximum weight is well known [6], [7], [8], [9], although the exact complexity of the problems is not yet settled and work continues on this aspect of the problem [23], [26]. In addition there is a rich mathematical theory that has developed about the matching problem that includes characterizations of graphs that admit perfect matchings [39] and (more generally) duality theorems on maximum matchings [1], [8], [12], [17], [32], [33], [34].

A matching in H may be viewed as a collection of disjoint subgraphs of H, each isomorphic to K_2.[1] In a perfect matching the vertex set $V(H)$ is completely partitioned by the vertex sets of the subgraphs. This suggests the following natural generalization: Let G be an arbitrary graph. A *G-packing* of a graph H is a set $\{G_1, \cdots, G_d\}$ of disjoint subgraphs of H such that each G_i is isomorphic to G. (Note that we do not require the G_i's to be induced subgraphs; that variant of the problem is discussed later in the paper). A perfect G-packing or *G-factor* of a graph H is a G-packing such that the sets $V(G_i)$ partition $V(H)$. Clearly, a K_2-packing is just a matching and a K_2-factor is a perfect matching.

* Received by the editors August 17, 1981, and in final revised form September 8, 1982. A preliminary version of the main result was presented at the Tenth Annual ACM Symposium on Theory of Computing, [30].

† Department of Computer Science University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5. The research of this author was supported by the Natural Sciences and Engineering Research Council grant A3583.

‡ Departments of Computing Science and Mathematics, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6. The research of this author was supported by the Natural Sciences and Engineering Research Council grant A5075.

[1] K_t denotes the complete graph on t vertices.

We were motivated to study this generalization of matching by both practical and theoretical considerations. Graph partitioning problems arise in a number of applications [2], [4], [11], [15], [21], [24], [29]. Our original motivation [19], [30] stemmed from the study of examination scheduling. After an assignment of courses to examination periods, eliminating what could be called first-order conflicts (essentially a graph coloring problem), has been accomplished, the problem arises of assigning the examination periods to real time periods, under fairly standard constraints (normally some k examination periods are scheduled in sequence each day). The objective here is to minimize second-order conflicts (or inconveniences). Typically, this might include an occurrence of a student writing two examinations on the same day or perhaps two consecutive examinations on the same day. Suppose H is the graph whose vertex set is the set of examination periods $\{p_i | 1 \leqq i \leqq t\}$ and whose edge (p_i, p_j) is weighted by the number of students common to courses examined in periods p_i and p_j. Then minimization of the types of second-order conflicts illustrated above corresponds to the construction of minimum weight K_3- and $P_3{}^2$-factors in H.

As we shall see, our investigation of generalized matching can also be viewed as furthering our understanding of the perceived threshold between NP-complete and polynomial time solvable problems [5], [14], [27], [28]. Specifically, it is of interest to know which members of this family of problems admit polynomial time bounded algorithms and which, like the general subgraph isomorphism problem of which they are all special instances, are NP-complete. We are able to provide a complete characterization (in the above sense) of the complexity of finding a G-factor. This characterization is similar, in spirit, to the results of Schaefer [38], Yannakakis [40] and Lewis [35] each of which establishes NP-completeness results over a broad family of interesting problems.

While our results here are essentially negative, it should be noted that an extension of the notion of G-packings and G-factors (replacing G by a family of graphs) has pointed the way to a very natural setting in which to extend the traditional theory of matching, giving rise to new polynomial algorithms and simple duality results [19], [20].

2. Generalizations of matching. Our notion of a G-packing (and G-factor) is by no means the only natural extension of the familiar concept of matching. Indeed, in § 5, we introduce and motivate the notion of a $\mathcal{G}$-packing (and $\mathcal{G}$-factor), where $\mathcal{G}$ denotes a family of graphs. This extension subsumes, and should not be confused with, the notion of F-factor introduced by Muhlbacher [37].

If the concept of matching is extended in the natural way to hypergraphs the problem of determining the existence of a perfect matching is known to be NP-complete. Karp [27] describes what is probably the simplest version of this problem as three-dimensional matching:

INSTANCE: An integer p and a set $U \subseteq \{1, 2, \cdots, p\}^3$.

QUESTION: Is there a subset $W \subseteq U$ of cardinality p such that no two elements of W agree in any coordinate?

It should be clear that the k-dimensional matching problem (replace three by k above) is also NP-complete, when $k \geqq 3$. The two-dimensional matching problem is equivalent to the matching problem for bipartite graphs.

Expressed as a language recognition problem, the existence problem for G-factors, which we denote FACT(G), becomes:

INSTANCE: A graph H.

QUESTION: Does H admit a G-factor?

[2] P_t denotes the path on t vertices.

The problem FACT (K_1) is trivial since every graph admits a K_1-factor. Furthermore, FACT (K_2) is just the question of existence of a perfect matching, and hence, FACT (K_2) $\in P$. More generally, if $G = \alpha \cdot K_1 \cup \beta \cdot K_2$, that is the disjoint union of α copies of K_1 and β copies of K_2 (or, equivalently, if each connected component of G has at most two vertices), then H admits a G-factor if and only if $|V(H)|$ is divisible by $|V(G)|$ and H admits a matching with at least $\beta|V(H)|/|V(G)|$ edges. Thus the usual algorithms for finding a maximum matching (e.g., [7], [34]) may be used to answer FACT (G) in polynomial time. Our central result suggests that all other problems FACT (G) are unlikely to admit efficient solutions.

THEOREM 4.2. *If G is not of the form $\alpha \cdot K_1 \cup \beta \cdot K_2$, then* FACT ($G$) *is* NP-*complete.*

Two important instances of this result, $G = K_3$ and $G = P_3$, were established earlier by T. Schaeffer [14], [28] and D. S. Johnson [25].

The proof of Theorem 4.2 is deferred to § 4. The following lemma allows us to restrict our attention to problems FACT (G), where G is a connected graph.

LEMMA 2.1. *Let G be a graph and G' any component of G with the maximum number of edges. Then,* FACT (G') $\leqq_P$ FACT (G).

Proof. Suppose G' has p vertices and suppose G has r distinct components isomorphic to G'. If H is any graph with dp vertices, then let $T(H)$ denote the graph $H \cup d(G - G')$. Obviously, if H admits a G'-factor, then $T(H)$ admits a G-factor. Suppose $T(H)$ admits a G-factor F. F must contain exactly dr components isomorphic to G'. But, by the maximality of G', the restriction of F to $d(G - G')$ contains at most $d(r-1)$ components isomorphic to G'. Hence the restriction of F to H must be a G'-factor of H. Thus H admits a G'-factor if and only if $T(H)$ admits a G-factor. □

3. Basic modules and their properties. Our objective in this and the next section is to demonstrate how, for an arbitrary connected graph G on k vertices, the k-dimensional matching problem can be polynomially reduced to the problem FACT (G). Our construction is component based (cf. [14]) in nature; in this section we describe the components (which we call modules) and their properties that we exploit in the general construction.

3.1. Modules and coherences. A *module* is a graph M with nonempty subset $C \subseteq V(M)$ of distinguished vertices. We call the elements of C (respectively $V(M) - C$) *connector* vertices (respectively *interior* vertices) of M. A *G-module* is any module that admits a G-packing covering all of its interior vertices (plus some, possibly empty, subset of its connector vertices).

A *modular extension* of the module M is any graph H, containing M as an induced subgraph, in which no interior vertex of M is adjacent to a vertex of $H - M$ (that is, M is connected to the rest of H only through its connector vertices). Let $\pi = \{G_1, \cdots, G_d\}$ be any G-packing of some modular extension H of M. A vertex v of M is said to be *bound* to M by π, if $v \in V(G_i)$ implies $V(G_i) \subseteq V(M)$. A G-module M is *internally G-coherent* if every G-factor of every modular extension of M binds to M all of its interior vertices (that is, it respects the modularity of M).

The simplest example of internally G-coherent G-module is the (connected) graph G itself with any one vertex $v \in V(G)$ designated as a connector vertex. We depict this schematically in Fig. 1.

3.2. Diamond modules. If G is any connected graph and $v \in V(G)$, then the graph, formed from G by splitting v into two nonadjacent vertices v_a and v_b, each of which is adjacent to all of the neighbors of v in G, is called a *G-diamond* and is denoted $D[G; v]$. We depict $D[G; v]$ schematically in Fig. 2.

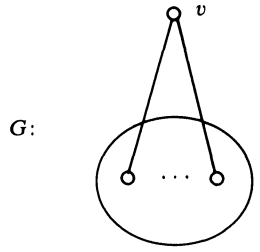


FIG. 1

If v_a and v_b are taken as connector vertices, then $D[G; v]$ is a G-module. Its coherence, it turns out, depends on the choice of vertex v, but a choice ensuring G-coherence always exists. Specifically, let v^* be any vertex of G that is not a cutpoint and belongs to a biconnected component of G containing at most one cutpoint. Every graph G is guaranteed to contain at least one such vertex (cf. [18, p. 36]).

LEMMA 3.1. *The module $D[G; v^*]$ with v_a^* and v_b^* as connectors is internally G-coherent.*

Proof. Let H be any modular extension of $D[G; v^*]$ and let φ be any G-factor of H. φ induces a partition π of the interior vertices of $D[G; v^*]$. Since $D[G; v^*]$ has exactly two connector vertices and each graph in φ is connected, π has at most two cells. All of the vertices of $D[G; v^*]$ that do not belong to the same biconnected component as v_a^* must belong to the same partition of π (otherwise there must be two vertex-disjoint paths from this set to v^* in G, contradicting the choice of v^*). Hence, if π has two cells then some element of φ must contain all of the vertices of $D[G; v^*]$ that do not belong to the same biconnected component as v_a^*, either v_a^* or v_b^*, and at least one vertex of $H - D[G; v^*]$. But this component has at least one more cutpoint (namely v_a^* or v_b^*) than G, a contradiction. Thus, π has exactly one cell and hence $D[G; v^*]$ is internally G-coherent. □

We can summarize the relevant properties of diamond modules as follows:

Property 3.2. (a) Every G-factor of every modular extension of $D[G; v^*]$ binds to $D[G; v^*]$ its interior vertices plus exactly one of its connector vertices.

(b) The graph $D[G; v^*]$ minus either one of its connector vertices admits a G-factor.

Thus, diamond modules, wherever they appear in a larger graph, force a "choice" of one or the other of their connector vertices.

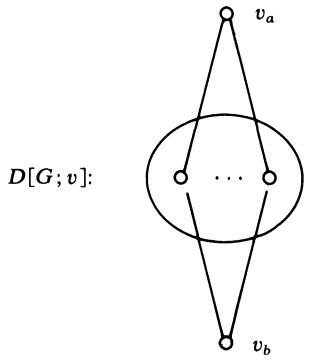


FIG. 2

3.3. Star modules. A *G-star*, denoted $S[G; v]$, is the graph formed from G by identifying, with each vertex $w \in V(G)$, the v_a-connector of a distinct copy of $D[G; v]$. If the v_b-connectors of the $|V(G)|$ G-diamonds used in the construction, relabelled as $x_1, \cdots, x_{|V(G)|}$, are taken as connector vertices, then $S[G; v]$ is a G-module. We depict $S[G; v]$ schematically in Fig. 3.

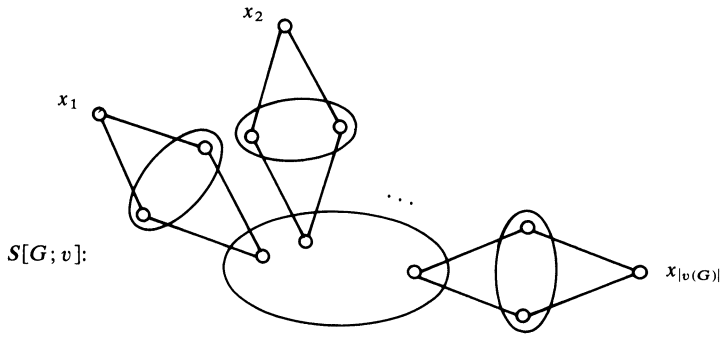


FIG. 3

$S[G; v]$ can be seen as a modular extension of $|V(G)|$ disjoint copies of $D[G; v]$. As would be expected the coherence of $S[G; v]$ depends on the coherence of $D[G; v]$. Specifically,

LEMMA 3.3. *If $D[G; v]$ is internally G-coherent, then so is $S[G; v]$.*

Proof. Let H be any modular extension of $S[G; v]$ and let φ be any G-factor of H. H must also be a modular extension of each of the $|V(G)|$ copies of $D[G; v]$ used in the construction of $S[G; v]$. Since $D[G; v]$ is internally G-coherent, it follows that all of the interior vertices of $S[G; v]$ that are internal to one of the copies of $D[G; v]$ must be bound to $S[G; v]$ by φ. Since none of the remaining interior vertices of $S[G; v]$ are adjacent to any of the connectors of $S[G; v]$, it follows that φ binds to $S[G; v]$ all of its interior vertices. □

COROLLARY 3.4. *$S[G; v^*]$ is internally G-coherent.*

In fact, $S[G; v^*]$ satisfies the following somewhat stronger property. A G-module M is *G-coherent* if it is internally G-coherent, if every G-factor of every modular extension of M binds to M either all or none of its connector vertices, and if in addition, both M and $M - C$ admit G-factors. G-coherence places a strong restriction on the G-modularity of M. It is clear from the definitions that G-coherent modules M, wherever they appear in a larger graph, can be viewed as forcing a "choice" of either all or none of their connector vertices (both of which are possible). Our central construction rests on the following:

LEMMA 3.5. *$S[G, v^*]$ is G-coherent.*

Proof. Let φ be any G-factor of any modular extension of $S[G, v^*]$. Note that $S[G, v^*]$ contains $|V(G)|^2$ vertices of which $|V(G)|$ are connectors. Since $S[G, v^*]$ is internally G-coherent and the total number of vertices bound to $S[G, v^*]$ by φ must be a multiple of $|V(G)|$, it follows that either all or none of the connector vertices must be bound to $S[G, v^*]$ by φ. □

4. The general construction. We are now prepared to state and prove our central lemma.

LEMMA 4.1. *If G is a connected graph, then $|V(G)|$-dimensional matching $\leqq_P$ FACT (G).*

Proof. Let p be any positive integer and $U \subseteq \{1, 2, \cdots, p\}^k$, where $k = |V(G)|$. It suffices to show how to construct (in polynomial time) a graph $R(U)$ with the property that $R(U)$ admits a G-factor if and only if U admits a k-dimensional matching.

$R(U)$ contains, among others, an independent set of kp vertices labelled by the pairs (i, j), where $1 \leqq i \leqq k$ and $1 \leqq j \leqq p$. For each k-tuple $(t_1, \cdots, t_k) \in U$, $R(U)$ contains a distinct copy of $S[G; v^*]$ whose k connector vertices are arbitrarily identified with the k vertices labelled $(1, t_1), \cdots, (k, t_k)$.

Suppose that U admits a k-dimensional matching W. We construct a G-factor φ of $R(U)$ as follows. To those copies of $S[G; v^*]$ associated with k-tuples in W, φ binds all of their vertices (in particular, their connector vertices). To all other copies of $S[G; v^*]$, φ binds only their interior vertices. Thus, φ binds the vertex (i, j) to the star module associated with that unique k-tuple in W containing j in position i. It follows that φ is a G-factor of $R(U)$.

Conversely, suppose that $R(U)$ admits a G-factor φ. We construct a k-dimensional matching W of U as follows. Call a copy of $S[G; v^*]$ in $R(u)$ "chosen" if φ binds to that copy all of its connector vertices. By Lemma 3.5 φ chooses exactly p of the star-modules in $R(u)$. Let W be the set of k-tuples associated with chosen star modules. Since each vertex (i, j) is bound to exactly one chosen star module, it follows that exactly one element of W contains j in its ith component. Hence W is a k-dimensional matching of U. □

We now restate and give a direct proof of our central result.

THEOREM 4.2. *If G is not of the form $\alpha \cdot K_1 \cup \beta \cdot K_2$, then* FACT (G) *is* NP-*complete.*

Proof. It is clear that all problems FACT (G) are in NP. By Lemma 2.1, it suffices to show that if G is a connected graph with at least three vertices, then FACT (G) is NP-complete. But this is immediate from Lemma 4.1 and the NP-completeness of k-dimensional matching, for $k \geqq 3$. □

Thus virtually all uniform factoring problems (with the exception of matching) are NP-complete. A similar characterization holds for what we call "strict" G-factors.

A G-packing (or G-factor) of H is *strict* if each G_i belonging to the packing is an induced subgraph of H. Corresponding to FACT (G) we have the question S-FACT (G) expressed as:

INSTANCE: A graph H.

QUESTION: Does H admit a strict G-factor?

Note that H admits a strict G-factor if and only if its complement $\bar{H}$ admits a strict $\bar{G}$-factor. Then S-FACT (G) and S-FACT $(\bar{G})$ are polynomially equivalent, and it is sufficient to consider problems S-FACT (G) for connected graphs G only. Clearly, if G has fewer than three vertices a polynomial algorithm for S-FACT (G) follows from algorithms for maximum matching (e.g., [7]). As with FACT (G) all other cases appear to be intractable.

THEOREM 4.3. *If G has at least three vertices, then* S-FACT (G) *is* NP*-complete.*

Proof. Observe that, in the proof of Lemma 4.1., the graph $R(U)$ admits a G-factor if and only if it admits a strict G-factor. (This follows from the construction of star modules.) Hence, Lemma 4.1 also proves that $|V(G)|$-dimensional matching $\leqq_P S$-FACT (G). Thus, the result follows from the obvious fact that S-FACT $(G) \in$ NP and the NP-completeness of k-dimensional matching for $k \geqq 3$. □

5. Family packings and factors. We introduced G-packings and G-factors as a generalization of conventional matchings and have reached the unfortunate conclusion

that this is an unlikely direction in which to generalize the rich theory—most notably the exsistence of polynomial time bounded algorithms—that is associated with the matching problem. However, a straightforward extension of the notion of G-packing suggests itself as another natural generalization of matching. While negative results still abound, this extension does give rise to a number of positive results which hint at a new generalized theory of matching including both polynomial algorithms and elegant duality results.

We extend the notion of G-packing by replacing G by a family $\mathcal{G}$ of "packing" graphs. A *$\mathcal{G}$-packing* of a graph H is a set $\{G_1, \cdots, G_d\}$ of disjoint subgraphs of H such that each G_i is isomorphic to some element of $\mathcal{G}$. A *$\mathcal{G}$-factor* is defined similarly. The existence problem for $\mathcal{G}$-factors, denoted FACT ($\mathcal{G}$), becomes:

INSTANCE: A graph H.

QUESTION: Does H admit a $\mathcal{G}$-factor:

As an example, if C_t denotes the cycle on t vertices and $\mathcal{G} = \{K_2, C_3, C_4, C_5, \cdots\}$, then FACT ($\mathcal{G}$) can be solved as an assignment problem, [33].

The NP-completeness of many problems FACT ($\mathcal{G}$) stems directly from our earlier constructions. As a simple example, consider:

Example 5.1. FACT ($\{K_t | t \geqq 3\}$) is NP-complete.

Proof. It suffices to observe that the graph used in our reduction of three-dimensional matching to FACT (K_3) contains no complete subgraphs of order four. Thus any $\{K_t | t \geqq 3\}$-factor must also be a K_3-factor. □

It is interesting to note that H has a $\{K_t | t \geqq 3\}$-factor if and only if its complement has a coloring in which each color class contains at least three vertices. This connection with coloring is explored in more detail in [19]. Example 5.1 is subsumed by the following theorem whose proof will appear elsewhere.

THEOREM 5.2. *Let $\mathcal{G}$ be any subset of $\{K_t | t \geqq 1\}$. If $K_1 \in \mathcal{G}$ or $K_2 \in \mathcal{G}$, then* FACT ($\mathcal{G}$) *is in* P, *otherwise* FACT ($\mathcal{G}$) *is* NP-*complete.*

One further example should help to substantiate our claim that the study of family factorizations is a fertile setting in which to generalize the traditional theory of matching.

Example 5.3. FACT ($\{K_{1,t} | t \geqq 1\}$) is in P.

Proof. It is straightforward to confirm that a graph H admits a $\{K_{1,t} | t \geqq 1\}$-factor if and only if it contains no isolated vertices.

The facility location (or domination number) problem [4], [11] can be viewed as trying to find a minimal $\{K_{1,t} | t \geqq 1\}$-factor. Our framework makes it natural to express the related problem of determining the existence of factors using only a restricted subset of facilities (star graphs). Example 5.3 is just one special case of the following:

THEOREM 5.4. *Let $\mathcal{G}$ be any subset of $\{K_{1,t} | t \geqq 1\}$. If for some $t \geqq 1$, $K_{1,t} \notin \mathcal{G}$ and $K_{1,t+1} \in \mathcal{G}$, then* FACT ($\mathcal{G}$) *is* NP-*complete. Otherwise* FACT ($\mathcal{G}$) *is in* P.

The proof of Theorem 5.4 also includes a duality result analogous to the theorems of Tutte [39] and Berge [1, p. 159] for star matchings; it will appear elsewhere. We have similar results for any set of complete bipartite graphs.

6. Conclusions. Our results completely classify the complexity of factorization problems. The vast majority of them are NP-complete. The few remaining cases can be reduced to the maximum matching problem and hence admit polynomial time algorithms. The results of § 5 suggest that the study of the more general family factorization problems is a fruitful direction in which to pursue a generalized theory of matching, including new polynomial algorithms.

REFERENCES

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

[2] F. T. BOESCH AND J. F. GIMPEL, *Covering the points of a digraph with point-disjoint paths and its application to code optimization*, J. Assoc. Comput. Mach., 24 (1977), pp. 192–198.

[3] N. CHRISTOFIDES, *Worst-case analysis of a new heuristic for the travelling salesman problem*, GSIA, Carnegie-Mellon University, 1976.

[4] E. COCKAYNE, S. GOODMAN AND S. HEDETNIEMI, *A linear algorithm for the domination number of a tree*, Inform. Processing Letters, 4 (1975), pp. 41–44.

[5] S. A. COOK, *The complexity of theorem-proving procedures*, Proc. Third ACM Symposium on Theory of Computing, pp. 151–158.

[6] W. H. CUNNINGHAM AND A. B. MARSH, III, *A primal algorithm for optimum matching*, Math. Programming Study, 8 (1978), pp. 50–72.

[7] J. EDMONDS, *Paths, trees, and flowers*, Canad. J. Math., 17 (1965), pp. 449–467.

[8] ———, *Maximum matching and a polyhedron with* (0, 1) *vertices*, J. Res. Nat. Bureau of Standards, 69B (1965), pp. 125–130.

[9] J. EDMONDS AND E. L. JOHNSON, *Matching: a well-solved class of integer linear programs*, in Combinatorial Structures and their Applications, R. K. Guy et al., eds., Gordon and Breach, New York, 1970, pp. 89–92.

[10] ———, *Matching, Euler tours, and the Chinese postman*, Math. Programming, 5 (1973), pp. 88–124.

[11] M. FARBER, *Domination and duality in weighted trees*, Congressus Numerantium, 33 (1981), pp. 3–13.

[12] L. R. FORD, JR. AND D. R. FULKERSON, *Flows on Networks*, Princeton University Press, Princeton, NJ, 1962.

[13] M. FUJII, T. KASAMI AND K. NINAMIYA, *Optimal sequencing of two equivalent processors*, SIAM J. Appl. Math., 17 (1969), pp. 784–789; Erratum, ibid., 20 (1971), p. 141.

[14] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, W. H. Freeman, San Fransisco, 1979.

[15] L. L. GARNISHTEYN, *The partitioning of graphs*, Engineering Cybernetics, 1 (1969), pp. 76–82.

[16] P. C. GILMORE AND R. E. GOMORY, *Sequencing a one-stage variable machine: a solvable case of the travelling salesman problem*, Oper. Res., 12 (1964), pp. 655–679.

[17] M. HALL, *Distinct representations of subsets*, Bull. Amer. Math. Soc., 54 (1948), pp. 922–926.

[18] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1968.

[19] P. HELL AND D. G. KIRKPATRICK, *Scheduling, matching and coloring*, in Algebraic Methods in Graph Theory, G. R. Szasz et al., eds., Colloq. Math. Soc. Janos Bolyai, Hungary, 1981.

[20] ———, *On generalized matching problems*, Inform. Processing Letters, 12 (1981), pp. 33–35.

[21] L. J. HERBERT, *Some applications of graph theory to clustering*, Psychometrika, 39 (1974), pp. 283–309.

[22] A. J. HOFFMAN AND H. M. MARKOWITZ, *A note on shortest path, assignment, and transportation problems*, Naval Res. Logist. Quart., 10 (1963), pp. 375–380.

[23] J. E. HOPCROFT AND R. M. KARP, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225–231.

[24] A. K. HOPE, *Component placing through graph partitioning in computer-aided printed-wiring-board design*, Electronic Letters, 8 (1972), pp. 87–88.

[25] D. S. JOHNSON, private communication, August 1977.

[26] O. KARIV, *An $O(n^{2.5})$ algorithm for finding maximum matching in a general graph*, Ph.D. Thesis, Weizmann Institute, Rehovot, Israel, 1976.

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

[28] ———, *On the complexity of combinatorial problems*, Networks, 5 (1975), pp. 45–68.

[29] B. W. KERNIGHAN AND S. LIN, *An efficient heuristic procedure for partitioning graphs*, Bell System Tech. J., 49 (1970), pp. 291–307.

[30] D. G. KIRKPATRICK AND P. HELL, *On the completeness of a generalized matching problem*, in Proc. Tenth Annual ACM Symposium on Theory and Computing, 1978, pp. 240–245.

[31] J. M. KLEIN AND H. TAKAMORI, *Parallel line assignment problems*, Management Sci. 19 (1972), pp. 1–10.

[32] D. KÖNIG, *Graphs and matrices*, Mat. Fiz. Lapok, 38 (1931), pp. 116–119.

[33] H. W. KUHN, *The Hungarian method for the assignment problem*, Naval Res. Logist. Quart., 2 (1955), pp. 83–97.

[34] E. L. LAWLER, *Combinatorial Optimization*, Holt, Rinehart and Winston, New York, 1976.

[35] J. M. LEWIS, *On the complexity of the maximum subgraph problem*, Proc. Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 265–274.

[36] L. MIRSKY, *Transversal Theory*, Academic Press, New York, 1971.

[37] J. MÜHLBACHER, *F-factors of graphs: a generalized matching problem*, Information Processing Lett., 8 (1979), pp. 207–214.
[38] T. J. SHAEFER, *The complexity of satisfiability problems*, Proc. Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 216–226.
[39] W. T. TUTTE, *The factorisation of linear graphs*, J. London Math. Soc., 22 (1947), pp. 107–111.
[40] M. YANNAKAKIS, *Node- and edge-deletion* NP-*complete problems*, Proc. Tenth Annual ACM Symposium on Theory of Computing, 1978, pp. 253–264.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0001 $01.25/0

THE RANKS OF $m \times n \times (mn-2)$ TENSORS*

M. D. ATKINSON† AND S. LLOYD†

Abstract. It is shown that there is essentially only one $m \times n \times (mn-2)$ tensor of rank $mn-1$. It is also proved that, except for this tensor, all $m \times n \times p$ tensors with $p \leqq mn-2$ have rank at most $mn-2$. The main tool is Kronecker's theory of matrix pencils which has already been applied directly by Ja'Ja' [SIAM J. Comput., 8 (1979), pp. 443–462] to study the ranks of $m \times n \times 2$ tensors. We show that each nondegenerate $m \times n \times (mn-2)$ tensor is determined by a related $m \times n \times 2$ tensor and apply the Kronecker theory to this related tensor.

Key words. computational complexity, bilinear forms, tensor rank

A classical problem in algebraic computational complexity is to determine the minimal number of nonscalar multiplications required to evaluate some set $\sum_{i,j} \alpha_{ijk} x_i y_j$, $k = 1,2, \cdots, p$, of bilinear forms in noncommuting variables $x_1, \cdots, x_m$, $y_1, \cdots, y_n$. This number can be variously described as the complexity of the set, the rank of the defining 3-tensor (α_{ijk}) or as the minimal number of rank 1 matrices whose linear span contains the $m \times n$ matrices $A_k = (\alpha_{ijk})$, $k = 1,2, \cdots, p$ [2].

The problem is rather trivial if $p = 1$ when the complexity is simply the matrix rank of A_1. It has also been completely solved if $p = 2$ for algebraically closed fields by the Kronecker theory of matrix pencils (see [3] which also summarizes Kronecker's results). For $p \geqq 3$ there is no corresponding theory and it seems unlikely that the tensor rank can in general be described in terms of simpler invariants of the tensor.

When working with some unknown $m \times n \times p$ tensor defined by $m \times n$ matrices $A_1, \cdots, A_p$ it is common to assume that the tensor is nondegenerate (3-nondegenerate in the terminology of [4], i.e., the matrices are linearly independent). If the tensor was degenerate it would reduce to one of size smaller than $m \times n \times p$. The assumption of nondegeneracy obviously implies that $p \leqq mn$; moreover, if $p = mn$, the tensor rank is mn. The observations in this paper centre on nondegenerate tensors over algebraically closed fields which have p just less than mn. The case $p = mn-1$ has already been treated in [1], where it was shown that every $m \times n \times (mn-1)$ tensor has rank at most $mn-1$ (and hence the nondegenerate tensors have rank precisely $mn-1$). It then follows that a nondegenerate $m \times n \times (mn-2)$ tensor has rank $mn-1$ or $mn-2$. We shall classify those $m \times n \times (mn-2)$ tensors of rank $mn-1$. Somewhat surprisingly it turns out that, among the infinitely many different tensors, only one of them has this rank. As a consequence we shall prove that every $m \times n \times (mn-3)$ tensor has rank at most $mn-2$ (this result was given in [1] but the proof, which at that time required very extensive calculations, was omitted). Both of these results depend on the following proposition for which we require two definitions:

1. A space of $m \times n$ matrices is said to be perfect if it is generated (as a vector space) by rank 1 matrices. (Notice that the above remarks imply that spaces of dimension mn or $mn-1$ are necessarily perfect.)

2. Two spaces $\mathscr{X}, \mathscr{Y}$ of $m \times n$ matrices are said to be equivalent if there exist nonsingular $m \times m$, $n \times n$ matrices P, Q such that

$$\mathscr{Y} = P\mathscr{X}Q = \{PXQ : X \in \mathscr{X}\}.$$

* Received by the editors August 25, 1981.
† Department of Computing Mathematics, University College, Cardiff, United Kingdom.

PROPOSITION. *If $\mathscr{X}$ is a vector space of $m \times n$ matrices of dimension $mn-2$, then $\mathscr{X}$ is perfect unless it is equivalent to the space of all matrices (x_{ij}) for which $x_{11}+x_{22}=0$ and $x_{12}=0$.*

The lemmas which lead up to the proof of this proposition all use the same general technique and notation. Let ϕ be the bilinear mapping on the space of all $m \times n$ matrices defined by

$$\phi(X, Y) = \text{trace}\,(XY')$$

(where Y' denotes the transpose of Y). With respect to ϕ every k-dimensional space $\mathscr{X}$ of $m \times n$ matrices has an annihilator space $\mathscr{X}^*$ of dimension $mn-k$, namely

$$\mathscr{X}^* = \{Y : \phi(X, Y) = 0\}.$$

Of course $\mathscr{X}^{**} = \mathscr{X}$. Also, a routine check shows that, if P and Q are nonsingular, the annihilator of $P\mathscr{X}Q$ is $(P')^{-1}\mathscr{X}^*(Q')^{-1}$ and hence equivalent spaces have equivalent annihilators. We eventually exploit this fact to prove the proposition by taking $\mathscr{X}^*$ to be defined by a pencil in Kronecker canonical form. In preparation for this our first three technical lemmas consider special cases for $\mathscr{X}^*$; these special cases are the building blocks of a general Kronecker canonical form. Note that the annihilator of the exceptional space in the proposition is generated by $E_{11}+E_{22}$ and E_{12} (where, in general, E_{ij} is the matrix with a 1 in the (i, j) position and zeros elsewhere).

LEMMA 1. *Let $\mathscr{A}$ be the (r^2+r-2)-dimensional space of $r \times (r+1)$ matrices whose annihilator is defined by the matrix pencil*

$$L_r = \alpha X + \beta Y = \begin{bmatrix} \alpha & \beta & & & 0 & \\ & \alpha & \beta & & & \\ & & & \ddots & \ddots & \\ & 0 & & & \alpha & \beta \end{bmatrix}.$$

Then $\mathscr{A}$ is perfect.

Proof. By definition $\mathscr{A}$ is the set of all matrices (a_{ij}) for which $\sum_{i=1}^{r} a_{ii} = \sum_{i=1}^{r} a_{i,i+1} = 0$. An obvious calculation verifies that the following r^2+r-2 matrices are all rank 1, linearly independent and satisfy the conditions for membership of $\mathscr{A}$:

$$\{E_{ij} : i \neq j, i+1 \neq j\}$$
$$\cup \{E_{ii} + E_{i,i+1} + E_{i,i+2} - E_{i+1,i} - E_{i+1,i+1} - E_{i+1,i+2} : i = 1,2,\cdots,r-1\}$$
$$\cup \{E_{ii} - E_{i,i+1} + E_{i,i+2} + E_{i+1,i} - E_{i+1,i+1} + E_{i+1,i+2} : i = 1,2,\cdots,r-1\}.$$

Similarly the transposed space $\mathscr{A}'$ of $(r+1) \times r$ matrices is also perfect.

LEMMA 2. *Let $\mathscr{A}$ be the space of $r \times r$ matrices whose annihilator is defined by the matrix pencil*

$$J_r(\lambda) = \alpha X + \beta Y = \begin{bmatrix} \alpha+\beta\lambda & \beta & & & 0 & \\ & \alpha+\beta\lambda & \beta & & & \\ & & & \ddots & \ddots & \\ & & & & & \beta \\ & 0 & & & & \alpha+\beta\lambda \end{bmatrix}.$$

Then, if $r \neq 2$, $\mathscr{A}$ is perfect.

Proof. The notation is intended to include the case $r=1$ when the lemma holds for trivial reasons. If $r=2$ $\mathscr{A}$ is not perfect (and this is why the proposition has an exceptional space). We now take $r>2$. In this case $\mathscr{A}$ is the set of all (a_{ij}) for which $\sum_{i=1}^{r} a_{ii} = \sum_{i=1}^{r-1} a_{i,i+1} = 0$ and $\mathscr{A}$ has dimension r^2-2. The following set is a basis of

rank 1 matrices:

$$\{E_{ij}: i \neq j, i+1 \neq j\}$$
$$\cup\{E_{ii}+E_{i,i+1}+E_{i,i+2}-E_{i+1,i}-E_{i+1,i+1}-E_{i+1,i+2}: i=1,2,\cdots,r-2\}$$
$$\cup\{E_{ii}-E_{i,i+1}+E_{i,i+2}+E_{i+1,i}-E_{i+1,i+1}+E_{i+1,i+2}: i=1,2,\cdots,r-2\}$$
$$\cup\{E_{r-2,r-1}-E_{r-2,r}+E_{r-1,r-1}-E_{r-1,r}+E_{r,r-1}-E_{rr}\}.$$

LEMMA 3. *Let $\mathscr{A}$ be the space of 4×4 matrices whose annihilator is defined by the matrix pencil*

$$J_2(\lambda)\oplus J_2(\mu)=\alpha X+\beta Y=\begin{bmatrix}\alpha+\beta\lambda & \beta & 0 & 0\\ 0 & \alpha+\beta\lambda & 0 & 0\\ 0 & 0 & \alpha+\beta\mu & \beta\\ 0 & 0 & 0 & \alpha+\beta\mu\end{bmatrix}.$$

Then $\mathscr{A}$ is perfect.

Proof. $\mathscr{A}$ is the set of all 4×4 matrices which satisfy $\sum_{i=1}^{4} a_{ii} = \lambda(a_{11}+a_{22})+\mu(a_{33}+a_{44})+a_{12}+a_{34}=0$. The following set is a basis of rank 1 matrices:

$\{E_{ij}: i \neq j, (i,j) \neq (1,2) \text{ or } (3,4)\}$

$$\cup\left\{\begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\\ \mu-\lambda & 0 & 0 & \mu-\lambda\\ -1 & 0 & 0 & -1\end{bmatrix},\begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 1 & 0 & 1\\ 0 & \mu-\lambda & 0 & \mu-\lambda\\ 0 & -1 & 0 & -1\end{bmatrix},\right.$$
$$\left.\begin{bmatrix}0 & 1 & 0 & 1\\ 0 & 0 & 0 & 0\\ 0 & -1 & 0 & -1\\ 0 & 0 & 0 & 0\end{bmatrix},\begin{bmatrix}0 & 1 & -1 & 1\\ 0 & 0 & 0 & 0\\ 0 & -1 & 1 & -1\\ 0 & -1 & 1 & -1\end{bmatrix}\right\}.$$

LEMMA 4. *Let $\mathscr{X}$ be a space of $m\times n$ matrices of dimension $mn-1$ or $mn-2$ all partitioned in some fixed way as $\begin{bmatrix}A & B\\ C & D\end{bmatrix}$ where A is an $r\times s$ matrix. Suppose that $\mathscr{X}$ contains all $m\times n$ matrices of the form $\begin{bmatrix}0 & B\\ C & 0\end{bmatrix}$ and suppose also that the subspace $\mathscr{A}=\{\begin{bmatrix}A & 0\\ 0 & 0\end{bmatrix}\in\mathscr{X}\}$ is perfect but not of dimension rs. Then $\mathscr{X}$ itself is perfect.*

Proof. Let

$$\mathscr{Y}=\{\begin{bmatrix}0 & B\\ C & 0\end{bmatrix}\} \quad \text{(a subspace of } \mathscr{X} \text{ by hypothesis)},$$
$$\mathscr{D}=\{\begin{bmatrix}0 & 0\\ 0 & D\end{bmatrix}\in\mathscr{X}\},$$
$$\mathscr{D}_1=\{\begin{bmatrix}0 & 0\\ 0 & D\end{bmatrix}:\begin{bmatrix}A & B\\ C & D\end{bmatrix}\in\mathscr{X} \text{ for some } \begin{bmatrix}A & B\\ C & D\end{bmatrix}\}.$$

Then $\mathscr{D}\subseteq\mathscr{D}_1$ and both have dimension at least $(m-r)(n-s)-2$. In fact $\mathscr{D}_1$ is perfect. For if this were not the case $\mathscr{D}_1$ would have dimension $(m-r)(n-s)-2$ and so would be equal to $\mathscr{D}$. But then $\mathscr{D}$ would be the image of $\mathscr{X}$ under the linear mapping $\begin{bmatrix}A & B\\ C & D\end{bmatrix}\to\begin{bmatrix}0 & 0\\ 0 & D\end{bmatrix}$ and $\mathscr{A}\oplus\mathscr{Y}$ would be its kernel from which we would obtain $\mathscr{X}=\mathscr{A}\oplus\mathscr{Y}\oplus\mathscr{D}$; it would follow that $\mathscr{A}$ had dimension rs contradicting the hypotheses.

Consequently we may take a basis of $\mathscr{X}$ consisting of all $E_{ij}\in\mathscr{Y}$, a basis of rank 1 matrices of $\mathscr{A}$ (since $\mathscr{A}$ is perfect), together with certain matrices of the form $\begin{bmatrix}A & 0\\ 0 & D\end{bmatrix}$, where each matrix D has rank 1. These latter matrices $\begin{bmatrix}A & 0\\ 0 & D\end{bmatrix}$ may not have rank 1 and to complete the proof we show that by adding a suitable matrix in $\mathscr{A}\oplus\mathscr{Y}$ to every such matrix we can obtain a new basis of $\mathscr{X}$ consisting entirely of rank 1 matrices.

Consider then any one of these basis matrices $\left[\begin{smallmatrix} A & 0 \\ 0 & D \end{smallmatrix}\right]$. If $\left[\begin{smallmatrix} A & 0 \\ 0 & 0 \end{smallmatrix}\right] \in \mathscr{A}$ we may replace $\left[\begin{smallmatrix} A & 0 \\ 0 & D \end{smallmatrix}\right]$ in the basis by $\left[\begin{smallmatrix} A & 0 \\ 0 & D \end{smallmatrix}\right] - \left[\begin{smallmatrix} A & 0 \\ 0 & 0 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0 & 0 \\ 0 & D \end{smallmatrix}\right]$. If $\left[\begin{smallmatrix} A & 0 \\ 0 & 0 \end{smallmatrix}\right] \notin \mathscr{A}$ then $\langle \mathscr{A}, \left[\begin{smallmatrix} A & 0 \\ 0 & 0 \end{smallmatrix}\right] \rangle$ has dimension rs or $rs-1$ and so is perfect, generated say by the basis of $\mathscr{A}$ and one further rank 1 matrix $\left[\begin{smallmatrix} T & 0 \\ 0 & 0 \end{smallmatrix}\right]$. Then we have $\left[\begin{smallmatrix} A & 0 \\ 0 & 0 \end{smallmatrix}\right] = \theta \left[\begin{smallmatrix} T & 0 \\ 0 & 0 \end{smallmatrix}\right] + \left[\begin{smallmatrix} S & 0 \\ 0 & 0 \end{smallmatrix}\right]$ with $\left[\begin{smallmatrix} S & 0 \\ 0 & 0 \end{smallmatrix}\right] \in \mathscr{A}$, $\theta \neq 0$, and hence $\left[\begin{smallmatrix} A & 0 \\ 0 & D \end{smallmatrix}\right] = \theta \left[\begin{smallmatrix} T & 0 \\ 0 & D \end{smallmatrix}\right] + \left[\begin{smallmatrix} S & 0 \\ 0 & 0 \end{smallmatrix}\right]$. Since θT and D each have rank 1 we may let $\theta T = u_1 v_1$, $D = u_2 v_2$ for suitable column vectors u_1, u_2 and row vectors v_1, v_2. The matrix $M = \left[\begin{smallmatrix} u_1 \\ u_2 \end{smallmatrix}\right] [v_1 v_2]$ has rank 1 and

$$\begin{bmatrix} A & 0 \\ 0 & D \end{bmatrix} = M - \begin{bmatrix} 0 & u_1 v_2 \\ u_2 v_1 & 0 \end{bmatrix} + \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix}.$$

Consequently $\left[\begin{smallmatrix} A & 0 \\ 0 & D \end{smallmatrix}\right]$ can be replaced in the basis by M. Since this can be done for each of the basis matrices $\left[\begin{smallmatrix} A & 0 \\ 0 & D \end{smallmatrix}\right]$ the proof is complete.

Proof of proposition. We begin by replacing $\mathscr{X}$ by an equivalent space chosen so that $\mathscr{X}^*$ is spanned by two matrices which define a pencil in Kronecker canonical form [3] without any infinite elementary divisors. By hypothesis the pencil decomposition is not of the form

$$J_2(\lambda) \oplus 0 = \alpha X + \beta Y = \begin{bmatrix} \alpha + \beta\lambda & \beta & 0 \\ & \alpha + \beta\lambda & \\ 0 & & 0 \end{bmatrix}$$

and therefore we may take it to be

$$\alpha \begin{bmatrix} X_1 & 0 \\ 0 & X_2 \end{bmatrix} + \beta \begin{bmatrix} Y_1 & 0 \\ 0 & Y_2 \end{bmatrix},$$

where the subpencil $\alpha X_1 + \beta Y_1$ is one of the pencils which figure in Lemmas 1,2,3. By definition of $\mathscr{X}^*$ the space $\mathscr{X}$ contains all matrices of the form $\left[\begin{smallmatrix} 0 & B \\ C & 0 \end{smallmatrix}\right]$ and the subspace $\mathscr{A}$ of all $\left[\begin{smallmatrix} A & 0 \\ 0 & 0 \end{smallmatrix}\right]$ in $\mathscr{H}$ has, as its annihilator, the space defined by the pencil $\alpha X_1 + \beta Y_1$. $\mathscr{X}$ therefore satisfies the conditions of Lemma 4 and consequently is perfect.

An immediate consequence of the proposition is that any nondegenerate $m \times n \times (mn-2)$ tensor of rank $mn-1$ is equivalent to the one defined by the $mn-2$ matrices $\{E_{11}+E_{22}\} \cup \{E_{ij} : (i,j) \neq (1,1), (1,2), (2,2)\}$. In fact the assumption of nondegeneracy can be dropped. For suppose $A_1, \cdots, A_{mn-2}$ are linearly dependent matrices defining some degenerate $m \times n \times (mn-2)$ tensor. Then the annihilator of the space $\langle A_1, \cdots, A_{mn-2} \rangle$ is of dimension more than 2, and it is easy to prove that it contains a two-dimensional subspace not equivalent to the one generated by $E_{11}+E_{22}$ and E_{12}. Hence $\langle A_1, \cdots, A_{mn-2} \rangle$ is contained in a perfect space of dimension $mn-2$ and so its tensor cannot have rank more than $mn-2$. In particular, every $m \times n \times (mn-3)$ tensor has rank at most $mn-2$.

Finally we note that the assumption about algebraic closure cannot in general be omitted. For example, with $m = n = 2$, the two-dimensional space of all matrices $\left[\begin{smallmatrix} a & b \\ -b & a \end{smallmatrix}\right]$ contains no real rank 1 matrix so, over the real field, is neither perfect nor equivalent to the one generated by $E_{11}+E_{22}$ and E_{12}.

Our results can be summarized as follows: For $m \times n \times (mn-2)$ tensors which do not reduce to smaller tensors we have completely solved the ranking problem in the case of algebraically closed fields. Hitherto the only other nontrivial class of tensors for which the ranking problem had been solved was the case of $m \times n \times 2$ tensors [3]. Both solutions rely on Kronecker's theory of pencils but apply it in very different ways.

REFERENCES

[1] M. D. ATKINSON AND N. M. STEPHENS, *On the maximal multiplicative complexity of a family of bilinear forms*, Linear Algebra and Appl., 27 (1979), pp. 1–8.

[2] R. W. BROCKETT AND D. DOBKIN, *On the optimal evaluation of a set of bilinear forms*, Linear Algebra and Appl., 19 (1978), pp. 207–235.

[3] J. JA'JA', *Optimal evaluation of pairs of bilinear forms*, this Journal, 8 (1979), pp. 443–462.

[4] J. KRUSKAL, *Three-way arrays: rank and uniqueness of trilinear decompositions, with applications to arithmetic complexity and statistics*, Linear Algebra and Appl., 18 (1977), pp. 95–138.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0002 $01.25/0

OPTIMIZING CONJUNCTIVE QUERIES THAT CONTAIN UNTYPED VARIABLES*

D. S. JOHNSON† AND A. KLUG‡

Abstract. This paper addresses questions of efficiency in relational databases. We present polynomial time algorithms for minimizing and testing equivalence of what we call "fan-out free" queries. The fan-out free queries form a more general and more powerful subclass of the conjunctive queries than those previously studied. In particular, they can be used to express questions about transitive properties of databases, questions that are impossible to express if one operates under the assumption, implicit in previous work, that each variable has an assigned "type," and hence can only refer to one fixed attribute of a relation. Our algorithms are graph-theoretic in nature, and the equivalence algorithm can be viewed as solving a special case of the graph isomorphism problem (by reducing it to a series of labelled forest isomorphism questions).

Key words. relational databases, query optimization

1. Introduction. There has been much recent work addressing the problem of optimizing queries in the relational database model of Codd [5]. Such work has been motivated by the fact that, in the relational data model, it is easy to express queries whose natural implementation would be very inefficient. Thus there could be great value in a method for converting queries into equivalent ones with more efficient implementations. From a complexity-theoretic point of view, the seminal papers in this area are those of Chandra and Merlin [4] and Aho, Sagiv and Ullman [2], [3], which concentrated on the natural class of "conjunctive queries."

For conjunctive queries (which we shall define later), the main factor in the efficiency of the implementation is the number of conjuncts contained in the query. Thus the query optimization problem becomes simply the problem of finding an equivalent query involving the minimum possible number of conjuncts. Moreover, the problem of telling whether two minimal queries are equivalent can be reduced to a simple syntactic test. Unfortunately, as shown in [4], that "simple syntactic test" is equivalent to the presumably intractable problem of graph isomorphism, and the conjunct minimization problem is NP-hard and hence even more likely to be intractable. (For a discussion of NP-hardness and intractability, see [1], [6].)

Chandra and Merlin [4] faced this obstacle by designing exponential time algorithms for the problem, arguing quite reasonably that since queries might well be applied to very large databases, the savings resulting from a more efficient implementation of a query might well justify spending a large amount of time optimizing the query. Queries typically only involve a few conjuncts and so exponential time with respect to query length may still be smaller than, say, linear, quadratic or some higher order polynomial time with respect to database size. Also, the same query may be run many times, thus amortizing the cost of optimization.

Aho, Sagiv and Ullman [2], [3] took the alternative route of looking for special cases that could be solved efficiently, i.e., in polynomial time. As a first step, they restricted attention to *typed* queries, i.e., queries in which each variable has a specific attribute of the database associated with it as its "type," and can only refer to entries

* Received by the editors March 29, 1982, and in final form October 25, 1982. A preliminary version of this paper appeared, under a slightly different title, in the *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Los Angeles, CA, 1981.

† Bell Laboratories, Murray Hill, New Jersey 07974, and University of Wisconsin, Madison, Wisconsin 53706.

‡ University of Wisconsin, Madison, Wisconsin 53706.

in columns of the database labelled by that attribute. This is a significant restriction, since it seriously interferes with our ability to derive transitive information from databases, for instance, to ask for (child, grandparent) pairs given a relation containing only (child, parent) pairs. Given this basic restriction and an additional technical restriction to only those queries representable by "simple tableaux," they derived a class of queries for which minimization and equivalence testing can both be performed in polynomial time: minimization in time $O(n^3)$ [3] and equivalence testing in time $O(n^4)$ [2]. Subsequent work by Sagiv [7] has reduced both running times to $O(n^2)$. Sagiv and Yannakakis [8] have also introduced two new classes of typed queries for which the problems can be solved in polynomial time, again $O(n^2)$ for minimization [7] and $O(n^2)$ for equivalence [8].

In this paper we show that polynomial time algorithms exist even when the restriction to typed queries is dropped, although new techniques are required for these more general algorithms. Our running times are, coincidentally, the same as those in the original Aho, Sagiv and Ullman papers [2], [3].

In § 2 we review the basic definitions and illustrate the limitations of typed queries, as well as the other restrictions of [2], [3], [7], [8]. In § 3 we define a class of conjunctive queries, the *fan-out free* queries, which allow *un*typed variables and can be viewed as a generalization of the other classes, even in the typed case. The "fan-out free" property is specified in terms of an *implication graph* that is defined for any conjunctive query. In § 4 we show how the minimization problem for fan-out free queries can be transformed into a series of connectivity tests on this implication graph and thus solved in time $O(n^3)$.

The algorithm of § 4 is in a sense just a generalization and extension of the ideas behind the earlier techniques of [3], [7] to a much wider class (about the widest possible such class that has a simple structural characterization). The techniques used for equivalence testing in [2], [8] do not, however, extend in any nice way once untyped variables are allowed. We obtain an algorithm by showing that the problem, although equivalent to graph isomorphism if we consider general (minimized) conjunctive queries, can be reduced to the well-solved problem of labeled tree isomorphism in the case of fan-out free queries. The algorithm runs in time $O(n^4)$ and is presented in § 5. Most of the work in the section is involved in showing that an appropriately labeled breadth-first spanning forest for the implication graph of such a query can be viewed as a canonical form for the query.

The paper concludes in § 6 with an examination of possible extensions and improvements on our algorithms, along with other directions for future research.

2. A few definitions. A *relation* R can be viewed as a finite two-dimensional table with columns labeled by distinct *attributes*. Each attribute A_i has a *domain* $D(A_i)$, and entries in a column labeled by A_i must be elements of the domain $D(A_i)$. The *relation scheme* for a relation R is the sequence of attributes labelling its columns and may be viewed as an ordered subset of the set of all attributes. Since the order of the *rows* in a relation has no significance, we can view a relation with scheme $(A_1, A_2, \cdots, A_k)$ as a finite subset of $D(A_1)\times D(A_2)\times\cdots\times D(A_k)$. A *database* is a finite set of tables. The *relation schema* for a database is the multiset of relation schemes for the tables it contains. In what follows we shall assume for simplicity of notation that all databases under consideration have the same relation schema $\{S_1, \cdots, S_s\}$, with scheme S_i corresponding to relation R_i in the database.

We shall also assume that there are no data dependencies. "Functional dependencies" can be handled just as in [2], [3], by means of a preprocessing step prior to the

application of our algorithms. More complicated dependencies, such as multivalued dependencies or the even more general "algebraic" dependencies of [9], give rise to much added complexity, and we follow [2], [3] in leaving these complexities for future work.

A *query* can be viewed as a function from databases to relations. A query Q has *target scheme* S_Q if it maps databases to relations with relation scheme S_Q. A *conjunctive query* Q can be specified by $S_Q = (A_1, \cdots, A_p)$ and the following: (1) A set $X_Q = \{x_1, \cdots, x_p\}$ of *distinguished variables* (DV's), (2) a set $Y_Q = \{y_1, \cdots y_q\}$ of *nondistinguished variables* (NDV's) and (3) a set $C_Q = \{c_1, \cdots, c_r\}$ of distinct *conjuncts*, each conjunct c_i being associated with a relation $R(c_i)$ of the underlying relation schema and having the form $(c_i[1], \cdots c_i[m])$, where $m = \text{Length}(c_i)$ is the number of columns (attributes) in $R(c_i)$ and each $c_i[j]$ is either a DV, an NDV or a *constant*, i.e., an element of the domain of the jth attribute of $R(c_i)$.

Given a database B and a conjunctive query Q, the relation constructed when Q is applied to B is

$$Q(B) = \{(x_1, \cdots, x_p): x_i \in D(A_i) \text{ and there exist } y_1, \cdots, y_q \text{ such that for all } c_i \in C_Q, (c_i[1], \cdots c_i[m]), \text{ with the valuations of the DV's and NDV's substituted in, is a row in relation } R(c_i) \text{ of } B\}.$$

For simplicity, we shall often specify conjunctive queries implicitly, using a format similar to the above, using "$R(x, y)$" to stand for the statement that (x, y) is a row in relation R. For example:

$$Q = \{(x_1, x_2): \text{there exist } y_1, y_2 \text{ such that } R_1(x_1, y_1, x_2) \text{ and } R_2(x_1, y_2, 3)\}.$$

The identities of the DV's, NDV's constants and conjuncts are all easily derived from this shorthand.

Chandra and Merlin show in [4] how a query Q can be computed using the operations of "selection," "restriction" and "generalized join," applied to the database which is the query's argument. The major factor in determining the running time for the resulting algorithm is the number of generalized joins performed, and this number is one less than the number of conjuncts in C_Q. Thus the query minimization problem for conjunctive queries is simply the problem of finding a query equivalent to Q which has the least possible number of conjuncts.

Until now, the classes of queries for which efficient minimization algorithms have been derived have all only allowed *typed* variables. A DV or NDV u is typed if it is associated with a unique attribute type (u), i.e., if $c_i[h] = u$, then the hth attribute of relation $R(c_i)$ is type (u).

If a query contains only typed variables, then the underlying relation schema might as well be assumed to have disjoint attribute domains, i.e., $D(A_i) \cap D(A_j) = \varnothing$ for all pairs $A_i \neq A_j$ of attributes in the schema. Even though in practice many databases have overlapping domains, typed queries cannot make use of this information (except insofar as constants in the query can be associated with more than one attribute). Consider the relation R_1 illustrated in Fig. 1, with its first column labelled by the attribute "parent," its second labelled by the attribute "child," and $R_1(x, y)$ if and only if x is the parent of y. The domain for each attribute is the set of people, and, of course, a person can be both a parent and a child. This suggests the possibility of deriving from R_1 the grandparent relation R_G, where R_G has grandparent and child columns and $(x, y) \in R_G$ if x is y's grandparent. This can be done quite simply using

Parent	Child
George III	Adolphus
Victoria	Alice
Albert	Alice
Adolphus	Mary
George V	George VI
Mary	George VI
Elizabeth	Elizabeth II
George VI	Elizabeth II
Victoria	Edward VII
Edward VII	George V

FIG. 1. *Relation R_1 with scheme ⟨Parent,Child⟩. If $R_1(x, y)$, then x is the parent of y.*

untyped variables:

$$\{(x_1, x_2)\text{: there is a } y_1 \text{ such that } R_1(x_1, y_1) \text{ and } R_1(y_1, x_2)\}.$$

However, no typed query will suffice, because such a query cannot use the fact that a parent can be a child.

One cannot get around this problem by simply giving both columns of the relation the same name, say "person," because our basic definitions say that all columns of a relation must be named by distinct attributes, and this fact is crucial to the algorithms of [2], [3], [7], [8]. A second approach, that of using auxiliary tables, *does* allow us to construct the grandparent relation with a typed query, but the results are awkward and inefficient: In the current example what we need is a second relation R_2, which captures the "equivalence" of parents and children in a typed fashion. R_2 has a parent column and a child column and $(x, y) \in R_2$ if and only if x and y refer to the same person. See Fig. 2. We then can specify the grandparent relation R_G with the following typed query:

$$\{(x_1, x_2)\text{: there are } y_1 \text{ and } y_2 \text{ such that } R_1(x_1, y_1), R_2(y_2, y_1) \text{ and } R_1(y_2, x_2)\}.$$

Note that the use of this subterfuge not only makes the query harder to understand, but also introduces a new conjunct and hence a new "join" to the query. We thus can circumvent the restriction to typed queries, but only at a cost of lowered query efficiency.

Another problem is that the three subclasses of typed queries for which algorithms have been derived each impose an additional constraint on the queries they contain and these constraints interfere with the use of equivalence tables. The first two of these restrictions involve the concept of a *repeated variable*, which is defined to be a NDV that occurs more than once in the conjuncts of Q.

(1) If an attribute has a repeated variable associated with it, then there is no other repeated symbol (variable or constant) associated with it.

Parent	Child
George III	George III
Victoria	Victoria
Albert	Albert
Adolphus	Adolphus
George V	George V
Mary	Mary
Elizabeth	Elizabeth
George VI	George VI
Edward VII	Edward VII
Elizabeth II	Elizabeth II

FIG. 2. *Relation R_2 with scheme $\langle Parent,Child\rangle$. If $R_2(x, y)$, then x is the same person as y.*

(2) No conjunct contains more than one occurrence of a repeated variable.

The first of these restrictions gives rise to the class of queries with "simple tableaux," as defined and discussed in [2], [3]. The second gives rise to the first of the two classes introduced in [7], [8].

The reader may verify that (2) makes it impossible to construct the grandparent relation using a typed query, even if equivalence tables are allowed (and rules out the "great-grandparent" relation R_{GG} even if untyped variables are allowed). Restriction (1) allows grandparents, but rules out great-grandparents using either equivalence tables or untyped variables. The third restriction, corresponding to the second class from [7], [8] and using the notion of "covering" which will be defined shortly, also prevents the construction of great-grandparents:

(3) No conjunct of Q is "covered" by more than one conjunct in C_Q other than itself.

Although the class of "fan-out free" conjunctive queries for which our algorithms are designed also obeys a restriction (a weaker one), this new class contains the queries required for constructing *any* fixed-height ancestor relation, and this should be indicative of its increased power over the previous classes.

A final objection to the restriction to typed queries is more theoretical in nature: the natural symmetry between conjunctive queries and relational algebra is lost. As shown by Chandra and Merlin [4], the set of relations constructible by conjunctive queries is precisely the set of relations that can be constructed using the operations of select, project and generalized join. No analogous characterization has been found for typed conjunctive queries, unless one allows an unnatural extension of the concept of "relation" [9].

Thus it is with no great sense of loss that we abandon the restriction to typed queries. Since we are dropping the assumption, there is no real restriction in assuming for simplicity that all attributes have the same domain D, and so we shall follow [4]

in doing so. The next section presents the series of definitions which leads to the definition of "fan-out free."

3. Implication graphs and fan-out free queries. Suppose two queries Q and Q' have the same target scheme and hence the same distinguished variables. Let U_Q be the union of the DV's, NDV's and constants occurring in Q and define $U_{Q'}$ similarly. A *symbol mapping* from Q to Q' is a function $f: U_Q \to U_{Q'}$ which leaves all DV's and constants fixed. If c and c' are conjuncts in Q and Q' respectively, we say that c' *covers* c if $R(c) = R(c')$ and there is a symbol mapping from Q to Q' that sends c to c'. A *conjunct mapping* from Q to Q' is a function $g: C_Q \to C_{Q'}$ such that for all $c \in C_Q$, $g(c)$ covers c.

A fundamental concept, both for query minimization and equivalence testing, is that of a *homomorphism*. A *homomorphism* from Q to Q' is a symbol mapping $f: U_Q \to U_{Q'}$ that induces a well-defined conjunct mapping. Alternatively, we may view a homomorphism as a conjunct mapping $g: C_Q \to C_{Q'}$ that induces a well-defined symbol mapping. In what follows we shall often view a homomorphism as *both* of the above, i.e., a function $h: Q \to Q'$ defined on both U_Q and C_Q, whose symbol mapping half induces its conjunct mapping half and vice versa.

The importance of homomorphisms comes from the following result.

THEOREM 3.1 [4]. *Two queries Q and Q' are equivalent (as functions defined on databases) if and only if there are homomorphisms $h: Q \to Q'$ and $h': Q' \to Q$.*

Thus in both minimization and equivalence testing, our main task will be the search for homomorphisms. In fact, it will turn out that most of our work is involved with finding *self-homomorphisms*, i.e., homomorphisms from a query Q to itself. In order to model potential self-homomorphisms, we introduce the notion of an *implication graph*. Given any conjunctive query Q, the *implication graph* of Q is a bipartite graph $G[Q] = (V[Q], E[Q])$ defined as follows:

The vertices correspond to the potential "elements" of a homomorphism, when that homomorphism is considered as a set of ordered pairs. Since a homomorphism can be viewed as either a conjunct mapping or a symbol mapping, the vertices are of two types. First, there is a set $V_C[Q]$ of *conjunct-pair* vertices, containing a vertex $\langle c, c' \rangle$ for each pair of (not necessarily distinct) conjuncts in C_Q such that c' covers c, and a special vertex $\langle \theta \rangle$ (θ is a special symbol indicating that no homomorphism is possible). Second, there is a set $V_S[Q]$ of *symbol-pair* vertices, containing a vertex $\langle y, z \rangle$ for each NDV $y \in Y_Q$ and each $z \in U_Q$, such that for some conjunct-pair vertex $\langle c_1, c_2 \rangle$ and index j, $1 \leqq j \leqq$ length (c_1), $c_1[j] = y$ and $c_2[j] = z$. (In the future we shall often omit the "$[Q]$" from $V_C[Q]$ and $V_S[Q]$, so long as there is no chance of confusion.) Each edge in $E[Q]$ joins a symbol-pair vertex to a conjunct-pair vertex. There is an edge between $\langle y, z \rangle$ and $\langle \theta \rangle$ if and only if there is a conjunct c and an index j, $1 \leqq j \leqq$ length (c), such that $c[j] = y$ and for all c' such that c' covers c, $c'[j] \neq z$. There is an edge between $\langle y, z \rangle$ and $\langle c, c' \rangle$ if and only if there is a j, $1 \leqq j \leqq$ length (c), such that $c[j] = y$ and $c'[j] = z$. Fig. 3 illustrates the implication graph for a conjunctive query that constructs an interestingly incestuous relation from the parent-child relation R.

Using the implication graph $G[Q]$, we can test whether a symbol mapping (or a conjunct mapping) of Q to itself corresponds to a self-homomorphism. A subset V' of the symbol-pair vertices of $G[Q]$ is said to satisfy the *symbol mapping property* (*partial symbol mapping property*) if each $y \in Y_Q$ occurs as the first component of exactly one (at most one) of its vertices. It is easy to see that any symbol mapping capable of yielding a self-homomorphism must correspond to a set V' satisfying the

BASE RELATION SCHEME $R = \langle \text{PARENT, CHILD} \rangle$

$$X_Q = \{x_1, x_2\}, Y_Q = \{y_1, y_2\}$$

$$C_Q = \{R(y_1, y_2), R(y_1, x_1), R(y_2, x_1), R(y_2, x_2)\}$$

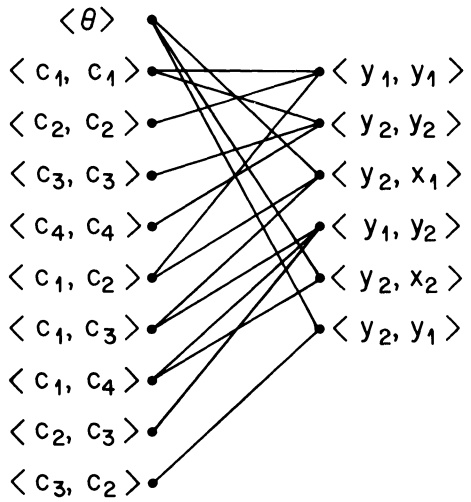


FIG. 3

symbol mapping property: the subset corresponding to the symbol mapping f is simply $V_{S,f} = \{\langle y, z\rangle : y \in Y_Q \text{ and } f(y) = z\}$.

The *conjunct mapping property* (*partial conjunct mapping property*) is defined analogously for subsets of the conjunct-pair vertices of $G[Q]$, none of which is $\langle\theta\rangle$. The subset corresponding to the conjunct mapping g is simply $V_{C,g} = \{\langle c, c'\rangle : c \in C_Q \text{ and } g(c) = c'\}$. A self-homomorphism h, viewed as a joint symbol and conjunct mapping, will then correspond to the set $V_{S,h} \cup V_{C,h}$ of symbol-pair and conjunct-pair vertices, where each half "induces" the other and obeys the appropriate mapping property.

A set V' of symbol-pair vertices induces a set V'' of conjunct-pair vertices if it contains all the neighbors of each vertex in V''. We capture the notion of the set of symbol-pair vertices induced by a set V' of conjunct-pair vertices by using the concept of *implication* in $G[Q]$.

The first part of the definition of implication concerns the conjunct-pair vertices: A conjunct-pair vertex *implies* all of its neighbors.

The following observation is immediate.

LEMMA 3.1. *A conjunct mapping g of Q to itself is a self-homomorphism of Q if and only if the set of symbol-pair vertices implied by* $V_{C,g}$ *obeys the symbol mapping property.*

We extend the definition of *implication* to symbol-pair vertices as follows: symbol-pair vertices also imply neighbors, but are more selective. A symbol-pair vertex of the form $\langle y, y\rangle$ implies nothing. A symbol-pair vertex which is adjacent to $\langle\theta\rangle$ implies

this vertex and nothing else. Otherwise, a symbol-pair vertex of the form $\langle y, z \rangle$, $y \neq z$, implies a neighboring conjunct-pair vertex $\langle c, c' \rangle$ if and only if this is the only conjunct-pair vertex having c as first component to which $\langle y, z \rangle$ is adjacent.

The extension of the definition of *implication* to symbol-pair vertices is necessary because we wish to use the implication graph to determine whether certain *partial* conjunct mappings can be extended to self-homomorphisms. To this end we introduce the notion of *implication closure*. If V' is a subset of the vertices of $G[Q]$, then the implication closure $\bar{V}'$ of V' is the smallest superset of V' that is closed under implication, i.e., that contains all the vertices implied by its vertices. The implication closure of $V_{C,g}$, where g is a partial conjunct mapping of Q to itself, is designed to include all the forced consequences of g.

Unfortunately, in the general case for conjunctive queries, these "forced consequences" do not provide enough information to tell us whether g can be extended to a self-homomorphism. There is a "fan-out" of possibilities, and one would have to examine them all in order to verify extendibility. Thus in what follows we impose a restriction on queries that guarantees that such fan-out cannot occur. We say that a symbol-pair vertex $\langle y, z \rangle$ is *fan-out free* if either (a) all its neighbors have the *same* first component or (b) all its neighbors have *distinct* first components. In the latter case $\langle y, z \rangle$ implies all of its neighbors, in the former it implies none of them (except in the special case where it has only one neighbor). A conjunctive query Q is *fan-out free* if and only if every symbol-pair vertex $\langle y, z \rangle$, $y \neq z$, in V_S is fan-out free.

To see that this restriction prevents the fan-out of possibilities, consider the consequences of adding a conjunct-pair $\langle c, c' \rangle$ to a partial conjunct mapping for a fan-out free query. If $\langle c, c' \rangle$ implies $\langle y, z \rangle$ and $y \neq z$, then either (a) $\langle y, z \rangle$ is adjacent to $\langle \theta \rangle$ and hence no extension to a full self-homomorphism is possible, (b) $\langle y, z \rangle$ is adjacent only to other conjunct-pairs with c as first component, none of which can be part of the same homomorphism as $\langle c, c' \rangle$, or (c) *all* the other conjunct-pairs adjacent to $\langle y, z \rangle$ are implied by $\langle y, z \rangle$ and must be present in any self-homomorphism containing $\langle c, c' \rangle$. If a symbol-pair vertex $\langle y, z \rangle$ with fan-out were allowed, it might have neighbors $\langle c_1, c_2 \rangle$ and $\langle c_1, c_3 \rangle$, $c_2 \neq c_3$ and we would have a *choice* as to which one to add in attempting to extend our partial conjunct mapping.

Note that we do allow fan-out at vertices of the form $\langle y, y \rangle$, but under our definitions these vertices do not imply any of their neighbors and the following theorem will show that they do not need to. Let us say that a subset V' of the conjunct-pair vertices of an implication graph $G[Q]$ is *transitively closed* if the implication closure of V' contains no new conjunct-pair vertex, i.e., if $\bar{V}' \cap V_C = V'$. A transitively closed set V' of conjunct-pair vertices obeys the *partial homomorphism property* if it satisfies the following two conditions:

(C1) V' does not contain $\langle \theta \rangle$.

(C2) The set V'' of symbol-pair vertices in $\bar{V}'$ obeys the partial symbol mapping property.

Observe that (C2) implies that $\bar{V}'$ itself obeys the partial conjunct mapping property, since if V' contained two distinct conjunct-pairs with the same first component, then V'' would have to contain two distinct symbol-pairs with the same first component. Theorem 3.2 validates our claim that, for fan-out free queries, we can tell whether a partial conjunct mapping extends to a full self-homomorphism merely by computing its implication closure:

THEOREM 3.2. *Let Q be a fan-out free query and suppose V' is a transitively closed subset of the conjunct-pair vertices of $G[Q]$ that obeys the partial homomorphism property. Then the set V^* of vertices obtained by adding to V' all symbol-pair vertices*

$\langle y, y\rangle$, where y does not occur as a first component in $\bar{V}'$, and all conjunct-pair vertices $\langle c, c\rangle$, where c does not appear as a first component in $\bar{V}'$, corresponds to a self-homomorphism of Q.

Proof. Let f and g be the symbol mapping and conjunct mapping corresponding to $V_{S^*} = V^* \cap V_S$ and $V_{C^*} = V^* \cap V_C$ respectively. By (C2) f obeys the symbol mapping property and, by (C1) and our observation following the definition of the partial homomorphism property, g obeys the conjunct mapping property. Thus by Lemma 3.1 all we need show is that V_{C^*} implies V_{S^*}. This can fail to happen only if (i) V_{C^*} implies some $\langle y, z\rangle$ not in V_{S^*} or (ii) V_{C^*} fails to imply some $\langle y, z\rangle$ which *is* in V_{S^*}. We show that both possibilities lead to contradictions.

(i) Suppose V_{C^*} implies a symbol-pair $\langle y, z\rangle$ which is not in V_{S^*}. Then $\langle y, z\rangle$ must be implied by some conjunct-pair $\langle c, c'\rangle$ that is in V^* but not in V' and hence has $c' = c$. This means that $z = y$. The fact that $\langle y, y\rangle$ is not in V_{S^*} means that there is some $\langle y, z'\rangle$, $z' \neq y$, which is in V_{S^*}. The fact that c contains an occurrence of y means that there is some conjunct-pair vertex $\langle c, c''\rangle$, $c'' \neq c$, adjacent to $\langle y, z'\rangle$ in G_Q. The fact that $\langle c, c\rangle$ is in V^* means that $\langle y, z'\rangle$ does not imply $\langle c, c''\rangle$. Since $z' \neq y$, the fact that Q is fan-out free thus allows us to conclude that all vertices adjacent to $\langle y, z'\rangle$ in G_Q have c as first component. In particular, the conjunct-pair in V' that implies $\langle y, z'\rangle$ has c as first component, in contradiction to the fact that $\langle c, c\rangle$ is in V^*.

(ii) Suppose V_{C^*} fails to imply some symbol-pair $\langle y, z\rangle$ in V_{S^*}. Then we must have that $\langle y, z\rangle$ is not in V' and hence $z = y$. By our definition of the implication closure, there must be some conjunct c that contains an occurrence of y. We cannot have $\langle c, c\rangle$ in V_{C^*} or else V_{C^*} would imply $\langle y, y\rangle$. Hence there is some $\langle c, c'\rangle$ in V_{C^*}, $c' \neq c$. The vertex $\langle c, c'\rangle$ cannot be adjacent to $\langle y, y\rangle$ in G_Q (or else it would imply it), but it must be adjacent to *some* symbol-pair $\langle y, z'\rangle$ with y as first component. But this means that $\langle y, z'\rangle$ is in V_{S^*} and $z' \neq y$, and hence we cannot have added $\langle y, y\rangle$ to $\bar{V}'$, again a contradiction.

Thus both cases are impossible and the theorem is proved. □

We thus see that our restriction to fan-out free queries does what we want it to. The fact that it is not too severe a restriction can be gleaned from the next two theorems.

THEOREM 3.3. *Any typed query Q satisfying restriction* (1) *or* (3) *of* § 2 *is fan-out free, and there exist fan-out free typed queries that obey neither* (1) *nor* (3).

Proof. In order for Q to fail to be fan-out free, there must be a symbol-pair vertex $\langle y, z\rangle$ in $G[Q]$ such that $y \neq z$ and $\langle y, z\rangle$ is adjacent to three conjunct-pair vertices $\langle c_1, c_2\rangle$, $\langle c_3, c_4\rangle$ and $\langle c_3, c_5\rangle$, where $c_1 \neq c_3$ and $c_4 \neq c_5$. Since $y \neq z$, this would mean that c_3 was covered by two conjuncts other than itself and so Q would violate (3). It would also mean that y was a repeated variable (occurring in both c_1 and c_3) and z was a second repeated symbol (occurring in c_4 and c_5) associated with the same attribute, thus violating (1). Hence any query satisfying (1) or (3) must be fan-out free.

An example of a typed query which is fan-out free but violates both (1) and (3) is the query which constructs the great-grandparent relation using the parent-child relation R_1 and the equivalence relation R_2 from § 2:

$$\{(x_1, x_2): \text{ there exist } y_1, y_2, y_3 \text{ and } y_4 \text{ such that } R_1(x_1, y_1), R_2(y_2, y_1), R_1(y_2, y_3), R_2(y_4, y_3) \text{ and } R_1(y_4, x_2)\}.$$

The third conjunct of this query is covered by both the first and the fifth, thus violating (3), and the "parent" attribute has two repeated variables (y_1 and y_3), thus violating (1). To see that the query is fan-out free, note that the only conjunct-pair vertices $\langle c, c'\rangle$ in $G[Q]$ with $c \neq c'$ are $\langle c_2, c_4\rangle$, $\langle c_4, c_2\rangle$, $\langle c_3, c_1\rangle$ and $\langle c_3, c_5\rangle$ and that no $\langle y, z\rangle$,

$y \neq z$, is adjacent to more than two of them or to *any* conjunct-pair of the form $\langle c, c \rangle$ (at least three neighbors are required for a symbol-pair vertex to have "fan-out"). □

Thus, the class of fan-out free queries is more general than either of the classes defined by the restrictions (1) and (3), even when restricted (as *they* are) to typed variables. The relationship between fan-out free queries and those obeying restriction (2) of § 2 is less straightforward. Although (2), by forbidding more than one repeated variable in any conjunct, rules out any sophisticated use of transitivity, it does allow some queries which are not fan-out free. A simple example would be

$$\{(x_1)\text{: there exist } y_1, y_2, y_3 \text{ and } y_4 \text{ such that } R_1(y_1, x_1), R_1(y_1, y_2), R_1(y_3, x_1) \text{ and } R_1(y_3, y_4)\}.$$

The query obeys (2) since the only repeated (nondistinguished) variables are y_1 and y_3. It is not fan-out free since $\langle y_3, y_1 \rangle$ is adjacent to $\langle c_3, c_1 \rangle$, $\langle c_4, c_1 \rangle$ and $\langle c_4, c_2 \rangle$.

However, note that the above query is equivalent to the fan-out free query containing the single conjunct $\langle y_1, x_1 \rangle$. In fact, it is not difficult to prove the following theorem (using Lemma 4.2 of the next section).

THEOREM 3.4. *Any minimal query satisfying restriction* (2) *of* § 2 *is fan-out free.*

Thus any question that can be phrased using a conjunctive query obeying restrictions (1), (2) or (3) can be phrased using a fan-out free query, and fan-out free queries can express many questions that would be impossible to ask if any one of those restrictions must be obeyed.

4. Minimizing fan-out free queries. Our minimization algorithm, like those of [3], [7], is based on the following result from [4]:

LEMMA 4.1. *If a conjunctive query Q is equivalent to a conjunctive query Q' that has fewer conjuncts, then there is a self-homomorphism of Q that sends the conjuncts of Q to a proper subset of themselves* (a *shrinking self-homomorphism* of Q).

Proof. By Theorem 3.1, if Q is equivalent to Q' then there exist homomorphisms $h: Q \to Q'$ and $h': Q' \to Q$. The desired self-homomorphism is h composed with h'. □

LEMMA 4.2. *If h is a self-homomorphism of Q then the set of conjuncts in the image of h represents a query Q' which is equivalent to Q.*

Proof. By Theorem 3.1 we need to show that there are homomorphisms in both directions. We already have the homomorphism h which goes from Q to Q'. The homomorphism in the other direction is simply the identity self-homomorphism on Q, restricted to the conjuncts and NDV's in Q'. □

Our minimization algorithm will be based on a subroutine S which, given Q, determines whether it has a shrinking self-homomorphism, and, if so, returns one. We use **S** in the following loop.

1. Set $Q^* = Q$.
2. While **S** applied to Q^* yields a shrinking self-homomorphism h, set $Q^* = h(Q^*)$. (Note that if Q^* is fan-out free, then so must be $h(Q^*)$.)
3. Return Q^* (if **S** applied to Q^* does not yield a shrinking self-homomorphism, then Q^* is our desired minimum equivalent query).

Note that if Q has N conjuncts, subroutine **S** will be invoked at most N times, since each successful application reduces the number of conjuncts, and we always must have at least one conjunct left.

We thus have reduced the problem of conjunctive query minimization to that of finding shrinking self-homomorphisms. In the case of fan-out free queries, we can use the results of the last section in doing the "finding";

LEMMA 4.3. *Suppose Q is a fan-out free query. Then the following two statements are equivalent.*

(a) *Q has a shrinking self-homomorphism.*

(b) *There is a conjunct-pair vertex $\langle c, c' \rangle$ in $G[Q]$ with $c \neq c'$ such that the set V' of conjunct-pair vertices in the implication closure of $\{\langle c, c' \rangle\}$ satisfies the partial homomorphism property and the set $C_1(V')$ of first components of conjunct-pairs in V' does not equal the set $C_2(V')$ of second components.*

Proof. That (b) implies (a) follows immediately from Theorem 3.3. By that theorem, V' extends to a self-homomorphism h of Q which is the identity on all conjuncts not in $C_1(V')$. Since there is a conjunct in $C_1(V') - C_2(V')$, that conjunct will not be in the image of C_Q under h, and so h is a shrinking self-homomorphism.

To prove the implication in the other direction, let us suppose that h is a shrinking self-homomorphism of Q, and let $V' = V_{C,h}$ be the associated set of conjunct-pair vertices in $G[Q]$. Consider the implication closure V', which is simply $V' \cup V_{S,h}$ since h is a self-homomorphism. By the definition of implication closure, $\bar{V}'$ is the union of the implication closure of $\{\langle c, c' \rangle\}$, for all $\langle c, c' \rangle$ in V'. Let $\bar{V}[c, c']$ be the set of conjunct-pairs in the implication closure of $\langle c, c' \rangle$. If $C_1(\bar{V}[c, c']) = C_2(\bar{V}[c, c'])$ for all $\langle c, c' \rangle$ in V', then we must have $C_1(V') = C_2(V')$ and so h would not be a shrinking self-homomorphism. Thus there is some $\langle c, c' \rangle$ such that $\bar{V}[c, c']$ has a set of first components *not* equalling its set of second components. We claim that this is the desired conjunct-pair. Clearly $c \neq c'$, since the implication closure of $\langle c, c \rangle$ is simply that vertex by itself, and $C_1(\bar{V}[c, c]) = C_2(\bar{V}[c, c]) = \{c\}$. The set $\bar{V}[c, c']$ does not contain $\langle \theta \rangle$ since $\bar{V}'$ did not, so it satisfies (C1) of the partial homomorphism property. Finally, $\bar{V}[c, c']$ satisfies (C2) since it is contained in $\bar{V}'$. Thus $\bar{V}[c, c']$, being transitively closed, satisfies the partial homomorphism property, and $\langle c, c' \rangle$ is the desired conjunct-pair. □

Lemma 4.3 is enough to give us a polynomial time algorithm for minimizing fan-out free queries: We can implement subroutine **S** by (1) constructing the implication graph for Q, (2) finding the implication closure of each of its conjunct-pair vertices $\langle c, c' \rangle$, $c \neq c'$, (3) testing each implication closure to see whether its conjunct-pairs satisfy property (b) of the lemma and finally (4) using Theorem 3.2 to generate the corresponding shrinking self-homomorphism when it exists.

As schematized above, our algorithm has a major potential inefficiency. If there are N conjuncts (and hence potentially $N(N-1)/2$ conjunct-pairs $\langle c, c' \rangle$, $c \neq c'$), the time to find all the implication closures might be no better than $O(N^2)$ times the size of $G[Q]$. Such a bound could be attained if many of the implication closures were large and overlapped, causing us to examine the same part of the graph over and over again. In order to avoid this, we make use of one more lemma about implication graphs of fan-out free queries.

LEMMA 4.4. *If Q is a fan-out free query and $\langle c_1, c_2 \rangle$, $\langle y, z \rangle$ and $\langle c_3, c_4 \rangle$ are vertices of $G[Q]$ such that $\langle c_1, c_2 \rangle$ implies $\langle y, z \rangle$ and $\langle y, z \rangle$ implies $\langle c_3, c_4 \rangle$, then $\langle c_3, c_4 \rangle$ implies $\langle y, z \rangle$ and $\langle y, z \rangle$ implies $\langle c_1, c_2 \rangle$.*

Proof. For the assumed implications to hold, we must have that $\langle y, z \rangle$ is adjacent to both $\langle c_1, c_2 \rangle$ and $\langle c_3, c_4 \rangle$. Thus $\langle c_3, c_4 \rangle$ implies $\langle y, z \rangle$, since a conjunct-pair implies all symbol-pairs to which it is adjacent. The argument for the second implication goes as follows. Since $\langle y, z \rangle$ implies $\langle c_3, c_4 \rangle$, we know that $\langle y, z \rangle$ is not adjacent to $\langle \theta \rangle$, $y \neq z$ and $c_3 \neq c_1$. Since Q is fan-out free, this means that *all* conjunct-pairs adjacent to $\langle y, z \rangle$ have distinct first components and thus $\langle y, z \rangle$ implies $\langle c_1, c_2 \rangle$. □

As a consequence of this lemma, we note that if $\langle c_1, c_2 \rangle$ is in the implication closure of $\{\langle c_3, c_4 \rangle\}$, then $\langle c_3, c_4 \rangle$ is in the implication closure of $\{\langle c_1, c_2 \rangle\}$, and, in fact,

the two implication closures are equal. Thus, in the process of generating the implication closures of individual $\langle c, c' \rangle$'s, we never need generate the implication closure of a conjunct-pair $\langle c, c' \rangle$ which has already appeared in the implication closure of some earlier conjunct-pair. Hence we only need look at each conjunct-pair vertex once in the entire process. A symbol-pair vertex *can* appear in more than one distinct implication closure, if it is one of those symbol-pairs that, for some reason, does not imply all its neighbors. However, it will only be encountered a number of times bounded by the number of its neighbors, since each neighbor is itself in at most one implication closure. Thus it is easy to see that the total amount of work required to generate the relevant implication closures under this modified scheme is proportional to the size of the implication graph, not N^2 times that size.

For those uninterested in writing special programs to generate implication closures, we note that we can modify the implication graph so that the relevant implication closures can be found using a standard algorithm for generating the connected components of a graph. The modification is straightforward: For each symbol-pair vertex $\langle y, z \rangle$ where either $y = z$ or all the neighbors of $\langle y, z \rangle$ have the same first component, split the vertex $\langle y, z \rangle$ into a number of copies of itself equal to the number of conjunct-pair vertices adjacent to it, and let each of these copies be adjacent to a corresponding one of those conjunct-pairs.

We leave to the reader the straightforward task of verifying that the connected components of the modified graph correspond to the implication closures of the sets $\{\langle c, c' \rangle\}$ in $G[Q]$, with the union of all implication closures that contain $\langle \theta \rangle$ corresponding to the single connected component containing $\langle \theta \rangle$. Since the connected components of a graph can be generated in time which is linear in the size of the graph [1] and since the modified graph can be constructed in time proportional to the size of $G[Q]$, this graph-connectivity approach will be about equivalent to the one already proposed.

Having found the relevant implication closures, we must then examine each to see whether its conjunct-pair vertices satisfy the partial homomorphism property and have a set of second components different from their set of first components. This can be done for each $V' = \bar{V}[c, c']$ in time proportional to the size of V', and hence the overall time for this part of the algorithm is also bounded by the size of $G[Q]$. We test V' as follows. Condition (C1) of the partial homomorphism property, that V' does not contain $\langle \theta \rangle$, can easily be checked in the claimed time. Condition (C2), that no two symbol-pair vertices of V' share the same first component, can be verified in time proportional to the number of such vertices by indexing into an array with entries for each NDV (we avoid initializing the table by using the trick described in [1, Exercise 2.12]). A similar trick is used to verify that $C_1(V') \neq C_2(V')$.

To complete the analysis of the running time of our minimization algorithm, we must consider how large the implication graph $G[Q]$ can be, and how much time is required to generate it. As argued above, $G[Q]$ contains at most $O(N^2)$ conjunct-pair vertices besides $\langle \theta \rangle$, with at most N having any given conjunct c as first component. The number of edges involving $\langle \theta \rangle$ is bounded by the number of symbol-pair vertices. Each conjunct-pair vertex $\langle c, c' \rangle$ is adjacent to at most length (c) symbol-pair vertices, one for each column of c. Thus the total number of symbol-pair vertices is at most N times the sum of the lengths of the conjuncts in C_Q, where this sum can be taken as proportional to n, the size of the query itself, and the total number of edges in $E[Q]$ is at most twice this much. Since $N \leqq n$, we thus obtain $O(n^2)$ as a bound on the size of $G[Q]$. The time to generate $G[Q]$ also obeys this bound: We can tell whether c' covers c in time proportional to length (c), and if so, generate all the edges incident on $\langle c, c' \rangle$ at the same time, merely by comparing $c[i]$ and $c'[i]$, $1 \leqq i \leqq$ length (c)

(using our indexing trick to make sure that no earlier j had $c[j]=c[i]$ and $c'[j]\neq c'[i]$). Generating the edges involving $\langle\theta\rangle$ requires similar indexing tricks.

Thus the time for our subroutine **S** is $O(n^2)$ and hence the time for the overall minimization algorithm is $O(n^3)$. Note however that for queries that are already near-minimal, as we would expect them to be in practice, the time bound may reduce to $O(n^2)$, since the number of times **S** is applied is at most one more than the number of redundant conjuncts in Q.

In cases where Q is far from minimal, we might hope to speed up the algorithm by a more sophisticated generation by **S** of the shrinking self-homomorphism h it returns. Clearly our goal should be to find a shrinking self-homomorphism which has the smallest image. One thing we could do is generate h from the $V'=\bar{V}[c,c']$ that has the largest value of $|C_1(V')-C_2(V')|$. This could be done without substantially increasing the overhead in **S**.

A more expensive approach would use the fact that some *set* of implication closures must correspond to a maximal shrinking self-homomorphism, i.e., one which sends C_Q to a minimum equivalent subset of itself. Thus we might attempt to combine implication closures and see whether the combined implication closure (the union of two implication closures is itself an implication closure) satisfies the conditions of Lemma 4.3 and has a larger value of $|C_1(V')-C_2(V')|$ than either of its progenitors. The complexity involved in doing these tests can be reduced somewhat by using the fact that there is a maximal shrinking self-homomorphism which is the identity self-homomorphism when restricted to its image (this fact can be proved by considering high-order compositions of a maximal shrinking self-homomorphism h^* with itself). However, we have been unable to discover any way to find good combinations that does not end up costing just as much as performing the algorithm in the standard way. It is not difficult to show that a greedy approach to building combinations can leave one arbitrarily far from the maximal shrinking self-homomorphism.

Before closing this section, we note that the approach to minimization described in this section can be extended to a wider class of queries than just those that are fan-out free. Given the implication graph of an arbitrary query, one can, by doing a search of the graph starting at $\langle\theta\rangle$, identify all those vertices that imply $\langle\theta\rangle$ (directly or indirectly). We then delete all these vertices and examine the remaining connected components to see if any of these will generate a shrinking self-homomorphism. If so, apply the shrinking self-homomorphism and repeat the procedure. If not, and all the remaining symbol-pair vertices are fan-out free, then we know that our query is minimal. If not, and there is a symbol-pair vertex with fan-out, we are stuck.

A simple characterization of the queries for which this minimization procedure succeeds has eluded us; the fan-out free queries seem to be the least restrictive class for which membership can be determined by "local" properties and yet minimization is guaranteed.

5. Equivalence testing for fan-out free queries. The first step of our algorithm for testing whether two fan-out free queries Q and Q' are equivalent is to perform the minimization algorithm of the previous section on both of them. This reduces the equivalence problem to one of isomorphism, as can be seen from the following easily-proved lemma:

LEMMA 5.1. *Suppose Q and Q' are minimal queries. Then Q is equivalent to Q' if and only if there is a homomorphism from Q to Q' which is one-one and onto for both* NDV*'s and conjuncts. Note that by setting $Q=Q'$ in Lemma* 5.1 *we obtain the fact that any self-homomorphism of a minimal query is an automorphism.*

For the rest of this section we shall assume that both Q and Q' are minimal fan-out free queries. In light of Lemma 5.1 we may also assume that they have the same number of NDV's and conjuncts. The basic outline of our equivalence algorithm is as follows:

For each conjunct $c \in C_Q$ we construct a forest of labelled trees forest $[c] =$ {tree $[c, c']$: $c' \in C_Q$, c' covers c and $c' \neq c$}, where tree $[c, c']$ is a labelled breadth-first spanning tree of the implication closure of $\{\langle c, c'\rangle\}$ in the implication graph $G[Q]$. Similar forests are constructed for each $c \in C_{Q'}$. We then make use of the following two lemmas (to be proved later). For brevity we use the term *tree isomorphism* to stand for *labelled forest isomorphism* and use the symbol $\cong$ to denote both tree and query isomorphism.

LEMMA 5.2. *If $f: Q \to Q'$ is a query isomorphism, then* forest $[c] \cong$ forest $[f(c)]$ *for every conjunct $c \in C_Q$.*

LEMMA 5.3. *Suppose $c_1, c_2 \in C_Q$ and* forest $[c_1] \cong$ [forest $[c_2]$. *Then there is a query automorphism $f: Q \to Q$ such that $f(c_1) = c_2$.*

These two lemmas yield the following key theorem.

THEOREM 5.1. *Suppose $c \in C_Q$, $c' \in C_{Q'}$ and* forest $[c] \cong$ forest $[c']$. *Then there is a query isomorphism $f: Q \to Q'$ if and only if there is a query isomorphism $f': Q \to Q'$ with $f(c) = c'$.*

Proof. Suppose there is a query isomorphism $f: Q \to Q'$. If $f(c) = c'$ we are done, so suppose $f(c) = c''$. Note that this implies that c and c'' cover each other. Since c and c' cover each other by hypothesis, it thus must be the case that c' and c'' cover each other. By Lemma 5.2, $f(c) = c''$ implies that there is a tree isomorphism t: forest $[c] \to$ forest $[c'']$. Since by assumption forest $[c] \cong$ forest $[c']$, we conclude that there is a tree isomorphism t': forest $[c'] \to$ forest $[c'']$. By Lemma 5.3 we thus conclude that there is a query automorphism $f': Q' \to Q'$ with $f'(c'') = c'$. The composition of f followed by f' yields our desired query isomorphism. □

Our equivalence algorithm can thus proceed as follows:

1. Set $Q^* = Q$, $Q^{**} = Q'$ and $f = \varnothing$ (the empty symbol mapping).
2. While Q^* contains a conjunct c that contains a NDV, do the following:
   If there is no conjunct $c' \in C_{Q^{**}}$ such that forest $[c] \cong$ forest $[c']$, then halt: Q and Q' are not equivalent by Lemma 5.2. Otherwise, let c^* be such a c' and extend the definition of f to include $f(c[j]) = c^*[j]$ for every j, $1 \leqq j \leqq$ length (c), such that $c[j]$ is a NDV.
   If f fails to be either well-defined or one-one, halt: the queries are inequivalent. Otherwise, for each NDV u added to the domain of f, replace u and $f(u)$ in both Q^* and Q^{**} by a new unique constant symbol u^*.
3. At this point, Q^* no longer contains any NDV's. If the sorted lists of conjuncts for Q^* and Q^{**} are identical, then the constructed f will correspond to a query isomorphism; otherwise, no such isomorphism exists (Theorem 5.1 guarantees that if there is a query isomorphism, the lists will be identical.)

We now present the details of the construction of tree $[c, c']$ and the proofs of Lemmas 5.2 and 5.3. We assume without loss of generality that c and c' are conjuncts of query Q. Our construction requires a bit more information than is present in the implication graph as defined in § 3. Therefore, in what follows, we replace the "honorary" conjunct-pair vertex $\langle\theta\rangle$ by a collection of vertices of the form $\langle c, \theta\rangle$, one for each conjunct $c \in C_Q$. There is an edge between a symbol-pair $\langle y, z\rangle$, $y \neq z$ and the conjunct-pair $\langle c, \theta\rangle$ if there is a j, $1 \leqq j \leqq$ length (c), such that $c[j] = y$ and for no c' that covers c does $c'[j] = z$. Note that if all the $\langle c, \theta\rangle$ vertices are coalesced into a

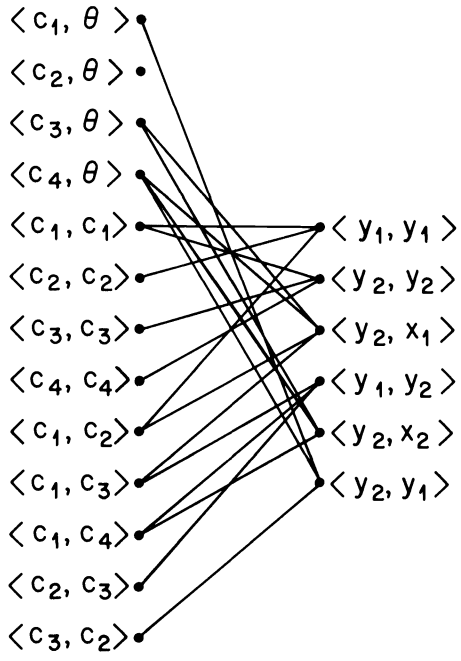


FIG. 4

single vertex, this vertex will be adjacent to precisely the same symbol-pair vertices as was $\langle\theta\rangle$. See Fig. 4. For the remainder of this section, the term *implication graph* and the abbreviation $G[Q]$ will always refer to an expanded graph of this type.

The tree $T = \text{tree}[c, c']$ need not actually be a labelled breadth-first spanning tree for the entire implication closure of $\{\langle c, c'\rangle\}$ in $G[Q]$, only for part of it. The basic idea is to start with the conjunct-pair vertex $\langle c, c'\rangle$ as root and expand the tree until either (a) the implication closure is completely spanned or (b) there is evidence that the set of conjunct-pair vertices in the implication closure of $\{\langle c, c'\rangle\}$ does not induce a query automorphism.

There will be two labels associated with each vertex ν of T, a *hidden label*, $H[\nu]$ and a *visible label*, label$[\nu]$. The hidden labels are used in our construction of T and in our proofs, but are not considered when testing for tree isomorphisms. The hidden label of a vertex ν is simply its name in $G[Q]$, i.e., $H[\langle c, c'\rangle] = \langle c, c'\rangle$ and $H[\langle y, z\rangle] = \langle y, z\rangle$. We shall use $H_1[\nu]$ and $H_2[\nu]$ to denote the first and second components of the hidden label of ν, respectively.

The visible labels are the ones used in testing for tree isomorphism. The visible label label$[\nu]$ for a symbol-pair vertex ν is of the form $\langle i, A\rangle$, where i, a column index, is the *local address* and A is the *mapping conflict indicator*. The visible label for a conjunct-pair vertex ν is of the form $\langle[R, j], P, A\rangle$. Here $[R, j]$, with R being a relation name and j being a column index, is the local address. The second component, P, is the *profile* and the third, A, is again the mapping conflict indicator. The visible labels are assigned so that every vertex ν has a unique global address $a(\nu)$, this being the sequence of visible labels for all the vertices on the path from the root of T up to and including ν.

The local address for a child u of ν is assigned based on the reason for the adjacency of $H[u]$ and $H[\nu]$ in $G[Q]$. If $H[u] = \langle y, z\rangle$ and $H[\nu] = \langle c_1, c_2\rangle$, then the

local address for u as a child of ν is the least column index i such that $c_1[i]=y$ and $c_2[i]=z$ (such an index must exist by the definition of implication graph). If $H[u]=\langle c_1, c_2\rangle$ and $H[\nu]=\langle y, z\rangle$, then the local address $[R, j]$ for u has $R=R(c_1)=R(c_2)$ and j the least index such that $c_1[j]=y$ and $c_2[j]=z$. If u is the root of tree $[c, c']$ and hence has no parent, its local address is $[R(c), 0]$ by convention.

Note that although no conjunct-pair vertex can have two children with the same local address in tree $[c, c']$, a symbol-pair vertex may have a number of conjunct-pair children with the same local address. One purpose of the profile is to distinguish between conjunct-pair children with the same local address. The profile is present in label $[\nu]$ for all conjunct-pair vertices ν and is determined entirely by $H_1[\nu]$. If $H_1[\nu]=c_1$ then the profile of ν, which we shall denote by profile $[c_1]$, is a list of the form $\langle[J_1, b_1], [J_2, b_2], \cdots, [J_k, b_k]\rangle$, where the J_i's are sets forming a partition of the integers in $\{h: 1 \leqq h \leqq \text{length } (c_1)\}$ and the b_i's are either DV's, constants or the special symbol $\varnothing$. Each set J_i is a maximal set of indices h such that $c_1[h]$ has the same value for all $h \in J_i$. The entry b_i is this common value if that value is either a DV or a constant. If, however, the common value is a NDV, then b_i is by convention set to $\varnothing$ (our visible labels must be independent of the names of the NDV's). Thus we know that each J_i with $b_i=\varnothing$ represents a different NDV, but not which one. Within *profile* $[c_1]$ the sets are ordered according to their minimal elements and each set has its members presented in increasing order.

If a vertex of tree $[c, c']$ has a mapping conflict indicator other than 0, we say that that vertex has a *mapping conflict*. This will mean that $\{\langle c, c'\rangle\}$ cannot be extended to a query automorphism. There will be at most one vertex in T with a mapping conflict (as soon as such a vertex is encountered in our tree generation process, we halt and declare T complete). If a symbol-pair vertex ν has a mapping conflict, its mapping conflict indicator A is the global address of an earlier symbol-pair vertex u such that $H_1[u]=H_1[\nu]$ and $H_2(u) \neq H_2[\nu]$ (the fact that T contains such vertices means that $\{\langle c, c'\rangle\}$ cannot extend to a well-defined query automorphism).

If a conjunct-pair vertex ν has a mapping conflict, there are three possibilities: the mapping conflict indicator for ν can either be "1", "2" or the global address of some earlier conjunct-pair vertex. If the mapping conflict indicator for ν is equal to 1, this means that $H[\nu]=\langle c'', \theta\rangle$ for some conjunct c'', in which case $\{\langle c, c'\rangle\}$ clearly cannot be extended to a query automorphism. If the mapping conflict indicator for ν is equal to 2, this means that there are at least two conjunct-pair vertices that could be children of parent(ν) with the same visible label. The presence of two such vertices clearly destroys the uniqueness of global addresses, but as we shall see, it also means that $\{\langle c, c'\rangle\}$ cannot be extended to a one-one self-homomorphism (and hence not to a query automorphism, since Q is minimal). If the mapping conflict indicator for ν is the global address for an earlier conjunct-pair vertex u, this means that $H_1[u]=H_1[\nu]$ and $H_2[u] \neq H_2[\nu]$ and so again $\{\langle c, c'\rangle\}$ cannot be extended.

This will be explained more fully as we now present an explicit procedure for constructing tree $[c, c']$. We proceed by levels. Odd levels contain conjunct-pair vertices and even levels contain symbol-pair vertices. The first level consists solely of the root, a single conjunct-pair vertex ν with $H[\nu]=\langle c, c'\rangle$ and label $[\nu]=\langle[R(c), 0]$, profile $[c], 0\rangle$. If all the vertices on level I, I odd, have been generated and the construction of tree $[c, c']$ has not yet been terminated, the generation of the vertices of level $I+1$ proceeds according to procedure EVENLEVEL. (In the following a vertex ν of tree $[c, c']$ is *unprocessed* if we have not as yet attempted to generate its children; a vertex $\langle c_1, c_2\rangle$ (or $\langle y, z\rangle$) in $G[Q]$ is *unspanned* if there is as yet no vertex in tree $[c, c']$ with $\langle c_1, c_2\rangle$ (or $\langle y, z\rangle$) as its hidden label.)

Procedure EVENLEVEL
begin
Lexicographically order the vertices of level I according to their global addresses. While there remains an unprocessed vertex on level I, let ν be the lexicographically first such vertex and do the following:
begin
Let $H[\nu] = \langle c_1, c_2 \rangle$ and for each unspanned $\langle y, z \rangle$ that is adjacent to $\langle c_1, c_2 \rangle$ in $G[Q]$, ordered by their potential local address (the least i such that $c_1[i] = y$ and $c_2[i] = z$), do the following:
begin
Create a new symbol-pair vertex u of tree $[c, c']$, with ν as parent, with i as local address and with $H[u] = \langle y, z \rangle$. If there is any already-generated vertex w with $H_1[w] = y$ (there can be at most one), then let A be the global address of w, set label $[u] = \langle i, A \rangle$ and halt the generation of tree $[c, c']$ (the tree is complete since we have encountered a mapping conflict—see Fig. 5). Otherwise, set label $[u] = \langle i, 0 \rangle$ and continue.
end
end
end EVENLEVEL

If all the vertices on level I, I even, have been generated and the construction of tree $[c, c']$ has not yet been terminated, then we generate the vertices of level $I+1$ using the procedure ODDLEVEL:

procedure ODDLEVEL
begin
Lexicographically order the vertices of level I according to their global addresses, deleting all those ν such that $H[\nu]$ either (a) has the same first and second components or (b) equals $\langle y, z \rangle$, where all the vertices adjacent to $\langle y, z \rangle$ in $G[Q]$ have the same first component (in other words, deleting all those ν which correspond to symbol-pair vertices in $G[Q]$ that do not imply anything).
While there remains an unprocessed vertex on level I, let ν be the lexicographically first such vertex, and do the following:
begin
Let $H[\nu] = \langle y, z \rangle$. If $\langle y, z \rangle$ is adjacent in $G[Q]$ to a vertex with θ as second component, do the following:
begin
Let $D = \{c_1 \in C_Q$: there is a vertex $\langle c_1, \theta \rangle$ adjacent to $\langle y, z \rangle$ in $G[Q]\}$. For each $c_1 \in D$, let $A[c_1]$, the potential local address of c_1, be $[R, j]$, where $R = R(c_1)$ and j is the least index such that $c_1[j] = y$ and for no c_2 with $c_2[j] = z$ does c_2 cover c_1 (such a j must exist by the definition of implication graph). Then create a new vertex u as a child of ν with $H[u] = \langle c^*, \theta \rangle$ and label $[u] = \langle A[c^*]$, profile $[c^*]$, $1\rangle$, where c^* is chosen to be a lexicographically minimum member of D with respect to the $(A[c^*]$, profile $[c^*])$ pair (see Fig. 6). Since a mapping conflict has been found, halt and terminate the construction of tree $[c, c']$.
end
Otherwise, all neighbors of $\langle y, z \rangle$ have distinct first components. For each potential local address $[R, j]$ of an unspanned neighbor of $\langle y, z \rangle$ in $G[Q]$, in lexicographic order, do the following:

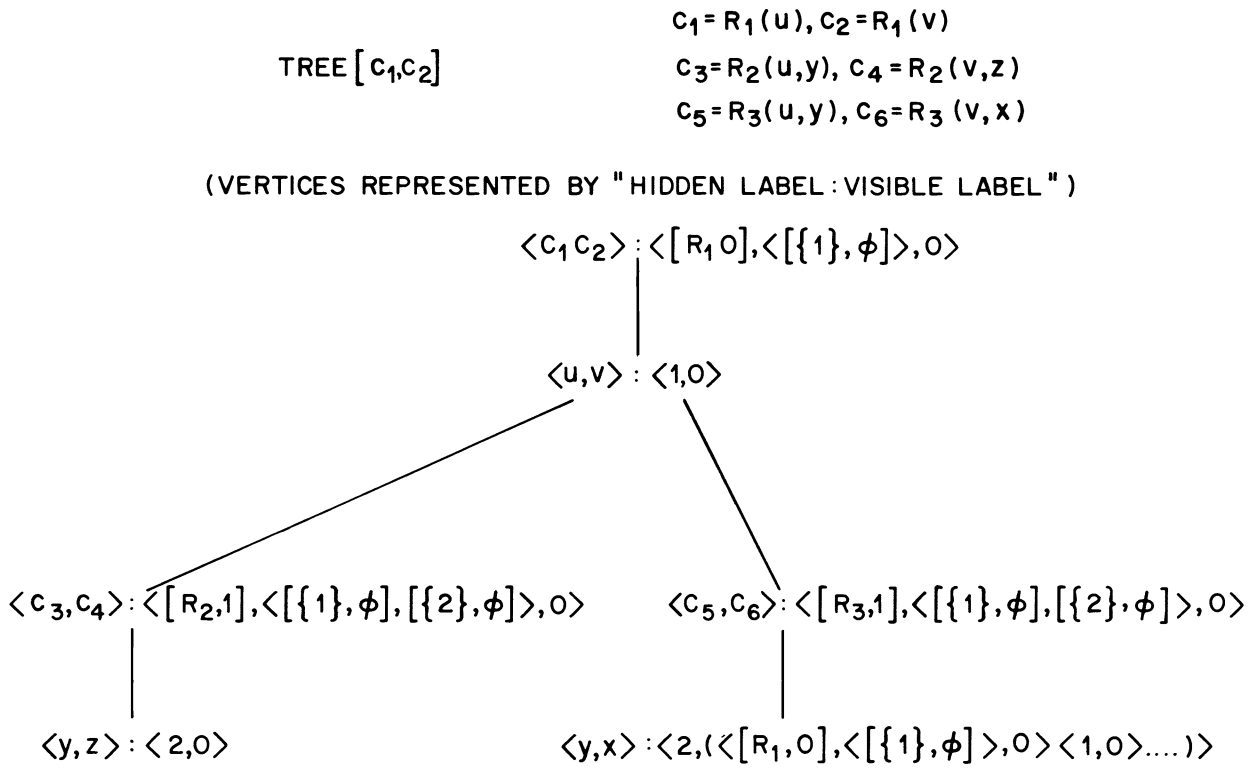

$c_1 = R_1(u), c_2 = R_1(v)$
$c_3 = R_2(u,y), c_4 = R_2(v,z)$
$c_5 = R_3(u,y), c_6 = R_3(v,x)$
TREE $[c_1, c_2]$
(VERTICES REPRESENTED BY "HIDDEN LABEL : VISIBLE LABEL")
$\langle c_1 c_2 \rangle : \langle [R_1 0], \langle [\{1\}, \phi] \rangle, 0 \rangle$
$\langle u, v \rangle : \langle 1, 0 \rangle$
$\langle c_3, c_4 \rangle : \langle [R_2, 1], \langle [\{1\}, \phi], [\{2\}, \phi] \rangle, 0 \rangle$
$\langle c_5, c_6 \rangle : \langle [R_3, 1], \langle [\{1\}, \phi], [\{2\}, \phi] \rangle, 0 \rangle$
$\langle y, z \rangle : \langle 2, 0 \rangle$
$\langle y, x \rangle : \langle 2, (\langle [R_1, 0], \langle [\{1\}, \phi] \rangle, 0 \rangle \langle 1, 0 \rangle \ldots) \rangle$

FIG. 5

begin
Let $D = \{\langle c_1, c_2\rangle$: $\langle c_1, c_2\rangle$ is an unspanned conjunct-pair vertex adjacent to $\langle y, z\rangle$ in $G[Q]$ with $R(c_1) = R$, $c_1[j] = y$ and $c_2[j] = z\}$. For each $\langle c_1, c_2\rangle \in D$, in lexicographic order by values of profile $[c_1]$ do the following:
begin
Create a new vertex u as a child of v with $H[u] = \langle c_1, c_2\rangle$ and construct its visible label as follows:
begin
If there is a $\langle c_3, c_4\rangle \in D$, $\langle c_3, c_4\rangle \neq \langle c_1, c_2\rangle$ with profile $[c_3]$ = profile $[c_1]$, then the fact that $\langle c_1, c_2\rangle$ preceded $\langle c_3, c_4\rangle$ in our processing of D is only due to whatever arbitrary tie-breaking procedure we used to order conjunct-pairs in D whose first components had the same profile. We could just as well have had $\langle c_3, c_4\rangle$ first and $H[u]$ would then have been assigned the value $\langle c_3, c_4\rangle$. This is a mapping conflict of type 2 (see Fig. 7). Set label $[u] = \langle [R, j]$, profile $[c_1]$, $2\rangle$ and terminate the construction of tree $[c, c']$.
If there is an already-generated vertex w of tree $[c, c']$ with $H_1[w] = c_1$ (there can be at most one), then let A be the global address of w, set label $[u] = \langle [R, j]$, profile $[c_1]$, $A\rangle$ and halt the generation of tree $[c, c']$. Otherwise, set label $[u] = \langle [R, j]$, profile $[c_1]$, $0)\rangle$ and continue.
end
end
end
end
end ODDLEVEL

$$c_1 = R_1(u),\ c_2 = R_1(v),\ c_3 = R_2(u, 5),\ c_4 = R_2(v, 10)$$

TREE $[c_1, c_2]$

$\langle c_1, c_2\rangle : \langle [R_1, 0], \langle [\{1\}, \phi]\rangle, 0\rangle$

$\langle u, v\rangle : \langle 1, 0\rangle$

$\langle c_3, \theta\rangle : \langle [R_2, 1], \langle [\{1\}, \phi], [\{2\}, 5]\rangle, 1\rangle$

FIG. 6

The construction of tree $[c, c']$ continues until terminated as above or until an entire level is processed without creating any new vertices.

It should be clear that this construction yields a forest $[c]$ whose topology and labelling is totally independent of the names of the NDV's in Q and of the order of the conjuncts in C_Q. This information is not used explicitly in our labellings, and whenever ties are broken lexicographically, the sorting is done in terms of relation names, column indices, DV's, constant names, profiles, and special symbols such as θ and $\varnothing$, none of which depend on the NDV's or conjunct ordering, and all of which would be present in any query equivalent to Q. Thus the proof of Lemma 5.2, that

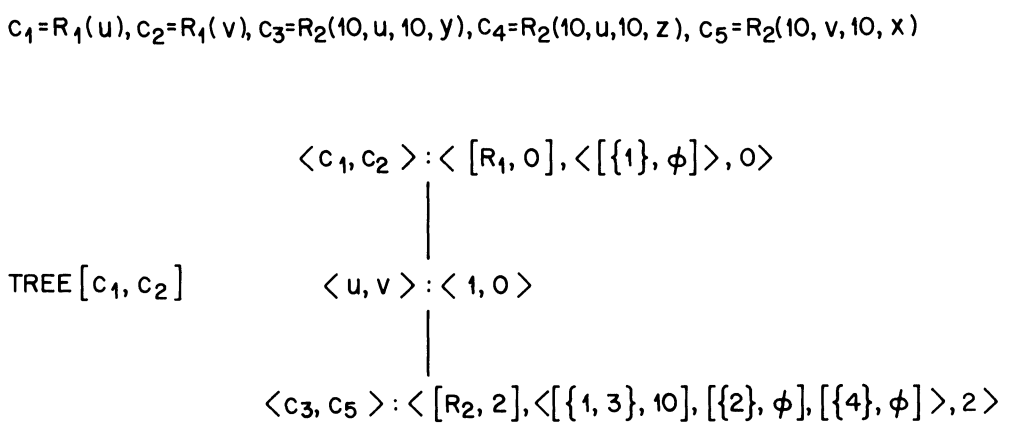


FIG. 7

if $Q \cong Q'$ then the corresponding conjuncts have isomorphic forests, would only involve a straightforward (and tedious) induction, which we shall omit.

The proof of Lemma 5.3 is much more interesting and will proceed via a sequence of sublemmas.

LEMMA 5.3.1. *If* profile $[c_1]$ = profile $[c_2]$, *then* c_1 *and* c_2 *cover each other and any conjunct that covers one must cover the other.*

Proof. The fact that c_1 and c_2 have the same profile means that they are the identical up to a renaming of their NDV's. Hence there is a symbol mapping f that sends c_1 to c_2 and a symbol mapping f' that sends c_2 to c_1 and so each covers the other. The second half of the lemma follows from the fact that the "covering" relation is transitive. □

LEMMA 5.3.2. *If there is a conjunct-pair vertex* u *in* tree $[c, c']$ *with a mapping conflict indicator of* 2, *then the implication closure of* $\{\langle c, c' \rangle\}$ *in* $G[Q]$ *contains two conjunct-pair vertices* $\langle c_1, c_2 \rangle$ *and* $\langle c_3, c_2 \rangle$, $c_1 \neq c_3$ *and* $c_2 \neq \theta$, *either of which could have been assigned as the value of* $H[u]$ *by procedure* ODDLEVEL.

Proof. Let $H[u] = \langle c_1, c_2 \rangle$. By the operation of ODDLEVEL in assigning a mapping conflict indicator of 2, u was the child of some symbol-pair vertex $\langle y, z \rangle$, all of whose neighbors had distinct first components, and there was a second conjunct-pair $\langle c_3, c_4 \rangle$ that could also have been assigned as $H[u]$ with $c_2 \neq \theta \neq c_4$. We now show $c_2 = c_4$.

Suppose $c_2 \neq c_4$. Since c_1 and c_3 have the same profiles and c_4 covers c_3, we would have by Lemma 5.3.1 that c_4 also covers c_1. Thus both c_2 and c_4 would cover c_1 and hence $\langle c_1, c_2 \rangle$ and $\langle c_1, c_4 \rangle$ would be distinct neighbors of $\langle y, z \rangle$ in $G[Q]$. However, $\langle y, z \rangle$ is adjacent to only one conjunct-pair with c_1 as first component and so $c_2 = c_4$ and $\langle c_1, c_2 \rangle$ and $\langle c_3, c_2 \rangle$ are the desired conjunct-pairs. □

LEMMA 5.3.3. *For all conjunct-pairs* $\langle c, c' \rangle$ *of* Q *such that* c' *covers* c *and* $c \neq c'$, *if* tree $[c, c']$ *contains no vertex with a mapping conflict, then there is a query automorphism* $f: Q \to Q$ *with* $f(c) = c'$.

Proof. If tree $[c, c']$ contains no vertex with a mapping conflict, then it must span the entire implication closure of $\{\langle c, c' \rangle\}$. Let V' be the set of conjunct-pair vertices in this closure. We claim that V' obeys the partial homomorphism property. Condition (C1) is satisfied since if V' had contained a vertex with θ as second component, this would have given rise to a mapping conflict of type 1 for a conjunct-pair vertex in

tree $[c, c']$. Condition (C2) is satisfied since if the set of symbol-pair vertices in $\bar{V}'$ did not obey the partial symbol mapping property, it would contain vertices $\langle y, z\rangle$ and $\langle y, z'\rangle$, $z \neq z'$ and the second of these to be encountered in the generation of tree $[c, c']$ would have had a mapping conflict (unless the generation of tree $[c, c']$ had already been terminated on account of some earlier mapping conflict).

Since V' obeys the partial homomorphism property and $\langle c, c'\rangle \in V'$, there is a self-homomorphism of Q that sends c to c' by Theorem 3.3 Since Q is minimal, this self-homorphism is the desired query automorphism. □

LEMMA 5.3.4. *If there is a tree isomorphism* t: forest $[c] \rightarrow$ forest $[c']$ *and* $c \neq c'$, *then for each vertex* v *in tree* $[c, c']$, $H_2[v] = H_1[t(v)]$.

Proof. Let tree $[c', c'']$ be the image under t of tree $[c, c']$. We proceed by induction on the levels of the two trees. In each the first level consists of a single root vertex. If v is the root vertex in tree $[c, c']$, then $t(v)$ must be the root vertex in tree $[c', c'']$. Hence we have $H[v] = \langle c, c'\rangle$ and $H[t(v)] = \langle c', c''\rangle$, and the lemma holds for the first level.

Suppose it holds for parent (u) and t(parent (u)). We show it holds for u. First note that t(parent (u)) = parent $(t(u))$, since t is an isomorphism. We now split into cases, depending on whether u is a symbol-pair or a conjunct-pair vertex.

Suppose u is a symbol-pair vertex. By hypothesis there exist c_1, c_2 and c_3 such that $H[\text{parent } (u)] = \langle c_1, c_2\rangle$ and $H[\text{parent } (t(u))] = \langle c_2, c_3\rangle$. Suppose $H[u] = \langle y, z\rangle$ and $H[t(u)] = \langle y', z'\rangle$. Both u and $t(u)$ must have the same local address, say i. This means that $c_1[i] = y$ and $c_2[i] = z$ and also that $c_2[i] = y'$ and $c_3[i] = z'$. But then $z = y'$ and so $H_2[u] = H_1[t(u)]$, as desired.

Suppose u is a conjunct-pair vertex. This is a more complicated case. By hypothesis there exist y, z and z' such that $H[\text{parent } (u)] = \langle y, z\rangle$ and $H[\text{parent } (t(u))] = \langle z, z'\rangle$. Suppose $H[u] = \langle c_1, c_2\rangle$ and $H[t(u)] = \langle c_3, c_4\rangle$. Since t preserves visible labels, we must have profile $[c_1]$ = profile $[c_3]$. Thus c_3 covers c_1 by Lemma 5.3.1. Moreover, $\langle c_1, c_3\rangle$ must be adjacent to $\langle y, z\rangle$ in $G[Q]$: the fact that t preserves visible labels means that u and $t(u)$ share the same local address $[R, j]$, which implies that $c_1[j] = H_1[\text{parent } (u)] = y$ and $c_3[j] = H_1[\text{parent } (t(u))] = z$. Thus both $\langle c_1, c_2\rangle$ and $\langle c_1, c_3\rangle$ are adjacent to $\langle y, z\rangle$. Since procedure ODDLEVEL gave children to vertex $\langle y, z\rangle$ in tree $[c, c']$, it must be the case that $y \neq z$ and that all conjunct-pair vertices adjacent to $\langle y, z\rangle$ in $G[Q]$ have distinct first components. Hence we must have $c_2 = c_3$, as claimed. □

Proof of Lemma 5.3. Suppose that t: forest $[c] \rightarrow$ forest $[c']$ is a tree isomorphism. We must show that there is a query automorphism $f: Q \rightarrow Q$ such that $f(c) = c'$. By Lemma 5.3.3, this reduces to showing that tree $[c, c']$ contains no vertex with a mapping conflict, since c and c' must have the same profile if their forests are isomorphic and hence must cover each other by Lemma 5.3.1. We shall show that each of the four possible mapping conflicts is impossible.

(1) Suppose there is a symbol-pair vertex u in tree $[c, c']$ with a nonzero mapping conflict indicator A, and let v be the vertex with global address A in tree $[c, c']$. We must have $H[u] = \langle y, z\rangle$ and $H[v] = \langle y, z'\rangle$ for some $z' \neq z$. Now consider $t(u)$ and $t(v)$. By Lemma 5.3.4, we must have $H_1[t(u)] = H_2[u] = z$ and $H_1[t(v)] = H_2[v] = z'$. But then $H_1[t(u)] \neq H_1[t(v)]$, and so there cannot be a mapping conflict between $t(u)$ and $t(v)$, even though the fact that t preserves labels means that there must be. Hence we have a contradiction.

(2) Suppose there is a conjunct-pair vertex u with a mapping conflict of type 1. Then $H_2[u] = \theta$. But this means, by Lemma 5.3.4, that $H_1[t(u)] = \theta$, an obvious contradiction since θ can never occur as a first component.

(3) Suppose that there is a conjunct-pair vertex u with a mapping conflict of type 2. This means, in our construction procedure ODDLEVEL, $H[u]$ was chosen by an arbitrary tie-breaking rule from two conjunct-pairs with the same local addresses and profiles, and by Lemma 5.3.2 these pairs are of the form $\langle c_1, c_2\rangle$ and $\langle c_3, c_2\rangle$, $c_1 \neq c_3$. Because t preserves labels, $H[t(u)]$ must also have been chosen from two conjunct-pairs $\langle c_1', c_2'\rangle$ and $\langle c_3', c_2'\rangle$, $c_1' \neq c_3'$. But by Lemma 5.3.4, applied to the two alternative hidden labels for $t(u)$ in tree $[c, c']$, we must also have both $c_1' = c_2$ and $c_3' = c_2$, so again we have a contradiction.

(4) The case where a conjunct-pair vertex u has a mapping conflict indicator which is the global address of some earlier conjunct-pair vertex is handled in the same way as case (1), leading to a similar contradiction.

Thus all possibilities for mapping conflicts are impossible and the desired query automorphism exists by Lemma 5.3.3. □

Since Lemma 5.3, together with Lemma 5.2, implies Theorem 5.1, we now know that our equivalence algorithm works. The remainder of this section is devoted to questions of its efficiency.

We first observe that labelled forest isomorphism can be tested in time proportional to the *size* of the forests, i.e., the number of vertices in the two forests plus the total length of the labels [1]. Since our basic algorithm performs at most $O(N^2)$ labelled forest isomorphism tests, where N is the number of conjuncts in Q or Q', we thus can bound the overall running time for the labelled forest ismorphism tests by placing a bound on the size of forest $[c]$ for any $c \in C_Q$. This size is in turn bounded by N times the maximum size of any tree $[c, c']$, where c' covers c, and so it is this latter quantity that we shall examine in detail.

In what follows, let us identify each vertex of tree $[c, c']$ with the value of its hidden label. In addition, we make the standard assumption that our computer word size is sufficient to be able to hold any relation name R, distinguished variable name x, constant z or index i. Thus the only visible labels that require more than a constant amount of space are the conjunct-pair labels (because of their profiles) and the (at most) one label that has a global address for its mapping conflict indicator. For the purpose of asymptotic analysis we can ignore the latter case, since a global address is merely a concatenation of other labels, and hence can at most double the total space required for labels.

We first examine the symbol-pair vertices of tree $[c, c']$. All but possibly the last symbol-pair vertex of tree $[c, c']$ have distinct NDV's as first components, and since the number of NDV's in Q is $O(n)$, where n is the sum of the lengths of the conjuncts in Q, this means that there are $O(n)$ symbol-pair vertices in tree $[c, c']$. From this and the fact that each symbol-pair label (except possibly the last) requires only constant space, we conclude that the size of tree $[c, c']$ attributable to symbol-pair vertices is $O(n)$.

The situation is only slightly more complicated for the conjunct-pair vertices. Discounting the last such vertex (should it have a mapping conflict), all conjunct-pair vertices have distinct first components. Moreover, the only nonconstant-sized part of the visible label for a conjunct-pair with c_1 as first component is profile $[c_1]$, whose length is proportional to length (c_1). Thus the total number of conjunct-pair vertices is $O(N)$ and the total length of their labels is at most $O(n)$.

We conclude that the size of tree $[c, c']$ is at most $O(n)$, Hence the overall time for the labeled forest isomorphism tests is $O(N^3 n) = O(n^4)$ time. This will also be a bound on the running time for the entire algorithm, so long as each tree $[c, c']$ can be *generated* in time $O(n)$.

Our general plan will be first to generate the implication graph for the current query and then to use this to generate the tree $[c', c'']$s. In our overall algorithm the original queries are modified at most N times each, so we need generate at most $2N$ implication graphs, at $O(n^2)$ time each (as shown in the previous section). Since we are assuming we generate $O(n^3)$ tree $[c', c'']$s, this works out to only a constant amount of time for each. In generating the implication graphs, we thus can afford to do some extra work. In particular, we order the adjacency lists for the symbol-pair vertices so that all conjunct-pair vertices with θ as second component precede all conjunct-pair vertices with a conjunct as second component. We now argue that the time for generating an individual tree $[c, c']$ is $O(n)$.

It is not difficult to see that the time for generating the children of any conjunct-pair vertex $\langle c_1, c_2 \rangle$ in tree $[c, c']$ will be proportional to length (c_1). Such a vertex $\langle c_1, c_2 \rangle$ is adjacent to at most length (c_1) neighbors. For each such neighbor $\langle y, z \rangle$ we first (temporarily) delete $\langle c_1, c_2 \rangle$ from the adjacency list for $\langle y, z \rangle$ in $G[Q]$, so that in generating the children of $\langle y, z \rangle$ we will not have to look at $\langle c_1, c_2 \rangle$, which is already in the tree. Then we add $\langle y, z \rangle$ to the tree if it is not already there. The total time is proportional to length (c_1) if adjacency lists are stored as linked lists in $G[Q]$, with special fields so that temporary deletions can be made using the trick from [1] that allows us to dispense with initialization. (We also use this trick to allow us to test in constant time whether a new symbol-pair vertex has a mapping conflict.)

For a symbol-pair vertex, none of whose children has a mapping conflict, it is not difficult to see that the time is proportional to the size of the children in tree $[c, c']$. By our deletion trick above, we know we only have to look at those children that are not already in the tree, and each label can be generated in time proportional to its length. (There is, in fact, no need to compute the profile, since profiles for all conjuncts can be generated in a preprocessing step.) What sorting needs to be done can be done in time proportional to the lengths of the profiles using standard lexicographic sorting techniques [1] and mapping conflicts with earlier vertices can once more be checked with our indexing trick.

The only potential difficulty comes when generating the children of a symbol-pair vertex $\langle y, z \rangle$ which *has* a child with a mapping conflict, since we might end up looking at all the neighbors of $\langle y, z \rangle$ and then generating only one child. However, note that, since Q and all the queries derived from it are fan-out free, $\langle y, z \rangle$ is either adjacent to a $\langle c_1, \theta \rangle$ vertex or else all of its neighbors have distinct first components.

In the first case, our generation of the implication closure ensures that the $\langle c_1, \theta \rangle$ vertices will head the adjacency list for $\langle y, z \rangle$ in the implication closure. In order to generate the only child of $\langle y, z \rangle$ (and last vertex of tree $[c, c']$), we must order these vertices lexicographically by (local address, profile) pairs. This ordering can be done in time $O(n)$, since there is at most one vertex to consider with any conjunct as first component, local addresses are of constant length and so the total length of relevant profiles is proportional to the sum of the lengths of the conjuncts.

In the latter case, the total work involved in generating all the children of $\langle y, z \rangle$ up to the one with the mapping conflict will also be $O(n)$, since it will again be proportional to the sum of the lengths of the first components. If a global address must be found for the mapping conflict indicator, this will again be doable in time proportional to the size of the already-generated tree.

Thus the time for generating tree $[c, c']$ will be proportional to its size, which is $O(n)$, plus a final $O(n)$ for the last vertex or $O(n)$ in total, and the overall running time of the equivalence testing algorithm is $O(n^4)$.

6. Concluding remarks. We have extended the class of queries for which minimization and equivalence testing can be performed in polynomial time to a new class, the "fan-out free" queries, that is a generalization of classes considered in [2], [3], [7], [8]. In particular, it allows for queries which are not restricted to untyped variables and hence can ask for transitive information about databases. The running times for our algorithms—$O(n^3)$ and $O(n^4)$ respectively—are the same as the corresponding running times in the original papers [2], [3] on "simple" queries. One direction for future research would be to try to improve the running times of these algorithms, as [7] did for the simple query algorithms. For example, in the equivalence testing problem, it would seem possible to do a certain amount of "pruning" to our tree $[c, c']$s and still get our algorithm to work, although whether such an approach will yield an asymptotic improvement in worst-case running time is more problematic.

Another direction for further research would be to further extend the class of queries that can be handled. The minimization algorithms of [7] for the class of typed queries in which no conjunct has more than one occurrence of a repeated variable (the queries obeying restriction (2) of § 2), would seem to extend more-or-less directly to the untyped case, yielding $O(n^2)$ algorithms. Although we have already remarked that queries in this class cannot ask any questions that a fan-out free query couldn't also ask, there are queries in the class that are not fan-out free. One might hence be able to construct a combined minimization algorithm, which not only handled all queries that either obeyed restriction (2) or were fan-out free, but also could deal with queries which were in neither class, but partook of certain aspects of both. Whether one could also obtain a polynomial time equivalence algorithm for such an expanded class appears to be a harder question.

One also might look to see what can be done when the restriction to *conjunctive* queries is removed. Sagiv and Yannakakis [8] examined this question in the case of typed queries. What can be said when untyped variables are allowed? Also, what about databases obeying more complicated dependencies than mere functional dependencies?

Finally, we note that the techniques of § 5 may have implications beyond database theory. Essentially what we did in this section was show that a special case of the graph isomorphism problem could be solved by constructing canonical labelled forests and using known algorithms to test for isomorphism between *them*. It may well be that there are other isomorphism or equivalence problems which, although equivalent to graph isomorphism in general, have meaningful special cases which can be solved in an analogous fashion.

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974, pp. 84–86.

[2] A. V. AHO, Y. SAGIV AND J. D. ULLMAN, *Equivalences among relational expressions*, this Journal, 8 (1979), pp. 218–246.

[3] ———, *Efficient optimization of a class or relational expressions*, ACM Trans. Database Systems, 4 (1979), pp. 435–454.

[4] A. K. CHANDRA AND P. M. MERLIN, *Optimal implementation of conjunctive queries in relational data bases*, Proc. 9th Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, New York, 1977, pp. 77–90.

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

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

[7] Y. SAGIV, *Quadratic algorithms for minimizing joins in restricted relational expressions*, Report UIUCDCS-R-79-992, Computer Science Dept., Univ. Illinois, Urbana, IL 1979.

[8] Y. SAGIV AND M. YANNAKAKIS, *Equivalences among relational expressions with the union and difference operators*, J. Assoc. Comput. Mach., 27 (1980), pp. 633–655.

[9] M. YANNAKAKIS AND C. H. PAPADIMITRIOU, *Algebraic dependencies*, Proc. 21st Symposium on Foundations of Computer Science, IEEE Computer Society, Long Beach, CA, 1980, pp. 328–332.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0003 $01.25/0

FAST PARALLEL COMPUTATION OF POLYNOMIALS USING FEW PROCESSORS*

L. G. VALIANT†, S. SKYUM‡, S. BERKOWITZ§ AND C. RACKOFF§

Abstract. It is shown that any multivariate polynomial of degree d that can be computed sequentially in C steps can be computed in parallel in $O((\log d)(\log C + \log d))$ steps using only $(Cd)^{O(1)}$ processors.

Key words. parallel computation, polynomials, complexity theory

Introduction. Hyafil [6] showed that any polynomial q of degree d that can be computed sequentially in C $\{+, -, \times\}$-steps can be computed in parallel in time proportional to $(\log d)(\log C + \log d)$. Unfortunately his method requires $C^{\log d}$ processors in general. Thus even if C and d are both bounded polynomially in terms of the number of indeterminates the number of processors required would not be. In this paper we give an improved construction that achieves the same time bound but with only $(Cd)^{\beta}$ processors, for an appropriate constant β. The achievability of such fast time bounds with only polynomially many processors was known previously only for certain specific polynomials such as the determinant [5]. Throughout we shall use the unrestricted model of parallelism described by Borodin and Munro [2].

Simultaneous resource bounds in discrete computations are discussed in detail by Cook [3]. An important positive result in that area, due to Ruzzo [8], is that context-free languages can be recognized by Boolean circuits that simultaneously have polynomial size and $\log^2 n$ depth. This is also a corollary of our present result as can be seen as follows: Consider the monotone Boolean circuit defined by the Cocke–Kasami–Younger algorithm. Regard this as an arithmetic program over the reals using the correspondence "and" $\rightarrow \times$ and "or" $\rightarrow +$. This program has linear degree and polynomial program size. An application of our construction therefore gives a fast parallel program using only polynomially many processors. In addition, the parallel program can be reinterpreted as a shallow monotone circuit, as required. The concept of "degree" for Boolean circuits exploited in this argument is discussed formally in [9].

Finally we observe that, in the terminology of [11], the question whether every polynomial family of polynomially bounded degree and program size is a p-projection of the determinant is open. An affirmative answer to this combined with Csanky's result would give an alternative proof of our main result here, at least in the case of fields of characteristic 0.

This paper is a simplification and improvement of an earlier result by Valiant and Skyum [12].

Definitions and the main theorem. Let F be a field and let $F[x_1, \cdots, x_n]$ be the ring of polynomials over indeterminates $x_1, \cdots, x_n$ with coefficients from F. A *program* f over F is a sequence of instructions

$$v_i \leftarrow v_i' \circ v_i'', \qquad i = 1, \cdots, C$$

* Received by the editors May 15, 1981, and in revised form September 16, 1982.
† Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138.
‡ Computer Science Department, Aarhus University, Aarhus, Denmark.
§ Computer Science Department, University of Toronto, Toronto, Canada M5S 1A7.

where for each value of i

(I) $v_i', v_i'' \in F \cup \{x_1, \cdots, x_n\} \cup \{v_1, \cdots, v_{i-1}\}$ and

(II) $\circ$ is one of the two ring operators $+, \times$.

Note that since $-1 \in F$, subtraction can be easily simulated. The formal polynomial computed at v_i can be defined in the obvious way and is denoted by $f(v_i)$. The degree of $f(v_i)$ in the usual sense is denoted by $d(v_i)$. For convenience, we will assume $d(v_i') \geqq d(v_i'')$. For the moment we will also assume that f is *homogeneous*; by this we mean that if $v_i \leftarrow v_i' + v_i''$, then $d(v_i') = d(v_i'')$. We say that f has size C (the number of instructions) and computes the polynomial $f(v_C)$. The following fact due to Strassen [10] shows that forcing f to be homogeneous is not a serious restriction.

FACT 1. *If a nonhomogeneous program f computes a polynomial p of degree d and has size C, then there is a homogeneous program which has size $O(Cd^2)$ and which computes $d+1$ polynomials whose sum is p.*

Outline of proof. For each line of the original program, we will have $d+1$ lines which compute the first $d+1$ homogeneous parts of the polynomial computed at the original line. If the operation is $+$, then we add similar degree components. If the operation is $\times$, then we can view each set of operands as the coefficients of a polynomial, and perform polynomial multiplication (about d^2 operations) to obtain the coefficients of a new polynomial; only the first $d+1$ of these are needed. □

We will also assume that f is a smallest possible program for computing $f(v_C)$. The following fact is easy to verify.

FACT 2. *If f is a smallest possible homogeneous program for computing $f(v_C)$, then $f(v_i)$ is not the zero polynomial for any i, and $d(v_i) \leqq d(v_C)$ for all i.*

We will denote the set $\{v_i\}$ by V, the set $\{x_i\}$ by X, and the set $V \cup X \cup F$ by $\bar{V}$. The *depth* of $v \in \bar{V}$ is the length of the longest possible sequence $u_1, \cdots, u_D$ such that $u_1 = v$, for each i, $u_{i+1} = u_i'$ or $u_{i+1} = u_i''$, and $u_D \in F \cup X$.

DEFINITION. Let $v, w \in \bar{V}$. We define $f(v; w) \in F[x_1, \cdots, x_n]$ by induction on the depth of w. Firstly, if $v = w$ (that is they are the same node) then $f(v; w) = 1$. Otherwise, if $w \in F \cup X$, then $f(v; w) = 0$. Otherwise, if $w \leftarrow w' + w''$ then $f(v; w) = f(v; w') + f(v; w'')$ and if $w \leftarrow w' \times w''$ then $f(v; w) = f(w'') \cdot f(v; w')$. (The motivation for this definition is that if $d(w) < 2d(v)$, then $f(v; w)$ turns out to be the coefficient of v' in $f(w)$ in a modified program obtained by replacing node v by a new indeterminate v'. In fact, the proof below can be rewritten so as to only use the value of $f(v; w)$ in this case.)

FACT 3. *It is easy to see that each $f(v)$ computed in the homogeneous program f is also homogeneous, that is, all of its monomials have the same degree. It is also easy to prove by induction on the depth of w that each nonzero $f(v; w)$ is homogeneous and satisfies*: $\text{degree}(f(v; w)) = d(w) - d(v)$.

DEFINITION. If $a > 0$, define $V_a = \{t \in V \mid d(t) > a, t \leftarrow t' \times t'', d(t') \leqq a\}$.

LEMMA 1. *Say that $v, w \in V$, $d(v) \leqq a < d(w)$. Then*

$$f(v; w) = \sum_{t \in V_a} (f(v; t) \cdot f(t; w)) \quad \textit{and} \quad f(w) = \sum_{t \in V_a} f(t) \cdot f(t; w).$$

Proof. Notice that $v \neq w$ since $d(v) < d(w)$. We will prove the lemma by induction on the depth of w, keeping v and a fixed.

Case 1. $w \in F \cup X$. This cannot happen since we can't have $d(w) > a > 0$.

Case 2. $w \leftarrow w' + w''$, $d(w') = d(w'') = d(w)$. Assume the lemma holds for w', w''.

$$f(v; w) = f(v; w') + f(v; w'') = \sum_{t \in V_a} (f(v; t) \cdot f(t; w')) + \sum_{t \in V_a} (f(v; t) \cdot f(t; w''))$$

$$= \sum_{t \in V_a} (f(v; t) \cdot (f(t; w') + f(t; w''))) = \sum_{t \in V_a} (f(v; t) \cdot f(t; w)).$$

Similarly, $f(w) = \sum_{t \in V_a} f(t) \cdot f(t; w)$.

Case 3. $w \leftarrow w' \times w''$, $a \geqq d(w') \geqq d(w'')$. (Recall that $d(w') \geqq d(w'')$ by definition.) Then $w \in V_a$. Clearly $f(v; w) = f(v; w) \cdot f(w; w)$. Let s be any other element of V_a. Then $f(s; w) = f(w'') \cdot f(s; w')$. But $f(s; w') = 0$ by Fact 3, since $d(s) > a \geqq d(w')$. So $f(s; w) = 0$. So $f(v; w) = \sum_{t \in V_a} (f(v; t) \cdot f(t; w))$. Similarly, $f(w) = \sum_{t \in V_a} f(t) \cdot f(t; w)$.

Case 4. $w \leftarrow w' \times w''$, $d(w) \geqq d(w') > a$. Assume the lemma holds for w'.

$$f(v; w) = f(w'') \cdot f(v; w') = f(w'') \cdot \sum_{t \in V_a} (f(v; t) \cdot f(t; w'))$$

$$= \sum_{t \in V_a} (f(v; t) \cdot f(w'') \cdot f(t; w')) = \sum_{t \in V_a} f(v; t) \cdot f(t; w).$$

Similarly, $f(w) = \sum_{t \in V_a} f(t) \cdot f(t; w)$. □

THEOREM 1. *Let f be a homogeneous program of size C which computes a polynomial p of degree d. Then there is a program f' of size $O(C^3)$ which computes p, such that the largest depth of any node is $O(\log C \log d)$.*

Proof. The construction of f' will proceed in $\lceil \log_2 d \rceil$ stages; each stage will add at most $\log C$ to the depth of the program.

At stage 0 we compute all $f(w)$ and $f(v; w)$ that have degree at most $2^0 = 1$. Since these are linear forms in n indeterminates and $C \geqq n - 1$ if f is minimal, depth $2 + \lceil \log_2 C \rceil$ is sufficient for this. At stage $i+1$ we compute all $f(w)$ and $f(v; w)$ that have degree in the range $(2^i, 2^{i+1}]$. By Fact 2 we are done after stage $\lceil \log_2 d \rceil$.

Say that $2^{i+1} \geqq d(w) > 2^i$. Let $a = 2^i$. Then by Lemma 1, $f(w) = \sum_{t \in V_a} f(t) \cdot f(t; w) = \sum_{t \in V_a} f(t') \cdot f(t'') \cdot f(t; w)$. By definition of V_a, each $f(t')$, $f(t'')$, $f(t; w)$ has already been computed. So $f(w)$ can be computed adding only $O(\log C)$ depth.

Say that $2^{i+1} \geqq d(w) - d(v) = \text{degree}\,(f(v; w)) > 2^i$. Let $a = d(v) + 2^i$. By Lemma 1, $f(v; w) = \sum_{t \in V_a} f(v; t) \cdot f(t; w) = \sum_{t \in V_a} f(t'') \cdot f(v; t') \cdot f(t; w)$. Each $f(v; t')$ and $f(t; w)$ has already been computed. It is possible, however, that $d(t'')$ is very large, say that $d(t') \geqq d(t'') > 2^{i+1}$. If $f(v; t') = 0$, then we're okay, since $f(t'') \cdot f(v; t') \cdot f(t; w)$ can still be computed. So say that $d(t'') > 2^{i+1}$ and $f(v; t') \neq 0$. Then $d(t') \geqq d(v)$, so $d(t) = d(t') + d(t'') > d(v) + 2^{i+1} \geqq d(w)$. So $f(t; w) = 0$. So $f(v; w)$ can be computed adding only $O(\log C)$ depth.

The *size* of the new program is dominated by the time to compute the $f(v; w)$. There are C^2 choices of pairs (v, w) and the computation of each $f(v; w)$ takes $O(C)$ steps. The overall size is therefore $O(C^3)$. □

By combining Theorem 1 and Fact 1 we have:

THEOREM 2. *Let f be a nonhomogeneous program of size C which computes a polynomial p of degree d. Then there is a program f' of size $O((Cd^2)^3)$ and depth $O((\log C + \log d) \log d)$ which computes p.*

There is another method for handling nonhomogeneous programs without first making them homogeneous. Given a nonhomogeneous program f, we first define the degree of a node v, $d(v)$, differently than above. The degree of a field member is 0; the degree of an indeterminate is 1; the degree of a multiplication node is the sum of the degrees of its inputs; the degree of an addition node is the maximum degree of its inputs. We define the degree of f to be the maximum degree of any node.

For $a > 0$, define $V'_a = \{t \in V \mid d(t) > a, t \leftarrow t' + t'', d(t'') \leqq a\}$. We state the following lemma and theorem without proof.

LEMMA 2. *Say that $v, w \in V$, $d(v) \leqq a < d(w)$. Then*

$$f(v; w) = \sum_{t \in V_a} (f(v; t) \cdot f(t; w)) + \sum_{t \in V'_a} (f(v; t'') \cdot f(t; w))$$

and

$$f(w) = \sum_{t \in V_a} (f(t) \cdot f(t; w)) + \sum_{t \in V'_a} (f(t'') \cdot f(t; w)).$$

THEOREM 3. *Let f be a nonhomogeneous program of size C and degree d. Then there is a program f' of size $O(C^3)$ and depth $O(\log C \log d)$ which computes the same polynomial.*

Remarks. 1. Strassen [10] showed that for infinite fields Fact 1 holds even when divisions are allowed in the original program but not in the transformed one. It follows that Theorem 2 holds under the same hypothesis. Recently Borodin, von zur Gathen and Hopcroft [1] have shown that the infinity assumption is inessential.

2. It is easy to verify that the above theorems hold for structures much less restricted than fields. For example, the constructions work for monotone arithmetics (i.e., with constants from the nonnegative reals). As observed in the introduction, the same results then follow for monotone Boolean circuits when the notion of degree is suitably interpreted. More formally, it can be verified that it is sufficient for F to be a semiring in the sense of Jerrum and Snir [7].

3. Using Lemmas 1 and 2, analogues of Theorems 1 and 3 can be proved with a size bound for f' of $C^\alpha \log d$ where α is such that $n \times n$ matrices can be multiplied in n^α operations ($\alpha < 2.496$ [4]).

REFERENCES

[1] A. BORODIN, J. VON ZUR GATHEN AND J. HOPCROFT, *Fast parallel matrix and GCD computations*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 65–71.

[2] A. BORODIN AND I. MUNRO, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975.

[3] S. A. COOK, *Deterministic CFL's are accepted simultaneously in polynomial time and log squared space*, Proc. 11th ACM Symposium on Theory of Computing, 1979, pp. 338–345.

[4] D. COPPERSMITH AND S. WINOGRAD, *On the asymptotic complexity of matrix multiplication*, Proc. 22nd IEEE Symposium on Foundations of Computer Science, 1981, pp. 82–90.

[5] L. CSANKY, *Fast parallel inversion algorithms*, this Journal, 5 (1976), pp. 618–623.

[6] L. HYAFIL, *On the parallel evaluation of multivariate polynomials*, Proc. 10th ACM Symposium on Theory of Computing, 1978, pp. 193–195.

[7] M. JERRUM AND M. SNIR, *Some exact complexity results for straight-line computations over semirings*, J. Assoc. Comput. Mach., 1982, to appear.

[8] W. L. RUZZO, *On uniform circuit complexity*, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1979, pp. 312–318.

[9] S. SKYUM AND L. G. VALIANT, *A complexity theory based on Boolean algebra*, Proc. 22nd IEEE Symposium on Foundations of Computer Science, 1981, pp. 244–253.

[10] V. STRASSEN, *Vermeidung von Divisionen*, J. Reine Angew. Math., 264 (1973), pp. 182–202.

[11] L. G. VALIANT, *Completeness classes in algebra*, Proc. 11th ACM Symposium on Theory of Computing, 1979, pp. 249–261.

[12] L. G. VALIANT AND S. SKYUM, *Fast parallel computation of polynomials using few processors*, Lecture Notes in Computer Science, 118, Springer-Verlag, New York, 1981, pp. 132–139.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0004 $01.25/0

UNIFORM RANDOM GENERATION OF STRINGS IN A CONTEXT-FREE LANGUAGE*

TIMOTHY HICKEY† AND JACQUES COHEN†

Abstract. Let S be the set of all strings of length n generated by a given context-free grammar. A uniform random generator is one which produces strings from S with equal probability. In generating these strings, care must be taken in choosing the disjuncts that form the right-hand side of a grammar rule so that the produced string will have the specified length. Uniform random generators have applications in studying the complexity of parsers, in estimating the average efficiency of theorem provers for the propositional calculus, in establishing a measure of ambiguity of a grammar, etc. Two methods are presented for generating uniform random strings in an unambiguous context-free language. The first method will generate a random string of length n in linear time, but must use a precomputed table of size $O(n^{r+1})$, where r is the number of nonterminals in the grammar used to specify the language. The second method precomputes part of the table and calculates the other entries as they are called for. It requires only linear space, but uses $O(n^2(\log n)^2)$ time to generate each string. Both methods generate strings by leftmost derivations where the probability that a given production will be used depends on the history of the derivation. It is also shown that, in the special cases of finite-state or linear languages, the generation can be performed in linear time with constant space.

Key words. uniform random generation, balanced parenthesis strings, convolution, enumeration problems, leftmost derivations, generating functions, parsing

1. Introduction.

Given a context-free grammar it is straightforward to generate "random" terminal strings: starting with the main nonterminal the sentential forms are constructed by considering equiprobable each of the disjuncts forming the right-hand side of a rule. The resulting strings will be of varying lengths.

To generate strings of a fixed length, one may have to bypass the random choice of disjuncts and choose a particular disjunct which is likely to lead more directly to a terminal string. Consider the set of all strings of length n in the language. A uniform random generator is one which will produce strings from this set *with equal probability*. The generation of such strings is a subtle process because the probabilities with which disjuncts are used at each step in a derivation depend on the previous steps of the derivation.

A generator of uniform random strings has several applications in computer science:

(1) The complexity of parsers is usually expressed as a function of the length of the string being parsed; thus the generator is useful in estimating their average case efficiency. For example, Cohen and Roth [5] describe a method for determining the execution times of top-down and bottom-up parsers. The execution times are expressed in terms of the number of occurrences of each terminal in the given string. The analyses of these parsers could be either experimental (by generating and parsing a large number of strings) or analytic (by determining a formula which takes into account the above probabilities). The generation of uniform random strings is also useful in estimating the average speedup in parallel parsing [4].

(2) Formulas in the propositional calculus can be generated by context-free grammars. If the length of formulas is of interest, one can use a uniform random generator to estimate the number of tautologies of a given length, the percentage which use certain operators, etc. Uniform random generators are also useful in estimating the relative efficiencies of theorem provers from the propositional calculus.

* Received by the editors November 20, 1981, and in revised form September 27, 1982. This research was supported by the National Science Foundation under grant number MCS 79-05522.

† Computer Science Program, Brandeis University, Waltham, Massachusetts 02254.

(3) When the generating techniques described below are applied to ambiguous grammars, the probability that an ambiguous string will be generated is directly proportional to the number of parses of the string. By generating strings of length n one can estimate the degree of ambiguity in the language as a function of n.

2. Objectives. In a recent paper Arnold and Sleep [2] propose a method for generating uniform random strings of a predetermined length in a nested parentheses language. Their method involves computing the probabilities with which a left or right parentheses should be generated. Based on the number of unmatched left parentheses already generated (u), and the number of symbols remaining to be generated (s), a left parentheses is generated with probability:

$$p_L(u, s) = \frac{(s-u)(u+2)}{2s(u+1)}. \tag{1}$$

Using their method, every string of the chosen length has an equal probability of being generated.[1]

Our objective is to extend the Arnold and Sleep technique to unambiguous context-free languages. We generate tables of probabilities that are consulted when performing a leftmost derivation of a uniform random string. These tables specify the probability with which a production should be applied. A crucial point is that our probabilities depend on the current state of the derivation and not just the productions.

We describe and analyze two generation methods in the sequel. The first computes all of the probabilities that could possibly be needed to produce a string of the given length. It can then generate uniform random strings in linear time. The total time complexity is $O(n^{r+1}+nm)$ and the space needed is $O(n^{r+1})$ where n is the desired string length, m the number of random strings generated and r the number of nonterminals in the grammar. The second method computes the probabilities as they are needed. This reduces the needed space to $O(n)$ but requires more time to generate each string. The total time complexity is $O(n^2 \log n + mn^2(\log n)^2)$.

3. Notation and preliminary remarks. Let $G = (V, \Sigma, P, N_1)$ be an unambiguous cycle-free[2] context-free grammar containing no ε-productions where $V = N \cup \Sigma$, $N = \{N_1, \cdots, N_r\}$ is the nonterminal vocabulary, Σ is the set of terminals, N_1 is the start symbol, and $P = \{\pi_{ij} : N_i \to \alpha_{ij} | i = 1, \cdots, r, j = 1, \cdots, s_i\}$ is the set of productions (see [6] for detailed definitions).

We choose to generate strings by leftmost derivation, but any canonical method of derivation would be adequate. A leftmost derivation in G of a terminal string ω consists of a sequence $\beta_1, \beta_2, \cdots, \beta_t$ of sentential forms where $\beta_1 = N_1$, $\beta_t = \omega$, and β_{k+1} is obtained by rewriting the leftmost nonterminal in β_k using one of the productions in P. If N_i is the leftmost nonterminal in β_k, there are s_i different choices for β_{k+1} corresponding to the productions π_{ij}, $j = 1, \cdots, s_i$. The methods described in the next two sections generate random strings of length n by determining the probabilities $p_{ij}(\beta, n)$ with which production π_{ij} should be applied to β.

The probabilities are chosen so that every string of length n has an equal probability of being generated. Let $g_\beta(n)$ denote the number of terminal strings of length n that can be derived from a sentential form β and let γ be the result of

[1] The methods described by Wetherell [8] involve assigning a fixed probability to each production in the grammar. This scheme cannot, in general, be used to produce strings of a predetermined length because it negects the history of the derivation.

[2] G is cycle-free if no nonterminal can be derived from itself.

applying π_{ij} to β in a leftmost derivation. Since $g_\gamma(n)$ is the number of terminal strings of length n which can be derived from β by first applying π_{ij}, the probability that must be assigned to π_{ij} is:

$$p_{ij}(\beta, n) = \frac{g_\gamma(n)}{g_\beta(n)}. \tag{2}$$

In the remainder of this section generating functions are introduced and used to express $g_\beta(n)$ as a convolution (defined below) of the simpler functions $g_{N_i}(n)$. A terminal string of length n derived from the concatenation of two strings, β_1, β_2, is the concatenation of two terminal strings whose lengths sum to n. The number of such length n terminal strings can therefore be expressed as a convolution:

$$g_{\beta_1\beta_2}(n) = \sum_{n_1+n_2=n} g_{\beta_1}(n_1) g_{\beta_2}(n_2) = (g_{\beta_1} * g_{\beta_2})(n). \tag{3}$$

Note that if the grammar is ambiguous this convolution gives the number of parses of terminal strings of length n.

Several properties of the convolution operator that will be used in this paper are discussed in this paragraph. Let $g(n)$, $g_1(n)$ and $g_2(n)$ be functions defined on the integers which are zero for negative values of n. The convolution of g_1 and g_2 is defined by

$$(g_1 * g_2)(n) = \sum_{n_1+n_2=n} g_1(n_1) g_2(n_2) = \sum_{k=0}^{n} g_1(k) g_2(n-k). \tag{4a}$$

Although convolution can be studied directly, its properties are easier to understand in the context of generating functions (see [7, p. 86]). The generating function of g is the formal power series $Z[g](x) = \sum_{n=0}^{\infty} g(n) x^n$. A simple computation shows that the generating function of the convolution of two functions is the product of their respective generating functions

$$Z[g_1 * g_2] = Z[g_1]\, Z[g_2]. \tag{4b}$$

An immediate consequence of this interpretation is the associativity and commutativity of convolution:

$$g_1 * g_2 = g_2 * g_1, \tag{4c}$$

$$(g_1 * g_2) * g_3 = g_1 * (g_2 * g_3), \tag{4d}$$

which follow from the corresponding properties of power series multiplication. Let δ_k be the function whose generating function is x^k:

$$\delta_k(n) = \begin{cases} 1, & n = k, \\ 0, & n \neq k. \end{cases} \tag{4e}$$

It satisfies $Z[\delta_k * g] = Z[\delta_k] * Z[g] = x^k Z[g]$ by (4b), and this implies the next two properties:

$$(\delta_k * g)(n) = g(n-k), \qquad \delta_0 * g = g, \tag{4f}$$

$$(\delta_k * \delta_j) = \delta_{k+j}. \tag{4g}$$

The notation $g^{(k)}$ is used to denote the convolution of a function g with itself k times, with the convention that $g^{(0)} = \delta_0$. With this notation the relation $g^{(k)} * g^{(j)} = g^{(k+j)}$ holds for any nonnegative integers k and j.

These properties can now be used to show that the value of $g_\beta(n)$, for any string β, depends only on the number of terminal symbols that remain to be generated by the derivation process and on the number of occurrences of each nonterminal in the string. Let $T(\beta)$ be the number of terminals in β (so $n-T(\beta)$ terminals remain to be generated) and let $A(\beta)=(A_1(\beta),\cdots,A_r(\beta))$ be the vector whose ith component is the number of occurrences of N_i in β. Define the function $f(t,a)$ for $a=(a_1,\cdots,a_r)$ by

$$f(t,a)=(g_{N_1}^{(a_1)}*\cdots*g_{N_r}^{(a_r)})(t). \tag{5}$$

When the vector a is fixed the notation $f_a(t)$ will sometimes be used instead of $f(t,a)$. If a and a' are vectors and $a+a'$ is their componentwise sum, then the relation $f_{a+a'}=f_a*f_{a'}$ holds by virtue of the commutativity and associativity of convolution.

In this paragraph, the following fundamental relation will be proved by induction on the length of β

$$g_\beta=\delta_{T(\beta)}*f_{A(\beta)},\qquad g_\beta(n)=f(n-T(\beta),A(\beta)). \tag{6}$$

Notice that the second equation is equivalent to the first by the convolution property of the delta function (4f). If β is a single symbol, the relation certainly holds. Suppose β is the concatenation of two strings β_1, β_2, neither of which is the empty string, then $g_\beta=g_{\beta_1}*g_{\beta_2}$ by (3), and the inductive step follows easily from properties (4a)–(4g):

$$\begin{aligned} g_\beta &= g_{\beta_1}*g_{\beta_2}=\delta_{T(\beta_1)}*f_{A(\beta_1)}*\delta_{T(\beta_2)}*f_{A(\beta_2)}=\delta_{T(\beta_1)}*\delta_{T(\beta_2)}*f_{A(\beta_1)}*f_{A(\beta_2)}\\ &=\delta_{T(\beta_1)+T(\beta_2)}*f_{A(\beta_1)+A(\beta_2)}=\delta_{T(\beta)}*f_{A(\beta)}. \end{aligned} \tag{7}$$

This implies that $g_\beta(n)$ depends only on $n-T(\beta)$ and $A(\beta)$, as we claimed.

In our new notation the formula (2) for the probability that production π_{ij} will be used becomes

$$p_{ij}(\beta,n)=\frac{f(n-T(\gamma),A(\gamma))}{f(n-T(\beta),A(\beta))}=\frac{f(n-T(\beta)-T(\alpha_{ij}),A(\beta)-A(N_i)+A(\alpha_{ij}))}{f(n-T(\beta),A(\beta))}. \tag{8}$$

Both methods use this formula to generate random strings. They differ in the way the values of f are calculated, and in the amount of precomputation performed.

4. Method 1. In the first method we compute $f(t,a)$ for every t and a that could possibly arise in a derivation of a string of length n and we store these values in an $r+1$ dimensional array, r being the number of nonterminals in the grammar. Since the grammar contains no ε-productions, the values of $f(t,a)$ that need to be considered are those with $0\leqq t\leqq n$ and $a_1+\cdots+a_r\leqq t$.

The entries in the array can be computed efficiently using the following recurrence

$$f(t,a)=\sum_{j=1}^{s_i} f(t-T(\alpha_{ij}),a+A(\alpha_{ij})-A(N_i)),\quad \text{if } a_i\neq 0, \tag{9}$$

which we now explain. The initial conditions are that $f(t,a)=0$ for all a and all $t\leqq 0$, with the single exception $f(0,(0,\cdots,0))=1$. Let β be a nonterminal string with $A(\beta)=a$ whose leftmost symbol is N_i; this is possible if $a_i\neq 0$. The left-hand side of (9) is the number of terminal strings of length t that can be derived from β. The jth term on the right-hand side of (9) is the number of length t terminal strings that can be derived from β after the leftmost symbol is rewritten using production π_{ij}. Since one of the productions $\pi_{ij}(j=1,\cdots,s_i)$ must be used to rewrite N_i, (9) holds.

If each α_{ij} contains a terminal (as is the case with grammars in Greibach normal form) the recurrence above relates $f(t, a)$ to values of f with a smaller value of t. The array can then be filled for increasing values of t in a straightforward manner.

However, if some production does not contain any terminals, care must be taken to ensure that new values of f are expressed in terms of previously computed values. let $|a| = a_1 + a_2 + \cdots + a_r$ for a vector a. Observe that for all α_{ij} satisfying $T(\alpha_{ij}) = 0$ and for all a with $a_i > 0$ we have $|a| \leqq |a - A(N_i) + A(\alpha_{ij})|$ with equality holding if and only if production π_{ij} has the form $N_i \to N_k$ for some k. Since the grammar is not cyclic the nonterminals can be ordered in such a way that this occurs only when $i > k$. The following loop structure will now compute the values of f needed to derive any string of length n.

$f \leftarrow 0$; $f(0, (0, \cdots, 0)) \leftarrow 1$;
for $t \leftarrow 1$ **to** n **do**
for $s \leftarrow t$ **downto** 0 **do**
{*s represents the component summation* $|a| = a_1 + \cdots + a_r$}
  for $a_1 \leftarrow s$ **downto** 0 **do**
  for $a_2 \leftarrow s - a_1$ **downto** 0 **do**
  $\cdots$
  for $a_{r-1} \leftarrow s - (a_1 + \cdots + a_{r-2})$ **downto** 0 **do**
    begin
      $a_r \leftarrow s - (a_1 + \cdots + a_{r-1})$;
      {*use* (9) *for any i with* $a_i \neq 0$ *compute* $f(t, (a_1, \cdots, a_r))$}
    end

This method requires time and space $O(n^{r+1})$ to fill the table. Since the grammar contains no ε-productions and is not cyclic, the generation of a random terminal string of length n will involve $O(n)$ rewritings. Therefore, $O(n)$ probability computations which can each be formed in constant time are required for every terminal string generated. The total time complexity is therefore $O(n^{r+1} + mn)$, where m is the number of terminal strings generated, and the space complexity is $O(n^{r+1})$.

5. Method 2. In Method 2 the values of $f(t, a)$ needed in the probability formula (8) are computed while the string is being generated using the definition of $f(t, a)$ by convolution (5)

$$f(t, a) = (g_{N_1}^{(a_1)} * \cdots * g_{N_r}^{(a_r)})(t).$$

The functions $g_{N_i}(t)(1 \leqq t \leqq n)$ are precomputed in time $O(n^2 \log n)$ and space $O(n)$ using the technique described in the next paragraph. The computation of $f(t, a)$ is performed by first calculating the convolution powers, $g_{N_i}^{(a_i)}(t)(1 \leqq t \leqq n)$, using the precomputed functions $g_{N_i}(t)(1 \leqq t \leqq n)$, and then performing r convolutions. These powers can each be computed with at most $2 \cdot \log_2 n$ convolutions since a_i is no larger than n. The convolution of two functions defined for $1 \leqq t \leqq n$ can be performed in time $O(n \log n)$ using the fast Fourier transform (see [1, p. 263, Cor. 1]) or in time $O(n^2)$ directly. Since $O(n)$ probability computations must be performed for every terminal string generated, the total time complexity of method 2 is $O(n^2 \log n + mn^2 (\log n)^2)$. The space requirement is $O(n)$ since the only precomputed values are $g_{N_i}(t)(1 \leqq t \leqq n, 1 \leqq i \leqq r)$,

The recurrence (9) for $f(t, a)$ and the convolution definition of f can be used to perform the precomputation of the functions g_{N_i} in time $O(n^2 \log n)$ and linear space.

Since $g_{N_i}(t) = f(t, A(N_i))$, (9) yields

$$g_{N_i}(t) = \sum_{j=1}^{s_i} f(t - T(\alpha_{ij}), A(\alpha_{ij})).$$

Let e_{kj} represent $a_k(\alpha_{ij})$. The convolution definition of f transforms this equation into one involving only the functions g_{N_j}:

$$g_{N_i}(t) = \sum_{j=1}^{s_i} (g_{N_1}^{(e_{1j})} * \cdots * g_{N_r}^{(e_{rj})})(t - T(\alpha_{ij})). \tag{10}$$

Since $g_{N_i}(0) = 0$ for all i and $T(\alpha_{ij}) \geqq 0$ for all i and j, this equation expresses $g(t)$ in terms of the values of $g_{N_i}(t')(1 \leqq t' < t, 1 \leqq i \leqq r)$. Equation (10) involves a constant number of convolutions and must be applied nr times to calculate the requisite values of the g_{N_i}. The total precomputation performed in this way requires time $O(n^2 \log n)$ and space $O(n)$ as claimed.

6. Examples. Consider the unambiguous context-free grammar $G_0 = (\{N_1, N_2, (\ , \ )\}, \{(\ , \ )\}, P, N_1)$ with productions:

$$\begin{aligned} P: \quad & \pi_{11}: N_1 \to (N_2 \\ & \pi_{21}: N_2 \to N_1 N_2 \\ & \pi_{22}: N_2 \to), \end{aligned}$$

which generates strings of well-balanced parentheses enclosed in a pair of matching parentheses.

If the outermost pair of parentheses are stripped from the strings in the language $L(G_0)$ one obtains the language of Arnold and Sleep [2].

The recurrence relations used by method 1 to calculate the values of f are

$$f(t, (a_1, a_2)) = \begin{cases} f(t-1, (a_1 - 1, a_2 + 1)), & \text{if } a_1 \geqq 1, \\ f(t, (a_1 + 1, a_2)) + f(t-1, (a_1, a_2 - 1)), & \text{if } a_2 \geqq 1. \end{cases}$$

If both a_1 and a_2 are nonzero either formula may be used. If the leftmost nonterminal of a sentential form β is N_1 then production π_{11} must be used to rewrite β. If an N_2 is leftmost the probability formula (8) must be used to determine which of π_{21} and π_{22} should be used. Let $T(\beta) = b$ and $A(\beta) = (a_1, a_2)$, then the probability formulas are:

$$p_{21}(\beta, n) = \frac{f(n-b, (a_1 + 1, a_2))}{f(n-b, (a_1, a_2))}, \qquad p_{22}(\beta, n) = \frac{f(n-b-1, (a_1, a_2 - 1))}{f(n-b, (a_1, a_2))}.$$

Figure 1 shows the probabilities needed to generate any string of length 8. This figure is analogous to [2, Fig. 1]. Table 1 shows the values of $f(t, a)$ needed to compute those probabilities.

Method 2 computes the values of g_{N_1} and g_{N_2} for $1 \leqq t \leqq n$ using the formulas

$$\begin{aligned} & g_{N_1}(t) = g_{N_2}(t-1), \\ & g_{N_2}(t) = (g_{N_1} * g_{N_2})(t) + (g_{N_1}^{(0)} * g_{N_2}^{(0)})(t-1) = (g_{N_2} * g_{N_2})(t-1) + \delta_1(t), \end{aligned}$$

and the initial conditions $g_{N_i}(t) = 0$ for $t \leqq 0$.

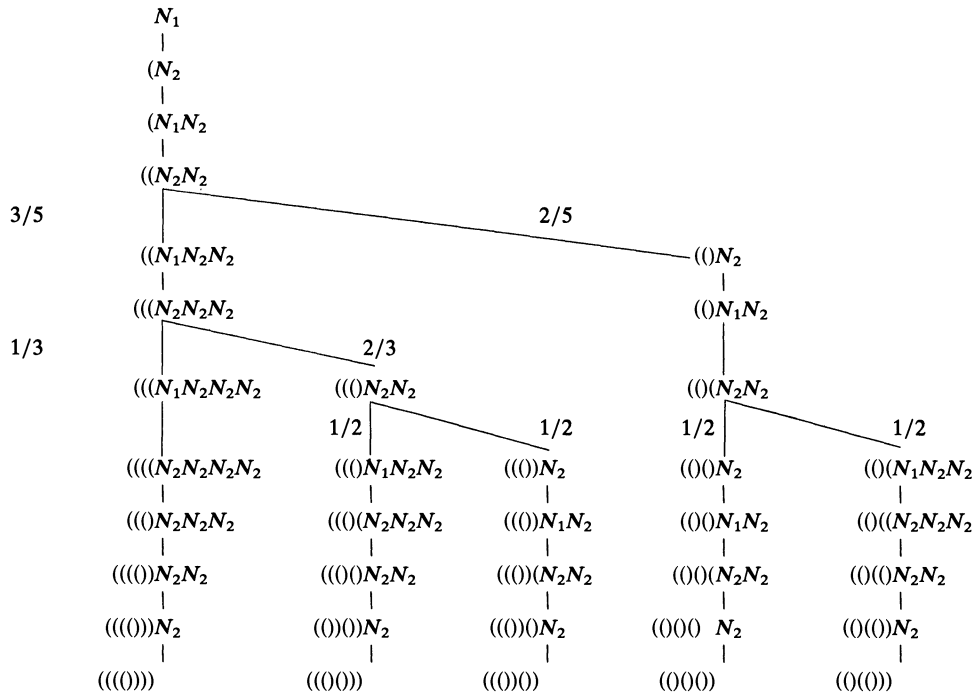


FIG. 1. *Generation of strings of length* 8. (*Note: The fractions indicate the computed probabilities.*)

TABLE 1
The function $f(t, (a_1, a_2)) = (g_{N_1}^{(a_1)} * g_{N_2}^{(a_1)})(t)$.

a_1	a_2	$f_{(a_1,a_2)}(t)$	t 0	1	2	3	4	5	6	7	8
0	0	$\delta_0(t)$	1	0	0	0	0	0	0	0	0
0	1	$g_{N_2}(t)$	0	1	0	1	0	2	0	5	0
0	2	$g_{N_2}^{(2)}(t)$	0	0	1	0	2	0	5	0	14
0	3	$g_{N_2}^{(3)}(t)$	0	0	0	1	0	3	0	9	0
0	4	$g_{N_2}^{(4)}(t)$	0	0	0	0	1	0	4	0	14
1	0	$g_{N_1}(t)$	0	0	1	0	1	0	2	0	5
1	1	$(g_{N_1} * g_{N_2})(t)$	0	0	0	1	0	2	0	5	0
1	2	$(g_{N_1} * g_{N_2}^{(2)})(t)$	0	0	0	0	1	0	3	0	9
1	3	$(g_{N_1} * g_{N_2}^{(3)})(t)$	0	0	0	0	0	1	0	4	0

7. Finite state grammars. In a leftmost derivation of a terminal string of length n using a regular grammar G, each sentential form β except for the last contains exactly one nonterminal. If β contains N_i then $g_\beta(n) = f(n - T(\beta), A(N_i)) = g_{N_i}(n - T(\beta))$ by the fundamental relation (6). Since the probabilities can be determined directly from the functions $g_{N_1}(t), \cdots, g_{N_r}(t)$, Methods 1 and 2 coincide. The recurrence relation (9) for G is a finite difference equation and so $g_{N_1}(t), \cdots, g_{N_r}(t)$, can be computed for $1 \leqq t \leqq n$ in linear time. Therefore the time complexity is $O(n + mn)$ and the space required is $O(n)$, but this is not always optimal! Cohen and Katcoff [3] describe and analyze an algorithm for finding closed form expressions for the functions $g_{N_1}, \cdots, g_{N_r}$ of a regular grammar G. The algorithm involves finding the (possibly complex) roots $Z_1, \cdots, Z_q$ of a polynomial with integer coefficients. The functions

g_{N_i} then have the form

$$g_{N_i}(t) = \sum_{j=1}^{g} U_{ij} Z_j^t + U_{i0}, \tag{11}$$

for some complex numbers U_{ij}. Let $R(n)$ be the time required to find the values of the Z_i and U_{ij} to an accuracy sufficient to determine $g_{N_i}(t)$ precisely for all $t \leqq n$. If the roots are integers then $R(n)$ is constant; otherwise it depends on the polynomial and the method used to find the roots.

The tth power of a complex number can be computed in time $O(\log n)$, if $t \leqq n$. This provides a method for generating random strings in time $O(R(n)+mn \log n)$ and constant space, where m is the number of terminal strings. This method will be superior to the previous one if complexity is measured by the product of the time-taken by the space-required.

Observe that the values of $g_{N_i}(t)$ can be computed from $Z_1^t, \cdots, Z_q^t$ in constant time and Z_i^t can be computed from Z_i^{t+1} and Z_i in constant time. This suggests that we precompute $Z_1, \cdots, Z_q$, and $Z_1^n, \cdots, Z_q^n$ in time $O(R(N)+\log n)$. The values of $g_{N_1}(t), \cdots, g_{N_r}(t)$ can then be computed from the stored values of $Z_1^t, \cdots, Z_2^t$ which are then replaced by $Z_1^{t-1}, \cdots, Z_q^{t-1}$ in constant time. This method will generate random terminal strings in the regular language in linear time using constant space. The total time requirement is $O(R(n)+\log n+mn)$. If $R(n)$ is constant this is the best possible performance for an infinite language.

Consider for example the regular grammar, G_1 given in [3] which generates all strings with an even numbers of 0's and 1's and any number of 2's and 3's

$$\begin{aligned} &G_1 = (\{N_1, N_2, 0, 1, 2, 3\}, \{0, 1, 2, 3\}, P, N_1), \\ &P: N_1 \to 0N_3|1N_2|2N_1|3N_1|2|3 \\ &\quad N_2 \to 0N_4|1N_1|2N_2|3N_2|1 \\ &\quad N_3 \to 0N_1|1N_4|2N_3|3N_3|0 \\ &\quad N_4 \to 0N_2|1N_3|2N_4|3N_4. \end{aligned} \tag{12}$$

The formulas (13) for $g_{Ni}(1 \leqq i \leqq 4)$ involve powers of the integer 2, and so $R(n) = 0$.

$$\begin{aligned} &g_{N_1}(t) = 2^{2(t-1)} + 2^{(t-1)}, \qquad g_{N_2}(t) = 2^{2(t-1)}, \\ &g_{N_3}(t) = 2^{2(t-1)}, \qquad g_{N_4}(t) = 2^{2(t-1)} - 2^{(t-1)}. \end{aligned} \tag{13}$$

The probabilities $p_{ij}(\beta, n)$ for this example are all very close to 0.25 when $n - T(\beta)$ is large. Let $\beta = \zeta N_i$ for some terminal string ζ of length $T(\beta)$ and let $t = n - T(\beta)$ be the number of terminal symbols still to be generated; then the probability with which production π_{ij} should be used is:

$$p_{ij}(\zeta N_i, n) = 0.25 + \frac{E(i,j)2^{t-3}}{g_{N_i}(t)} \quad \text{for } t > 1,$$

where $E(i, j)$ is determined by Table 2.

When $t = 1$, π_{15} and π_{16} have probability 0.5 while π_{25} and π_{35} have probability 1.0.

The results in this section extend naturally to linear grammars (where each production contains at most one nonterminal on its right-hand side).

TABLE 2

$E(i,j)$		j 1	2	3	4
i	1	−1	−1	1	1
	2	−2	2	0	0
	3	2	−2	0	0
	4	1	1	−1	−1

8. Final remarks. We have restricted our attention to unambiguous grammars. If our methods are applied to an ambiguous grammar, the probability that a given string of the predetermined length will be generated is proportional to the number of leftmost derivations of the string. In fact, our methods will generate uniform random derivations of terminal strings of length n. Since the strings with the most parses are most likely to be generated, these methods could be used empirically to search for ambiguity in a grammar.

Let A_n be the expected number of parses of a string of length n. If T_n is the number of strings of length n and D_n is the number of leftmost derivations of strings of length n, then $A_n = D_n/T_n$. The probability that the methods discussed in this paper will generate an ambiguous string of length n is the proportion of all derivations of length n strings that yield ambiguous strings. A lower bound for the proportion is $(D_n - T_n)/D_n = 1 - 1/A_n$, because at most T_n derivations can yield unambiguous strings. The probability that M samples will fail to yield an ambiguous string is less than $(1/A_n)^M$. This observation can be used to obtain an upper bound on the average ambiguity of strings of length n for a grammar with a given level of confidence. If $A_n \geqq c$, the probability that M samples by our method fail to yield an ambiguous string is at most $(1/c)^M$. Thus, the estimate $A_n < c$ can be made with a confidence level of $1 - 1/R$ provided at least $\log_c(R)$ samples are generated and reveal no ambiguity.

Arnold and Sleep's algorithm requires linear time and constant space even though the language is not regular. Their formula, (1), for the probabilities can be derived from the present work as follows. In a leftmost derivation using the grammar G_0 of § 6 the sentential forms β will have the form $\tau N_1 N_2^u$ or τN_2^u where τ is a terminal string of length $T(\beta)$. Rather than storing β we can output τ, store u, and store the Boolean value "N_1 *is in the stack*". In this way, the current state of the derivation can be stored using two locations.

Let the probabilities $p_{2j}(\tau N_2^u, n)$ for $j = 1, 2$ be denoted by the abbreviated notation $p_{2j}(u, s)$, where $s = n - T(\beta)$. Observe that u is the number of unmatched open parentheses in τ and s is the number of terminals that remain to be generated. Since the strings generated by Arnold and Sleep can be obtained from the strings in the language $L(G_0)$ by removing the outermost pair of parentheses, our probabilities are related to theirs by:

$$p_{21}(u, s) = p_L(u-1, s-1), \qquad p_{22}(u, s) = p_R(u-1, s-1).$$

In the Appendix an analytic formula for $f(t, a)$ is derived using generating functions and the Lagrange inversion formua [6]. When $a = (0, k)$ the formula specializes to

$$f(t, (0, k)) = \frac{k}{t} \frac{t!}{((t-k)/2)!((t+k)/2)!}.$$

By relation (9) $f(t, (1, k)) = f(t-1, (0, k+1))$ and so by relation (8) we obtain a simple formula for the probability of generating a left parenthesis:

$$p_{21}(u, s) = \frac{f(s-1, (0, u+1))}{f(s, (0, n))} = \frac{(u+1)(s-u)}{2u(s-1)}. \tag{14}$$

This agrees with Arnold and Sleep's formula (1).

An interesting open problem is to determine the class of languages for which random strings can be generated in linear time and space. We have shown that all regular languages are in this class. Moreover, any grammar for which the probability functions can be computed in constant time (as in (14)) will generate a language in this class.

Appendix. This appendix illustrates the use of the Lagrange inversion formula (see [6, p. 40]) to obtain analytic formulas for the function $f(t, a)$ in the case of the parentheses grammar, G_0, of § 6. The method we describe is applicable to any unambiguous grammar in Greibach normal form which has only one nonterminal symbol (i.e., generalized prefix polish notation).

Let $G_i(x) = \sum_{t=0}^{\infty} g_{N_i}(t)x^t$ be the generating functions of g_1 and g_2. In § 6 the following equations, (15), were needed to perform the precomputation of method 2

$$g_{N_1} = \delta_1 * g_{N_2}, \qquad g_{N_2} = \delta_1 * g_{N_2} * g_{N_2} + \delta_1. \tag{15}$$

These equations translate into the following set of power series equations for generating functions (see 4b):

$$G_1 = xG_2, \qquad G_2 = xG_2^2 + x = x(G_2^2 + 1).$$

The Lagrange inversion formula states that if $H_1(w)$ and $H_2(w)$ are analytic functions of w (e.g., polynomial) and if $G(x)$ is a power series in x that satisfies $G(x) = xH_1(G(x))$, then $H_2(G(x))$ has the following form:

$$H_2(G(x)) = \sum_{m=0}^{\infty} \frac{x^m}{m!}\left(\frac{d^{m-1}}{dw^{m-1}}[H_2'(w) \cdot H_1(w)^m]\Big|_{w=0}\right). \tag{18}$$

In our case $H_1(w) = w^2 + 1$.

To derive a general formula for $f(t, (a_1, a_2))$ we can use the recurrence equations derived in § 6 to simplify the problem. Recall that $f(t, (a_1, a_2)) = f(t-1, (a_1-1, a_2+1))$ whenever $a_1 > 0$. Iteration of this relation shows that $f(t, (a_1, a_2)) = f(t-a_1, (0, a_2+a_1))$. Thus it will suffice to find an analytic formula for $f(t, (0, k)) = g_{N_2}^{(k)}(t)$. Since the power series of $g_{N_2}^{(k)}$ is G_2^k, (16) with $H_2(w) = w^k$ will give the desired formula.

The polynomial $H_2'H_1^m$ when expanded by the binomial theorem has the form

$$kw^{k-1} \sum_{j=0}^{m} \binom{m}{j} w^{2j}.$$

The $(m-1)$st derivative of this evaluated at zero and divided by $m!$ will yield the formula

$$g_{N_2}^{(k)}(m) = \frac{1}{m!}k(m-1)!\binom{m}{(m-k)/2} = \frac{k}{m}\frac{m!}{((m-k)/2)!((m+k)/2)!}.$$

From this we obtain the desired result:

$$f(t, (a_1, a_2)) = \frac{(a_1+a_2)}{(t-a_1)}\frac{(t-a_1)!}{((t-2a_1-a_2)/2)!((t+a_2)/2)}.$$

REFERENCES

[1] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1978.

[2] D. B. ARNOLD AND M. R. SLEEP, *Uniform random generation of balanced parentheses strings*, ACM Trans. Programming Languages and Systems, 2 (1980), pp. 122–128.

[3] J. COHEN AND J. KATCOFF, *Automatic solution of a certain class of combinatorial problems*, Inform. Proc. Letters, 6 (1977), pp. 101–104.

[4] J. COHEN, T. HICKEY AND J. KATCOFF, *Upper bounds for speedup in parallel parsing*, J. Assoc. Comput. Mach., 29 (1982), pp. 408–428.

[5] J. COHEN AND M. ROTH, *Analyses of deterministic parsing algorithms*, Comm. ACM, 21 (1978), pp. 448–458.

[6] M. A. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

[7] D. E. KNUTH, *The Art of Computer Programming, Vol.* 1: *Fundamental Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1973.

[8] C. S. WETHERELL, *Probabilistic languages: A review and some open questions*, ACM Computing Surveys, 12 (1980), pp. 361–379.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0005 $01.25/0

AUTHENTICATED ALGORITHMS FOR BYZANTINE AGREEMENT*

D. DOLEV† AND H. R. STRONG†

Abstract. Reaching agreement in a distributed system in the presence of faulty processors is a central issue for reliable computer systems. Using an authentication protocol, one can limit the undetected behavior of faulty processors to a simple failure to relay messages to all intended targets. In this paper we show that, in spite of such an ability to limit faulty behavior, and *no matter what message types or protocols* are allowed, reaching (Byzantine) agreement requires at least $t+1$ phases or rounds of information exchange, where t is an upper bound on the number of faulty processors. We present algorithms for reaching agreement based on authentication that require a total number of messages sent by correctly operating processors that is polynomial in both t and the number of processors, n. The best algorithm uses only $t+1$ phases and $O(nt)$ messages.

Key words. authentication, reliable distributed systems, Byzantine agreement, consistency, unanimity

1. Introduction. In this paper we consider algorithms for achieving agreement among multiple processors. The context for this agreement is a network of unreliable processors that have a means for conducting several synchronized phases of information exchange, after which they must all agree on some set of information. We will assume for simplicity that this set of information consists of a single value from some set of values V.

The type of agreement we will study is called Byzantine agreement (LSP), unanimity (Db) or interactive consistency (PSL). It results when in the presence of undetected faulty processors, all *correct* (nonfaulty) processors are able to agree either on a value or on the conclusion that the originator of the value is faulty. More explicitly, Byzantine agreement is achieved when

(I) all correct processors agree on the same value, and

(II) if the sender is correct, then all correct processors agree on its value.

Implicit in (I) and (II) is the idea that the agreement is synchronous in the sense that all processors reach this agreement at the same time. In other words, there must be some real time at which each of the processors has completed the execution of its algorithm for reaching agreement, and this time must be known and agreed on by all processors in advance.

Our analysis of the problem is based on the worst case assumption that faulty processors are not predictable and possibly even malicious. An algorithm should sustain any strange behavior of faulty processors, even a collusion to prevent the correct processors from reaching agreement. Even if the correct processors cannot identify the faulty processors, they must still reach Byzantine agreement. The algorithm should not depend in any way on anticipated behavior of faulty processors.

We establish an exact lower bound for the number of phases of information exchange required. This lower bound $(t+1)$ was known for the case in which only unauthenticated messages are exchanged (FL). We have generalized the proof given by Lynch and Fischer to apply to any kind of message. The lower bound result is somewhat surprising in our context. It indicates that even though we allow correct processors to exchange any kind of verifiable information, and even though we restrict the possible behavior of faulty processors to simply failing to relay messages, Byzantine

* Received by the editors January 4, 1982, and in revised form September 28, 1982. This paper is a revision of material that appeared in IBM Research Report RJ3342. It does not include all the material in the earlier report. It does contain an improvement of earlier results.

† IBM Research Laboratory, San Jose, California 95193.

agreement cannot be reached in t or fewer phases. Note that if we relax (I) slightly as in crusader agreement (Da), then we can obtain agreement within two phases.

The algorithms considered provide a method for a single processor to send a single value to all other processors. Generalizations to many processors sending values to each other will be obvious.

We assume some reliable means of communication by which any correct processor can send a message to any other correct processor. For example, this reliability might be achieved by sending duplicate messages along many paths in a network. In any case, for this paper, unless otherwise stated, we assume a completely connected, totally reliable communication network, and in counting the total number of messages sent, we ignore any duplication or repetition inherent in the communication medium. Note that we only count the messages sent by correct processors.

For algorithms using authentication, we assume a protocol that will prevent any processor from introducing a new value or message into the information exchange and claiming to have received it from another (DH), (RSA). In a typical authentication protocol (PSL), the transmitter appends a signature to the message to be sent. This signature contains a sample portion of the message encoded in such a way that any receiver can verify that the message is authentic and that it was sent by the sender, but no processor can forge the signature of another. Thus no processor can change the content of a message undetectably.

All previous algorithms for reaching Byzantine agreement are exponential in the number of messages ($O(n^t)$ where n is the number of processors and t is an upper bound on the number of undetected faulty processors). The new results presented here include algorithms polynomial in the number of bits exchanged, *using authentication*. If d is the number of phases and m is the total number of messages, then the algorithms previously presented used $d = t+1$ and $m = O(n^t)$.

Lynch and Fischer established a lower bound of $t+1$ for d, but their proof depended on disallowing any authentication protocol. Here we establish the same lower bound in a general context allowing authentication. Note, however, that our proof does not depend on the use of any particular authentication protocol. In fact it contains no reference to authentication or any other particular type of message.

We present an algorithm for Byzantine agreement with $d = t+1$ and $m = O(n^2)$, and a modification with $d = t+2$ and $m = O(nt)$. These algorithms are first presented in the context of a complete network and then generalized to arbitrary networks with sufficient connectivity. The total number of messages is on the order of the number of edges in the network, but the more general networks require more phases. Finally we present an algorithm that achieves the lower bound $t+1$ for number of phases and also requires only $O(nt)$ messages.

2. Histories. In order to give proofs of correctness and especially to establish lower bounds, we will describe the message related behavior of the collection of processors during the phases of information exchange as a single object of directed graphs called phases. We intend the notion of history to capture any synchronous information exchange behavior, including any number of authentication protocols and the exchange of arbitrary message types. The lower bound result of § 3 can be extended to asynchronous algorithms with a suitable generalization of the notion of phase.

A *phase* is a directed graph with nodes corresponding to processors and with labels on the edges. A label represents the information sent from a given processor to another during the given phase. We assume that when no message is sent there is no edge. An n processor *history* is a finite sequence of n node phases, with nodes

labelled by the names of the processors, together with a special initial phase called *phase* 0, such that phase 0 contains only a single inedge to one processor called the *sender.* (The assumption is that the inedge at phase 0 carries the value that the sender is to send.) Figures 1 and 2 represent histories with labels and phase 0 omitted.

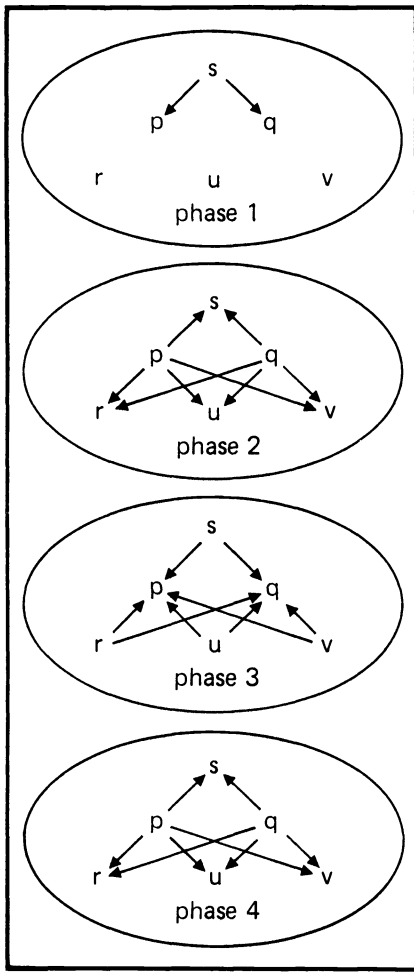


FIG. 1. *A six processor four phase history with edge labels and phase* 0 *omitted.*

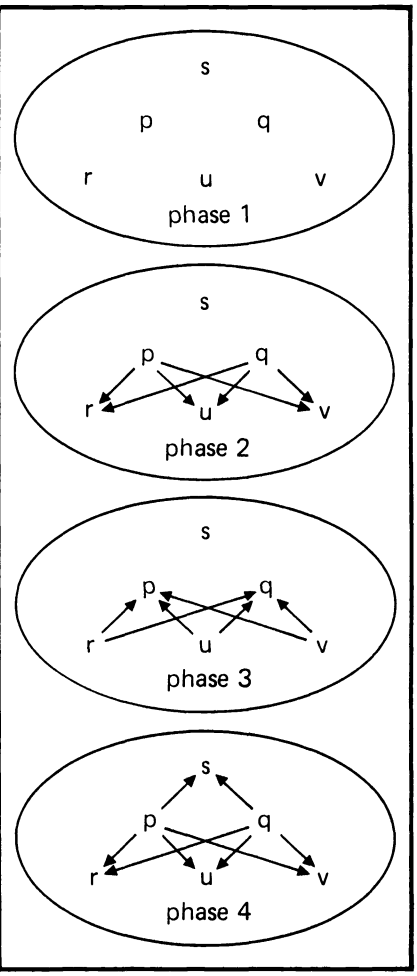


FIG. 2. *The result of hiding sender s at phase* 1.

A *subhistory* of a history H is a copy of H with some edges removed. For each history H and processor p there is a unique subhistory pH called the *subhistory according to p*, consisting of only the edges with target p. Thus, the subhistory according to the sender includes the value it is supposed to send even if it sends nothing.

An *agreement algorithm* on a class of histories C consists of a *correctness rule* (a function which given a subhistory according to p and an edge in a phase to be added to the history as the next phase, produces a possibly empty label for that edge) and a *decision function* (a function from subhistories according to processors of histories in C to the union of V with a symbol 0 representing "sender fault"). With respect

to a given correctness rule, a processor p is said to be *correct at* phase k if each edge from p in phase k has the label produced by the correctness rule operating on the subhistory according to p of the previous $k-1$ phases. A processor p is *correct* for history H if it is correct at each phase of H. Observe that the difference between Fig. 1 and Fig. 2 is that in the first, all processors are correct (assuming some appropriate correctness rule) while in the second, the sender is faulty. We call a history *t-faulty* (with respect to a correctness rule) if at most t of its processors are incorrect.

A correctness rule is actually a union of possibly distinct correctness rules, one for each processor. Likewise, the decision function is a union of individual decision functions.

An example of a simple correctness rule is the rule that each processor simply sign and relay (according to the authentication protocol) each incoming message of the previous phase to every other processor.

We say *Byzantine agreement can be achieved for n processors with at most t faults within d phases* if there is an agreement algorithm for the class C of n processor, t-faulty (with respect to the correctness rule of the algorithm), d phase histories so that the decision function F obeys the rules for Byzantine agreement:

(I) if p and q are correct for H in C then $FpH = FqH$, and

(II) if the sender is correct at the first phase of H and p is correct for H in C then $FpH = v$ where v is the sender's value.

Note that we do not define Byzantine agreement for $n < 3$ or for $t > n$. In the context of an authentication protocol, the class C of histories is assumed to be limited to those consistent with the semantics of authentication.

3. The lower bound result.

THEOREM 1 (LSP). *Byzantine agreement with authentication can be achieved for n processors with at most t faults within $t+1$ phases, assuming $n > t+1$.*

Proof. For the correctness rule, at phase i let each node sign and relay every incoming message with i different signatures to exactly those processors that have not already signed it. Note that messages are distinct even though they carry the same value, if they have travelled distinct paths. The function F, operating on a subhistory, will delete messages that do not conform to the correctness rule (including those with repeated signatures) and then extract every authenticated (from the sender) value from V carried by the remaining messages. If exactly one value v is extracted, then F will produce v as output; otherwise, F will produce 0.

Let H be a t-faulty history with $t+1$ phases, consistent with the semantics of authentication. At the end of phase t, each message carries t signatures (not counting that of the current recipient). Thus each value that appears in the correct messages of H will have been seen by some correct processor. Therefore, each correct processor will have the same set of extracted values after $t+1$ phases. □

The following lower bound result is the principal result of this section. It shows that the result of Theorem 1 is tight.

THEOREM 2. *Byzantine agreement cannot be achieved for n processors with at most t faults within t or fewer phases, provided $n > t+1$.*

The proof of Theorem 2 is inspired by, but a nontrivial generalization of, the proof given by Lynch and Fischer for the restricted case without authentication (FL). Lynch and Fischer used the $n > 3t$ result of (PSL) to show that any algorithm for Byzantine agreement must be uniform. Assuming uniformity, they established an equivalence relation on their version of t-faulty histories and obtained a contradiction by showing that too many histories were contained in a single equivalence class. Their

proof of this equivalence relied on the ability to preserve equivalence while changing one message at a time. Their proof of this ability, without using the uniformity assumption, is essentially the proof of the base case in the induction that follows.

Proof of Theorem 2. Assume that Byzantine agreement can be achieved for some $n > t+1$ within t phases. Let R be the correctness rule and let F be the decision function on subhistories such that $\langle R, F\rangle$ achieves Byzantine agreement on n processor, depth t, t-faulty histories.

Let C be the class of n processor, depth t, t-faulty histories that have a *critical sequence* such that all incorrect processors appear on the sequence and any incorrect node appears at or after the level corresponding to the order its label appears on the sequence. The class C contains histories that exhibit *serial faultiness*, in the sense that the set of faulty processors is allowed to increase by at most one processor per phase, starting with no faults before phase 1, and once allowed in the set these faulty processors may exhibit their faultiness at any node corresponding to a phase at or after their entry. Note that any nodes corresponding to such a faulty processor may be correct.

Define an equivalence relation on histories in C by saying H is *equivalent* to H' if, whenever p is correct for H and q is correct for H', then $FpH' = FpH$. Note that C includes histories in which all processors behave correctly. Since we assume V has more than one value, this means that there must be histories in C that are not equivalent. But, as we will show, C is a single equivalence class. Under an appropriate definition of $\langle R, F\rangle$, both Fig. 1 and Fig. 2 could describe histories from the set C. However, in Fig. 2 the result of the algorithm must be independent of any information from the sender since the sender sends nothing. This fact is the key idea behind the contradiction we obtain.

We say that a processor is *hidden* at phase k if it has no outedges at k or any later phase. We will also refer to the node at phase k as hidden if the processor is. In particular we will show by induction on the phase k that, if r is a node representing a processor at phase k of history H in C, then:

(a) there is a history H' in C, equivalent to H, identical to H through phase k except for outedges of r, with r correct and all processors correct after phase k; and

(b) if all other nodes at phase k are correct, then there is a history H' in C, equivalent to H, identical to H through phase k except for outedges of r, with r hidden and all other processors correct after phase k.

Note that if a processor labels a hidden node, then changing the information on its inedge cannot affect the subhistory according to any other processor. In Fig. 2 the sender is hidden at phase 1.

In short we will show by induction that we can correct a node at any phase or hide a node if all other nodes at its phase are correct, and that the resulting history will be in C and equivalent to the one from which we started, while all changes will be to the outedges of the particular node and to edges at later phases. Thus we will have shown that every history in C is equivalent to any history in which the root is hidden and all other processors are correct.

Case 1. *Let* $k = t$.

(a) Let r be an incorrect node at phase k of history H in C. If we correct the outedges of r one at a time, then for each individual change there is a processor correct for H that sees the same subhistory after the change as before. Thus each individual change preserves equivalence with H. Since we cannot make any correct node incorrect, each individual change preserves membership in C. Changes are only

made to the outedges of r. The final result H' has r correct and all processors trivially correct after k.

(b) Let r be a node at phase k in history H and C and let all other nodes at phase k be correct. Proceeding as in (a), we remove the outedges of r, one at a time. Here we may change r from correct to incorrect but since there were no other incorrect nodes at phase k we could replace the kth entry in the critical sequence by the label of r, preserving membership in C. The rest of the argument is the same as that for (a).

Case 2. *Assume the induction hypotheses* (a) *and* (b) *for all phases after* k.

(a) Let r be an incorrect node at phase k of a history H in C. The following steps will preserve membership in C and equivalence to H and change only outedges of r and edges at later phases.

1. Correct all nodes after phase k (induction hypothesis (a)).
2. While incorrect outedges of r remain,
   replace position $k+1$ in the critical sequence by s, a target of an incorrect outedge e from r;
   hide s at phase $k+1$ (induction hypothesis (b)) ;
   correct e (some correct processor will see the same subhistories both before and after the change);
   correct all nodes at phase $k+1$ (induction hypothesis (a)).
   End of while.

The final result H' will have r and all processors after phase k correct.

(b) Assume all processors correct at phase k and let r be a node at phase k. The following steps will preserve membership in C and equivalence to H and change only outedges of r and edges at later phases.

1. Correct all nodes at phase $k+1$ (induction hypothesis (a)).
2. Replace the kth position in the critical sequence by the label of r.
3. While outedges of r remain,
   replace position $k+1$ in the critical sequence by s, a target of an outedge e from r;
   hide s at phase $k+1$ (induction hypothesis (b));
   remove e (some correct processor will see the same subhistories both before and after the change);
   correct all processors after phase k (induction hypothesis (a)).
   End of while.
4. Hide the processor labelling r at phase $k+1$ (induction hypothesis (b)). The final result H' will have r hidden at phase k and all other processors after phase k correct.

This completes the proof of Theorem 2. □

Remark. Whenever it is defined, Byzantine agreement can be achieved for n processors within $n-1$ phases. Thus the provision $n > t+1$ is necessary for the lower bound of Theorem 2.

4. Polynomial algorithms using authentication. As mentioned in the introduction, we assume the existence of some authentication technique that prevents faulty processors from undetectably changing the content of messages.

For purposes of counting messages we supply the following specific syntax for the labels on the edges of directed graphs called phases.

(1) The set of values V is contained in the set of *atomic messages*.

(2) A *label* is either an atomic message (an authentication) or a sequence of labels.

(3) An *authentication* is a label of the form

$$(\text{label } a)\, p,$$

where p is the name of a processor and label a is a label.

(4) A *sequence of labels* is a label of the form

$$\text{label } a, \text{label } b,$$

where label a and label b are labels.

Note that $(a, b, c)p$ is not the same label as $(a)p$, $b(p)$, $(c)p$.

A label a is *part* of label b if either:

(i) $a = b$;

(ii) there is a label c and a process or p such that a is part of c and $b = (c)p$; or

(iii) there are labels c and d such that $b = c, d$ and a is part of c or d.

A *message* is a label with no commas.

Thus, at any phase any processor can send any message to any other processor, except that no processor can alter an authenticated message received at a previous phase and forward it as an authenticated message at the next phase, nor can any processor pretend to have received an authenticated message it did not receive and forward that as an authenticated message. In the rest of this paper, attention will be restricted to histories consistent with the semantics of authentication. In particular, if $(a)q$ is part of a label on an edge from processor p then either $p = q$ or $(a)q$ appears as part of a label on an inedge to p in a previous phase.

The basic idea behind the following two algorithms is to minimize the number of messages on each edge by restricting the cases in which a processor must relay a message. In the proof of Theorem 1, we assume a complete graph, so that when a correctly authenticated value is revealed to a correct processor, all correct processors will have it at the next phase. For Theorem 3 we restrict the number of values about which a processor must relay information. For Theorems 4 and 5 we restrict the paths over which messages travel so that when a correct processor receives a correctly authenticated value, other correct processors will receive it within some constant number of phases. Finally, for Theorem 6, we restrict the number of processors that are required to relay information. In this case when a correct relay processor receives a correctly authenticated value, the others will receive it one phase later, but correct processors that are not relay processors may receive the value long before it is known to the others.

Let e be the number of edges in the directed graph that has an edge between two processors exactly when our algorithm may require some message along that edge.

THEOREM 3. *Byzantine agreement can be achieved for n processors with at most t faults within $t+1$ phases using at most $O(e) = O(n^2)$ messages.*

Proof. Our correctness rule will be a restriction of that of the proof of Theorem 1 so that no processor relays more than two messages to any other, regardless of the number of messages received or the number of distinct paths incoming messages may have travelled. At the beginning of phase $i+1$, each processor totally (lexicographically) orders all messages received during the previous phase, *discarding* messages that are not of the form $(\cdots((v)p_1)p_2\cdots)p_i$ where v is a value not seen before, p_1 is the signature of the sender, and all signatures are distinct. If a message carrying value v is not discarded, then the processor is said to *extract* v. If the processor has not yet relayed any messages, then it relays the first two with distinct values (the first one if there are not two distinct values). If the processor has relayed only one message

during all previous phases, then it relays the first of its messages. The relay process consists of signing the message and forwarding it to all those whose signatures do not already appear in the message. A processor relays a value only if it is either the first or the second different value extracted. Once a processor has relayed two distinct values, it stops processing messages for the algorithm and at the end it will decide "sender fault," i.e. the decision function F from the proof of Theorem 1 will produce 0 for this processor. If it gets through $t+1$ phases without extracting any value, F will also produce 0; but if it has extracted exactly one value v, then F will produce v.

Each correct processor sends at most two messages over each edge. Thus, the total number of messages sent by correct processors is bounded by twice the number of edges, e.

If the sender correctly sends $(v)s$ to each other processor, then authentication prevents faulty processors from importing more values, so F will produce v for each correct processor. If at phase $t+1$ a correct processor receives and does not discard a message of the form $(\cdots((v)p_1)p_2\cdots)p_{t+1}$, then the first t processors on the list of signatures must be faulty, so the last one must be correct and all other correct processors have simultaneously received the same message. If, at the end of $t+1$ phases, a correct processor has extracted only one value, then each correct processor has extracted only that value, and the decision function F will produce the same value for each correct processor. If any correct processor extracts more than one value, then all will. Consequently, although the sets of extracted values may not agree, they yield sufficient information to reach Byzantine agreement. □

If we restrict the number of possible edges to $e = O(nt)$ by restricting the edges available for transmission we can reduce the number of messages. First we show a simple way to achieve this reduction that requires one extra phase.

THEOREM 4. *Byzantine agreement can be achieved for n processors with at most t faults within $t+2$ phases using at most $O(nt)$ messages.*

Proof. We further restrict the correctness rule of the proof of Theorem 3 by arbitrarily choosing $t+1$ processors to be relay processors and requiring any nonrelay processor to send messages only to relay processors (the sender is not a relay processor). Now the number of messages is $O(nt)$. If a correct processor extracts a new value by phase $t+2$, then some correct processor extracted that value by phase t, so some correct relay processor extracted it by phase $t+1$. Thus by phase $t+2$ every correct processor will have extracted either that value or two others from that relay processor. If any correct processor extracts only one value by phase $t+2$, then every correct processor must have extracted only that value. As in the proof of Theorem 3, although the sets of extracted values may not agree, they yield sufficient information to reach Byzantine agreement. □.

The restricted network used in the proof of Theorem 4 is a $t+1$ connected graph. It is straightforward to show that the graph must be at least $t+1$ connected in order to reach Byzantine agreement (Da), (Db). In (LPS) $t+1$ connectivity was shown to be sufficient using an exponential number of messages. Here using methods similar to those of (LPS) we generalize the algorithm of the proof of Theorem 4 to show that $t+1$ connectivity is sufficient even for a polynomial number of messages.

The *diameter* of a graph is the least upper bound of the lengths of shortest paths between pairs of vertices, where by length we mean the number of edges. If a graph is k connected, then there are at least k vertex disjoint paths between any pair of vertices. The *k-diameter* of a graph is the least upper bound of the lengths of the k shortest vertex disjoint paths between pairs of vertices.

THEOREM 5. *If d is the $(t+1)$-diameter of a $(t+1)$-connected network of n processors with at most t faults, then Byzantine agreement can be achieved within $t+d$ phases using at most $O(e)$ messages.*

Proof. We use the correctness rule and the decision function of Theorem 3, restricted of course so that only available edges of the graph are used for messages. If a processor extracts a new value at phase $t+i$, then some correct processor has extracted the value by phase t, so each correct processor will have extracted it (or two others) by phase $t+d$. Again, each edge carries at most two messages, so the total number of messages is $O(e)$. □

Now we return to our assumption of a complete network and present our best algorithm for Byzantine agreement with authentication.

THEOREM 6. *Byzantine agreement can be achieved on a complete network in $t+1$ phases with $O(nt)$ messages.*

Proof. If $n<2t+1$ then $O(n^2)=O(nt)$ so we are done by Theorem 3. Assume $n>2t$. We choose $2t+1$ processors including the sender to play active roles and let all the others be passive. The correctness rule for the active processors is that of Theorem 3, except that they must ignore all messages signed by passive processors. The passive processors are not to send messages. The decision function for active processors is that of Theorem 3. Thus they reach Byzantine agreement among themselves by phase $t+1$.

Passive processors modify the decision function so that it also counts the number of active processors that have sent more than one message, producing 0 if this number is at least $t+1$. Passive processors discard the same messages as active processors, but they extract only values that have been signed (in the total collection of messages received) by at least $t+1$ distinct active processors. Note that if a passive processor receives a message at phase $t+1$ and does not discard it, then it will extract the value carried by that message because the $t+1$ signatures must occur in that message. However, if a message is received at an earlier phase and not discarded, its value may not be extracted until later confirming messages supply $t+1$ distinct signatures.

If the correct active processors never extract a value, then the correct passive processors will never extract one because there are at most t faulty active processors. If any correct active processor extracts a new value then some correct active processor extracts that value by phase t and relays it to all. If a processor extracts a value at phase t, then the message it relays will contain the signature of $t+1$ active processors; otherwise, if it extracts the value by phase $t-1$, then at least t other correct active processors will have extracted the value by phase t. Thus if any correct active processor extracts only one value, then each correct active processor extracts only that value and each correct passive processor will be able to extract that and only that value by phase $t+1$.

If every correct active processor has extracted more than one value by phase t, then the passive processors will have received more than one message from $t+1$ active processors by phase $t+1$.

The only difficult case is that in which each active processor extracts more than one value by phase $t+1$, but some correct active processor has not extracted more than one value by the end of phase t. It remains to show that in this case, the passive processors will be able to extract more than one value.

In this case, no correct active processor can extract more than one value by phase $t-1$. At least two values are extracted by phase t (possibly by different processors). If the two values are extracted at phase t, then the passive processors will extract them at phase $t+1$. But at least one of them is extracted at phase t. Moreover, if

some correct active processor has extracted v by phase $t-1$, then each correct active processor has extracted v or nothing by phase $t-1$. Thus, if the first value is extracted at phase t or if all correct active processors extract that first value by $t-1$, then the passive processors will be able to extract two values.

This leaves the case that v is extracted by some but not all correct active processors by phase $t-1$. Each of the correct active processors that extracted v relays it to the passive processors by phase t. If one of the processors that did not extract v by phase $t-1$ extracts two other values at phase t, the passive processors can extract those two values. Otherwise, each of the processors that did not extract v by phase $t-1$ will extract v at phase t and relay it to the passive processors. In any case the passive processors will extract at least two values. □

5. Conclusion. The lower bound of $t+1$ phases with authentication means that we must look elsewhere to achieve Byzantine agreement quickly. In fact we must relax some requirement because the lower bound of Theorem 2 applies no matter what kind of message we send. Although the proof is given in the context of synchronous phases, any purported asynchronous algorithm for Byzantine agreement would certainly be imbeddable within the synchronous phase context by simply imposing the phases on its behavior.

One possibility would be to look at algorithms that *probably* achieve Byzantine agreement. In a probabilistic context, if we had a realistic upper bound t on the number of possible faults, then we would likely also have information on the probability of exactly t faults, exactly $t-1$ faults, etc.

While they do not reduce the minimum number of phases required, our algorithms do reduce the total number of messages required for Byzantine agreement from exponential to polynomial in the number of processors or in the number of bits exchanged by correct processors. It would be useful to find algorithms that stop after a smaller number of phases whenever possible.

We have not established a tight lower bound on the number of messages required in the worst case. In (DR) lower bounds on the number of messages and the number of signatures that must be exchanged in order to obtain Byzantine agreement are obtained. The algorithm in (DR) requires fewer messages than ours but uses more phases.

Acknowledgments. The authors thank Nancy Lynch for helpful suggestions about this manuscript. The proof of Theorem 6 uses a suggestion of Lynch made in private correspondence with respect to a different problem. Subsequent to the completion of the proof of Theorem 2 in its present form, the authors received a private communication from Michael Merritt containing a somewhat similar proof of this result.

REFERENCES

(DH) W. DIFFIE AND M. HELLMAN, *New direction in cryptography*, IEEE Trans. Inform. Theory, IT-22 (1976), pp. 644–654.

(Da) D. DOLEV, *The Byzantine generals strike again*, J. Algorithms, 3 (1982), pp. 14–30.

(Db) ———, *Unanimity in an unknown and unreliable environment*, Proc. IEEE 22nd Symposium on Foundations of Computer Science, 1981, pp. 159–168.

(DR) D. DOLEV AND R. REISCHUK, *Bounds on information exchange for Byzantine agreement*, Proc., ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, Aug. 1982. See also IBM Research Report RJ3587 (1982).

(FL) M. FISCHER AND N. LYNCH, *A lower bound for the time to assure interactive consistency*, Inform. Proc. Letters, 14 (1982), pp. 183–186.

(L) L. LAMPORT, *Using time instead of timeout for fault-tolerant distributed systems*, Tech. Rep., Computer Science Laboratory, SRI International, June 1981.

(LSP) L. LAMPORT, R. SHOSTAK AND M. PEASE, *The Byzantine generals problem*, ACM Trans. Programming Languages and Systems, to appear.

(PSL) M. PEASE, R. SHOSTAK AND L. LAMPORT, *Reaching agreement in the presence of faults*, J. Assoc. Comput. Mach., 27 (1980), pp. 228–234.

(RSA) R. L. RIVEST, A. SHAMIR AND L. ADLEMAN, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0006 $01.25/0

ANALYSIS OF EARLY-INSERTION STANDARD COALESCED HASHING*

WEN-CHIN CHEN† AND JEFFREY SCOTT VITTER†‡

Abstract. This paper analyzes the *early-insertion* standard coalesced hashing method (EISCH), which is a variant of the standard coalesced hashing algorithm (SCH) described in [Knu73], [Vit80] and [Vit82b]. The analysis answers the open problem posed in [Vit80]. The number of probes per successful search in full tables is 5% better with EISCH than with SCH.

Key words. analysis of algorithms, hashing, coalesced hashing, early-insertion, data structures, average-case

1. Introduction. One of the well-known data structures for information storage and retrieval is *coalesced hashing*, which was introduced in [Wil59] and analyzed in [Vit80], [Vit82b], [Knu73] and [GK81]. We will assume that each package of information is stored in computer memory as a *record*. There is a special field in each record, called the *key*, that uniquely identifies it. The job of a searching algorithm is to take an input K and return the record (if any) that has K as its key.

For purposes of notation, we let M' denote the number of slots in the hash table. The first M slots, which serve as the range of the hash function, are called the *address region*; the remaining $M'-M$ slots make up the *cellar*. We assume that the pre-defined hash function

$$\text{(1)} \qquad hash: \{\text{all possible keys}\} \to \{1, 2, \cdots, M\}$$

assigns each record to its *hash address* in a random (uniform) manner. We say that a *collision* occurs when the hash address of a record is already occupied, and the record must be inserted elsewhere. The special case in which $M = M'$ and there is no cellar is called *standard coalesced hashing*.

The coalesced hashing method has the property that a record is not moved once it is inserted. The algorithm can be described as follows: Given a record with key K, the algorithm searches for it in the hash table, starting at its hash address $hash(K)$ and following the links in the chain. If the record is found, the search is successful; otherwise, the end of the chain is reached and the search is unsuccessful, in which case the record is inserted as follows: If position $hash(K)$ is empty, then the record is stored at that location; otherwise, the record is stored in the largest-numbered empty slot in the table and is linked into the chain that contains slot $hash(K)$ (at some point after slot $hash(K)$). There are two different ways to link that record into the chain. The conventional method is to link the record to the *end* of chain that contains slot $hash(K)$. The second method, which was named *early-insertion* by Gary Knott, inserts the record into the chain *immediately after* slot $hash(K)$ by rerouting pointers. Insertion can be done faster with the early-insertion method when it is known a priori that the record is not already present in the table, since it isn't necessary to search to the end of the chain.

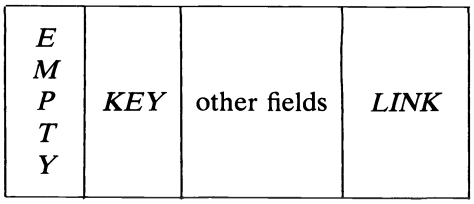
A formal description of the conventional insertion method appears below. Let us assume that each of the M' contiguous slots in the coalesced hash table has the

* Received by the editors March 10, 1982, and in revised form October 5, 1982. This research was supported in part by an IBM Research contract.

† Department of Computer Science, Brown University, Providence, Rhode Island 02912.

‡ The research of this author was supported in part by National Science Foundation grant MCS-81-05324.

following organization:



For each value of i between 1 and M', $EMPTY[i]$ is a one-bit field that denotes whether the ith slot is unused, $KEY[i]$ stores the key (if any), and $LINK[i]$ is either the index to the next spot in the chain or else the null value 0.

Algorithm C (*conventional coalesced hashing search and insertion*). This algorithm searches an M'-slot hash table, looking for a given key K. If the search is unsuccessful and the table is not full, then K is inserted.

The size of the address region is M; the hash function *hash* returns a value between 1 and M (inclusive). For convenience, we make use of slot 0, which is always empty. The global variable R is used to find an empty space whenever a collision must be stored in the table. Initially, the table is empty, and we have $R = M' + 1$; when an empty space is requested, R is decremented until one is found. We assume that the following initializations have been made before any searches or insertions are performed: $M \leftarrow \lceil \beta M' \rceil$, for some constant $0 < \beta \leqq 1$; $EMPTY[i] \leftarrow$ **true**, for all $0 \leqq i \leqq M'$; and $R \leftarrow M' + 1$.

C1. (Hash) Set $i \leftarrow hash(K)$. (Now $1 \leqq i \leqq M$.)

C2. (Is there a chain?) If $EMPTY[i]$, then go to step C6. (Otherwise, the ith slot is occupied, so we will look at the chain of records that starts there.)

C3. (Compare.) If $K = KEY[i]$, the algorithm terminates successfully.

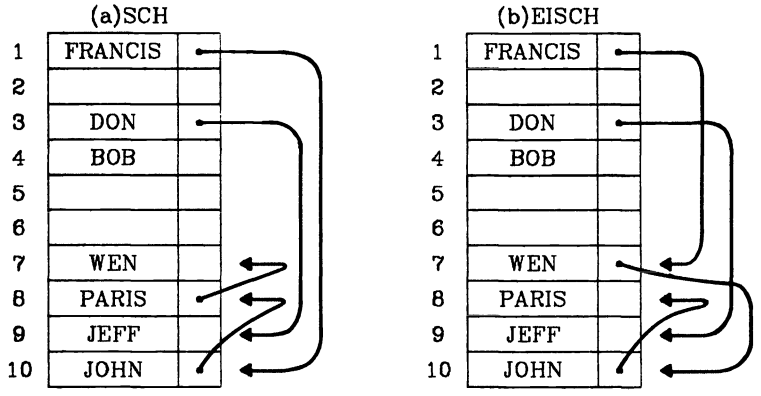
C4. (Advance to next record.) If $LINK[i] \neq 0$, then set $i \leftarrow LINK[i]$ and go back to step C3.

C5. (Find an empty slot. The search for K in the chain was unsuccessful, so we will try to find an empty table slot to store K.) Decrease R one or more times until $EMPTY[R]$ becomes **true**. If $R = 0$, then there are no more empty slots, and the algorithm terminates *with overflow*. Otherwise, append the Rth cell to the chain by setting $LINK[i] \leftarrow R$; then set $i \leftarrow R$.

C6. (Insert new record.) Set $EMPTY[i] \leftarrow$ **false**, $KEY[i] \leftarrow K$, $LINK[i] \leftarrow 0$, and initialize the other fields in the record.

The early-insertion method can be implemented by the following two modifications: First, we add the assignment "Set $j \leftarrow i$" at the end of step C2, so that j stores the hash address $hash(K)$. The second modification replaces the last sentence of step C5 by "Otherwise, link the Rth cell into the chain immediately after the hash address j by setting $LINK[R] \leftarrow LINK[j]$, $LINK[j] \leftarrow R$; then set $i \leftarrow R$."

An example of the two methods is given in Fig. 1. The record WEN collides with FRANCIS at slot 1. With the conventional insertion method pictured in Fig. 1(a), WEN is linked to the end of the chain containing FRANCIS, whereas in the early-insertion method in Fig. 1(b), WEN is inserted into the chain at the point between FRANCIS and JOHN. The average successful search time in Fig. 1(b) is slightly better than in Fig. 1(a), because inserting WEN immediately after FRANCIS (rather than at the end of the chain) reduces the search time for WEN from four probes to two



Keys:	FRANCIS	DON	JOHN	BOB	JEFF	PARIS	WEN
Hash Addresses:	1	3	1	4	3	10	1

ave. # probes per successful search: (a) 13/7 ≈ 1.86, (b) 12/7 ≈ 1.71.
ave. # probes per unsuccessful search: (a) 17/10 = 1.7, (b) 17/10 = 1.7.

FIG. 1. *Standard coalesced hashing,* $M' = M = 10$, $N = 7$. (a) SCH, (b) EISCH.

and increases the search time for JOHN from two probes to three. That results in a net decrease of one probe.

The analyses of coalesced hashing that have appeared in the literature have concentrated on the conventional method, in which a record is always linked to the end of the chain. Knuth [Knu73] analyzes the special case of standard coalesced hashing (SCH), in which $M = M'$ and there is no cellar. Vitter [Vit82b] analyzes the more general coalesced hashing method (CH), for which the cellar may be nonempty.

In this paper, we derive exact formulas for the average number of probes per search for the early-insertion standard coalesced hashing method (EISCH). This solves the open problem posed in [Vit80]. The average unsuccessful search times for the EISCH and SCH methods are the same, but the average successful search time is up to 5% better with EISCH than with SCH. The performance of early-insertion method when there is a cellar (EICH) is still unknown.

2. The main result. In this section we develop the probability model used in our analysis of early-insertion standard coalesced hashing EISCH and we state our main result, Theorem 1, which expresses the average number of probes per successful search for EISCH. We use the following parameters in our analysis:

N = the number of inserted records,

M' = the number of slots in the hash table,

$\alpha = N/M'$ = the load factor.

These quantities satisfy $0 \leqq N \leqq M'$ and $0 \leqq \alpha \leqq 1$. Since there is no cellar in the EISCH method, the address region size M is equal to the table size M'.

In the average-case analysis, we assume that an unsuccessful search can begin at any of the M slots in the address region with equal probability. This includes the special case of insertion. Similarly, each record in the hash table has the same chance of being the object of any given successful search. In other words, all searches and

insertions involve random keys. This model can be formalized by the following definitions.

DEFINITION 1. The sequence $a_1a_2\cdots a_N$, where $1 \leqq a_i \leqq M$, is called a *hash sequence*. Every such sequence represents the insertion of N records into a hash table of address size M; element a_j denotes the hash address of the jth inserted record (i.e., *hash*(key of jth record) $= a_j$).

In our analysis, we will use the number of probes per search, i.e., the number of slots traversed, as a measure of search performance.

DEFINITION 2. We let P'_N and P_N denote the random variables describing the number of probes in unsuccessful and successful searches in a SCH table containing N records. Similarly, we let $\bar{P}'_N$ and $\bar{P}_N$ be the random variables describing the number of probes in unsuccessful and successful searches in a EISCH table containing N records.

The sample space for P'_N and $\bar{P}'_N$ is

$$S' = \{[a_1, a_2, \cdots, a_N; a] \mid 1 \leqq a_j \leqq M, 1 \leqq a \leqq M\}, \tag{2}$$

where $a_1a_2\cdots a_N$ represents the hash sequence of the N inserted records, and a is the starting address of the unsuccessful search. The M^{N+1} elements in S' each have probability $1/M^{N+1}$. The values of P'_N and $\bar{P}'_N$ at sample point $[a_1, a_2, \cdots, a_N; a] \in S'$ are denoted by $P'_N[a_1, a_2, \cdots, a_N; a]$ and $\bar{P}'_N[a_1, a_2, \cdots, a_N; a]$.

Similarly, the sample space for P_N and $\bar{P}_N$ is

$$S = \{[a_1, a_2, \cdots, a_N; n] \mid 1 \leqq a_j \leqq M, 1 \leqq n \leqq N\}, \tag{3}$$

where $a_1a_2\cdots a_N$ represents the hash sequence of the N inserted records, and n specifies that the nth inserted record is the object of the successful search. The NM^N elements in S each have probability $1/NM^N$. The values of P_N and $\bar{P}_N$ at sample point $[a_1, a_2, \cdots, a_N; n] \in S$ are denoted by $P_N[a_1, a_2, \cdots, a_N; n]$ and $\bar{P}_N[a_1, a_2, \cdots a_N; n]$.

DEFINITION 3. For SCH, the average unsuccessful and successful search times are defined by

$$C'_N = \frac{1}{M^{N+1}} \sum_{\substack{M^N \text{ hash sequences} \\ 1 \leqq a \leqq M}} P'_N[a_1, a_2, \cdots, a_N; a], \tag{4}$$

$$C_N = \frac{1}{NM^N} \sum_{\substack{M^N \text{ hash sequences} \\ 1 \leqq n \leqq N}} P_N[a_1, a_2, \cdots, a_N; n]. \tag{5}$$

For EISCH, the average unsuccessful and successful search times are defined by

$$\bar{C}'_N = \frac{1}{M^{N+1}} \sum_{\substack{M^N \text{ hash sequences} \\ 1 \leqq a \leqq M}} \bar{P}'_N[a_1, a_2, \cdots, a_N; a], \tag{6}$$

$$\bar{C}_N = \frac{1}{NM^N} \sum_{\substack{M^N \text{ hash sequences} \\ 1 \leqq n \leqq N}} \bar{P}_N[a_1, a_2, \cdots, a_N; n]. \tag{7}$$

Knuth [Knu73] analyzes the standard coalesced hashing method SCH, and derives the following unsuccessful and successful search times, as functions of loading factor

$\alpha = N/M$:

$$C'_N = 1 + \frac{1}{4}\left(\left(1+\frac{2}{M}\right)^N - 1 - \frac{2N}{M}\right) \approx 1 + \frac{1}{4}(e^{2\alpha} - 1 - 2\alpha), \tag{8}$$

$$\begin{aligned} C_N &= 1 + \frac{1}{8}\frac{M}{N}\left(\left(1+\frac{2}{M}\right)^N - 1 - \frac{2N}{M}\right) + \frac{1}{4}\frac{N-1}{M} \\ &\approx 1 + \frac{1}{8\alpha}(e^{2\alpha} - 1 - 2\alpha) + \frac{1}{4}\alpha. \end{aligned} \tag{9}$$

In particular, we have $C'_M \approx 2.10$ and $C_M \approx 1.80$, when the table is full (i.e., when load factor $\alpha = 1$).

As Fig. 1 shows, the chains in SCH and EISCH contain the same elements, possibly in different orders. Since the average time for an unsuccessful search depends only on the length of the chains and not on the order of keys within the chains, the average unsuccessful search times C'_N and $\bar{C}'_N$ for SCH and EISCH are equal.

The following theorem is the main result of this paper. It gives the average successful search time for EISCH, using the probability model described in Definitions 1–3.

THEOREM 1. *In an M-slot early-insertion standard coalesced hash table containing N inserted records, the average number of probes in a random successful search is*

$$\bar{C}_N = \frac{M}{N}\left(1+\frac{1}{M}\right)^N - \frac{M}{N}. \tag{10}$$

In asymptotic form, this can be expressed as

$$\bar{C}_N \approx \frac{1}{\alpha}(e^{\alpha} - 1), \tag{11}$$

where $M, N \to \infty$ and $\alpha = N/M$ remains constant. The approximation error is roughly $e^{\alpha}/(2M)$.

The graphs in Fig. 2 reflect the fact that EISCH is slightly better than SCH. For example, when the table is full (i.e., load factor $\alpha = 1$), we have $C_N \approx 1.80$ and $\bar{C}_N \approx 1.72$, so EISCH is about 5% faster. The difference between the expected number of probes per successful search for SCH and EISCH is not significant when α is small. When the table is half full (i.e., load factor $\alpha = .5$), we have $C_N \approx 1.31$ and $\bar{C}_N \approx 1.30$, which is only a 0.5% improvement. One reason for this is that search times improve only when searching for records contained in chains of length greater than 3; when the load factor is small, the chains are usually short.

3. Proof of the theorem. The derivation of the successful search time for EISCH is more difficult than the analysis of SCH for the following reason: Any coalesced hash table has the property that the hash address of the kth record in a chain ($k > 1$) must be the location of one of the $k-1$ predecessors in the chain. In a random SCH table, each of the $k-1$ locations can be the hash address of the kth record with equal probability $1/(k-1)$. However, that is not true in a random EISCH table, as Fig. 3 illustrates. The hash address of the fourth ($k = 4$) record in a random chain of length 4 is the location of the first record in the chain with probability 2/6, it is the location of the second record with probability 1/6, and it is the location of the third record

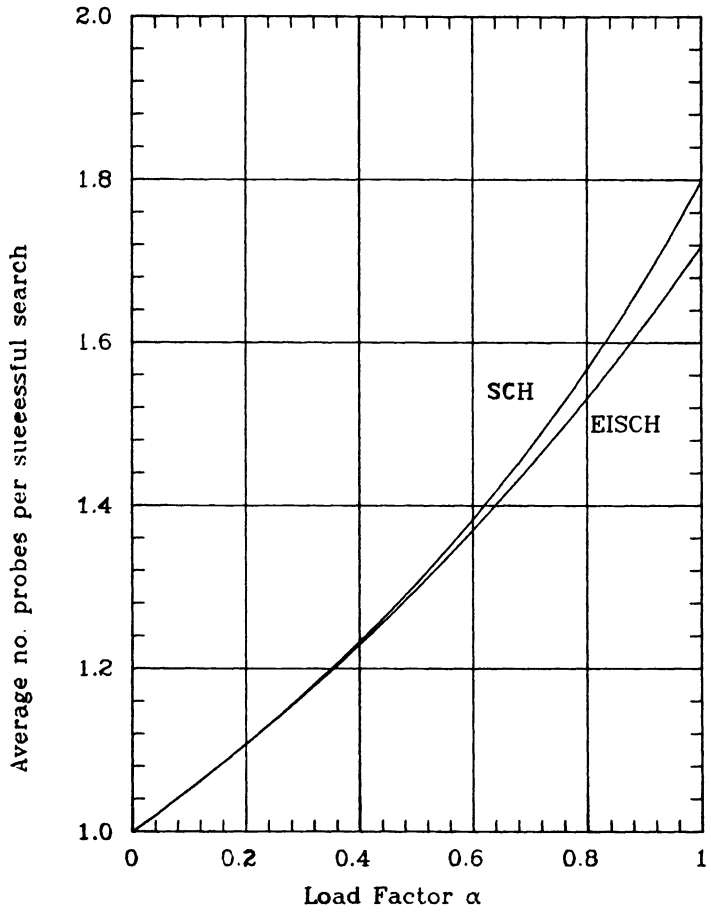


FIG. 2. *The average number of probes per successful search for* EISCH *and* SCH.

with probability 3/6. Our analysis is more difficult because it must take this into account.

For notational simplicity, the parameter M, which denotes the number of slots in the hash table, will not be written, but rather shall be implicit in all the terms defined in this section. In order to prove Theorem 1, let us start with some definitions.

DEFINITION 4. We let A_N denote the quantity $NM^N\bar{C}_N$, where $\bar{C}_N$ is the average successful search time defined in Definition 3.

DEFINITION 5. For $1 \leqq i \leqq j \leqq l$, we let $c_N(l, i, j)$ denote the number of chains among all the M^N hash tables that satisfy the following two properties:

(1) the length of the chain is l;

(2) the hash address of the jth record in the chain is the location of one of the first i records in the chain.

We should note that $c_N(l, i, j) = 0$ for $l > N$. For $j = i$ or $j = i + 1$, the term $c_N(l, i, j)$ is equal to the total number of chains of length l among all the M^N hash tables, since every chain has the property that the hash address of the jth record in the chain ($j > 1$) must be the location of a preceding record.

From the above remarks we have the following lemma.

Sequence of hash addresses	Order of the keys in the chain	Key that occupies the hash address of the last key in the chain	Relative position in the chain of that key
1111	adcb	a	1
1141	adbc	b	3
1144	abdc	b	2
1114	acbd	b	3
1113	acdb	a	1
1143	abcd	c	3

FIG. 3. *The keys a, b, c and d are inserted (in that order) into an* EISCH *table containing* $M' = 4$ *slots. This table describes all six possible chains of length* 4 *in an* EISCH *table in which the keys a, b, c and d are stored in locations* 1, 4, 3 *and* 2, *respectively. The last column shows that the hash address of the last record in the chain is more likely to be the position of the third record in the chain rather than that of the second or the first.*

LEMMA 1. *The summation* $\sum_{1\leq l\leq N, 1\leq i\leq l} c_N(l, i, i)$ *is equal to* NM^N.

The following lemma expresses the quantity A_N we want to evaluate in terms of $c_N(l, i, j)$.

LEMMA 2. *The term* A_N *is equal to* $NM^N + B_N$, *where* $B_N = \sum_{1\leq l\leq N, 1\leq i<j\leq l} c_N(l, i, j)$.

Proof. First we will show that $A_N = \sum_{1\leq l\leq N, 1\leq i\leq j\leq l} c_N(l, i, j)$. A search for the jth record in a chain of length l requires $j-i+1$ probes if the hash address of the jth record is the location of the ith record of the chain. Thus, searching for the jth record contributes $j-i+1$ to A_N, which, by definition, is the sum of the contributions of the NM^N searches among all the M^N hash tables. The search for the jth record contributes 1 to each of the following $j-i+1$ terms: $c_N(l, i, j), c_N(l, i+1, j), \cdots, c_N(l, j, j)$. Hence, we have

$$A_N = \sum_{\substack{1\leq l\leq N \\ 1\leq i\leq j\leq l}} c_N(l, i, j) = \sum_{\substack{1\leq l\leq N \\ 1\leq i=j\leq l}} c_N(l, i, j) + \sum_{\substack{1\leq l\leq N \\ 1\leq i<j\leq l}} c_N(l, i, j).$$

The first summation is equal to NM^N, by Lemma 1. The second summation is the definition of B_N. □

To evaluate B_N, we need the following recurrence.

LEMMA 3. *The term* $c_N(l, i, j)$ *defined in Definition* 5 *satisfies the recurrence*

$$\begin{aligned} c_{N+1}(l, i, j) &= (M-l)c_N(l, i, j) + (i-1)c_N(l-1, i-1, j-1) \\ &\quad + (j-i-1)c_N(l-1, i, j-1) \\ &\quad + (l-j)c_N(l-1, i, j) + \delta_{j=i+1}c_N(l-1, i, i) \end{aligned} \tag{12}$$

for $1\leq N\leq M-1$, $1\leq i<j\leq l\leq N+1$. *The notation* δ_R *denotes* 1 *if the relation* R *is true and* 0 *otherwise.*

Proof. Let's consider the M^N distinct hash sequences for N inserted records. The only chains that can contribute to $c_{N+1}(l, i, j)$ after the $(N+1)$st insertion are chains that contribute to $c_N(l, i, j)$, $c_N(l-1, i-1, j-1)$, $c_N(l-1, i, j-1)$, $c_N(l-1, i, j)$ and $c_N(l-1, i, i)$.

When $l<N+1$, a chain that contributes to $c_N(l, i, j)$ after N insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted record is the location of one of the $M-l$ records outside the chain. This accounts for the $(M-l)c_N(l, i, j)$ term.

When $i>1$, a chain that contributes to $c_N(l-1, i-1, j-1)$ after N insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted

record is the location of one of the first $i-1$ records in the chain. This accounts for the $(i-1)c_N(l-1, i-1, j-1)$ term.

When $j > i+1$, a chain that contributes to $c_N(l-1, i, j-1)$ after N insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted record is the location of one of the records between the ith and $(j-2)$nd records, inclusively, in the chain. This accounts for the $(j-i-1)c_N(l-1, i, j-1)$ term.

When $j < l$, a chain that contributes to $c_N(l-1, i, j)$ after N insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted record is the location of one of that last $l-j$ records in the chain. This accounts for the $(l-j)c_N(l-1, i, j)$ term.

When $j = i+1$, any chain of length $l-1$ after N insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted record is the location of the ith record in the chain. This accounts for the $\delta_{j=i+1}(l-1, i, i)$ term. □

LEMMA 4. *The term B_N defined in Lemma 3 is equal to $M(M+1)^N - (M+N)M^N$, for $1 \leqq N \leqq M$.*

Proof. Substituting Lemma 3 into B_N, we have, for $1 \leqq N \leqq M-1$,

$$
\begin{aligned}
B_{N+1} &= \sum_{\substack{1\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} c_{N+1}(l, i, j) \\
&= \sum_{\substack{1\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (M-l)c_N(l, i, j) + \sum_{\substack{2\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (i-1)c_N(l-1, i-1, j-1) \\
&\quad + \sum_{\substack{2\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (j-i-1)c_N(l-1, i, j-1) + \sum_{\substack{2\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (l-j)c_N(l-1, i, j) \\
&\quad + \sum_{\substack{2\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} \delta_{j=i+1}c_N(l-1, i, i) \\
&= \sum_{\substack{1\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (M-l)c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 0\leqq i<j\leqq l}} ic_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i\leqq j\leqq l}} (j-i)c_N(l, i, j) \\
&\quad + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l+1}} (l+1-j)c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i\leqq j\leqq l}} \delta_{j=i}c_N(l, i, i) \\
&= \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} (M-l)c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} ic_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} (j-i)c_N(l, i, j) \\
&\quad + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} (l+1-j)c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i=j\leqq l}} c_N(l, i, i) \\
&= (M+1) \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i\leqq l}} c_N(l, i, i) \\
&= (M+1)B_N + NM^N.
\end{aligned}
$$

The last step follows from the definition of B_N and from Lemma 1. By telescoping and by the fact that $B_1 = 0$, we get

$$B_N = M(M+1)^N - (M+N)M^N. \qquad \square$$

Now, we are finally ready to prove Theorem 1. Combining Lemma 2 and Lemma 4, we have

$$A_N = NM^N + B_N = M(M+1)^N - M^{N+1}.$$

Thus, we get

$$\bar{C}_N = \frac{A_N}{NM^N} = \frac{M}{N}\left(1+\frac{1}{M}\right)^N - \frac{M}{N}. \tag{13}$$

We can express this in asymptotic form as

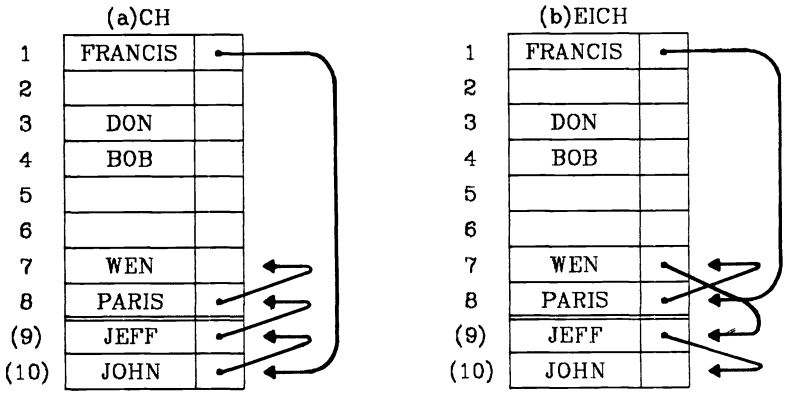
$$\bar{C}_N \approx \frac{1}{\alpha}(e^\alpha - 1), \tag{14}$$

where $N, M \to \infty$ and $\alpha = N/M$ remains constant in the range $0 \leqq \alpha \leqq 1$. The error term is approximately $e^\alpha/(2M)$.

4. Conclusions and open problems. We have analyzed the early-insertion standard coalesced hashing method (EISCH) and have shown that this method is slightly better than the standard coalesced hashing method (SCH). The average unsuccessful search times for these two methods are the same, but the average successful search time for EISCH is 5% better when the table is full.

Some interesting open problems concerning early-insertion remain unsolved. The early-insertion method can be used when there is a cellar and $M < M'$. We call this generalization EICH. The search performance of EICH is still unanalyzed. However, we conjecture that EICH is inferior to the coalesced hashing method (CH), since in EICH a chain's records that are stored in the cellar come at the *end* of the chain, whereas in CH they come *immediately* after the first record in the chain. In Fig. 4(b), the insertion of WEN causes both the cellar records JOHN and JEFF to move one link further from their hash addresses. That doesn't happen in Fig. 4(a).

Many hashing applications require that certain records be inserted and then later deleted. The best deletion algorithms are the ones that *preserve randomness*, because deleting a record is in some sense like never having inserted it. In particular, the formulas for the average search times after N random insertions intermixed with d deletions are the same as the formulas for the average search times after $N-d$ random insertions. The formal notion of what it means to preserve randomness is defined in [Vit82a].



Keys:	FRANCIS	DON	JOHN	BOB	JEFF	PARIS	WEN
Hash Addresses:	1	3	1	4	1	1	8

ave. # probes per successful search: (a) 14/7 ≈ 2.0, (b) 16/7 ≈ 2.286.
ave. # probes per unsuccessful search: (a) 13/8 = 1.625, (b) 17/8 = 2.125.

FIG. 4. *Coalesced hashing, $M' = 10$, $M = 8$, $N = 7$.* (a) CH, (b) EICH.

A deletion algorithm for coalesced hashing is given [Vit82a] and shown to preserve randomness for standard coalesced hashing (SCH). To delete a record, the algorithm removes it from the table, and then it repeatedly reinserts the record in the remainder of the chain, if any. There is a modification of this deletion algorithm that performs the reinsertions using the early-insertion idea. It does not preserve randomness for SCH, but it may possibly be "better-than-random." An interesting question is how this modified deletion algorithm (which uses early-insertion) affects the search times for EISCH. It does not preserve randomness, but it may possibly be better-than-random. There is currently no known deletion algorithm that preserves randomness for the EISCH, EICH and CH methods.

Addendum. Since the time this article was submitted, the authors have solved the first open problem listed in § 4. The analysis of the search time of early-insertion coalesced hashing for the general case (when there is a cellar) appears in *Analysis of some new variants of coalesced hashing*, Department of Computer Science, Brown University, Providence, RI, Technical Report No. CS-82-18, June 1982. That report also describes and analyzes a new method called varied-insertion coalesced hashing that performs better than both the unmodified method and the early-insertion method.

REFERENCES

[GK81] D. H. GREENE AND D. E. KNUTH, *Mathematics for the Analysis of Algorithms.* Birkhauser, Boston, 1981.

[Knu73] D. E. KNUTH, *The Art of Computer Programming. Volume* 3: *Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[Vit80] J. S. VITTER, *Analysis of coalesced hashing*, PhD dissertation, Department of Computer Science, STAN-CS-80-817 Tech. Rep. Stanford University, Stanford, CA, August 1980.

[Vit82a] ———, *Deletion algorithms for hashing that preserve randomness.* J. Algorithms, 3 (1982), pp. 261–275.

[Vit82b] ———, *Analysis of the search performance of coalesced hashing*, J. Assoc. Comput. Mach., 30 (April 1983).

[Wil59] F. A. WILLIAMS, *Handling identifiers as internal symbols in language processors*, Comm. ACM, 2 (1959), pp. 21–24.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0007 $01.25/0

VARIETIES OF "IF-THEN-ELSE"*

STEPHEN L. BLOOM† AND RALPH TINDELL‡

Abstract. Four classes of algebras are considered. The algebras in each class contain functions whose behavior models a version of the "if-then-else" instruction. In one version, for example, the algebras contain a function κ of four arguments such that $\kappa(x, y, u, v) = u$ if $x = y$ and $\kappa(x, y, u, v) = v$ if $x \neq y$. None of the considered classes is an equational class, but equational axioms are found for each class such that an equation is valid in the class iff it is derivable in standard equational logic from the axioms.

Key words. "if-then-else", equational logic, control flow

Introduction. The "if-then-else" instruction occurs in most high level programming languages and has been used in many theoretical studies of various aspects of computation. We mention, only as a small sampling, the papers by Arnold [1], Courcelle [4], Courcelle and Nivat [5], Cousineau and Nivat [6], Manna and Vuillemin [8], and Wagner, Thatcher, and Wright [12].

However, the exact formulation of "if-then-else" has not been uniform. In [4] for example, the construction takes the form of a function of three arguments: say $\kappa(p, x, y) = x$ iff p is true and y if p is false. In [12] on the other hand, "if-then-else" appears as a function of four arguments: $\kappa(x, y, u, v) = u$ if $x = y$; and v otherwise. Furthermore, it is sometimes desirable to allow the arguments to be "undefined," so that for example, $\kappa(p, x, y)$ would be undefined if p is undefined.

In at least two of the above places, Backus [2] and most explicitly in McCarthy [9], it is noted that certain equations between "if-then-else" expressions are valid: for example, the "redundancy law"

$$\text{if } p \text{ then (if } p \text{ then } u, \text{ else } v) \text{ else } w$$
$$= \text{if } p \text{ then } u \text{ else } w.$$

This observation leads naturally to the question: what are all of the valid identities between "if-then-else" expressions?

McCarthy [9] answered this question for a formalization of "if-then-else" as a function taking three arguments. His answer was framed in a syntactic way. He found *normal forms* for his "if-then-else" expressions, and gave axioms which were strong enough so that each expression was provably equal to one in normal form. Furthermore, his "if-then-else" expressions are two-sorted: Boolean and "general". Sethi [11] gave an algorithm for determining the validity of equations between "if-then-else" expressions with equality tests. The algorithm uses both syntactic and semantic methods.

In this paper, we define four formalizations of "if-then-else". In each such formalization, a class K of algebras is defined, each containing a function whose behavior models the version of "if-then-else" under consideration. We then find a set of equational axioms such that an equation $t \approx t'$ (between K-terms) is valid in K iff $t \approx t'$ is provable using standard equational logic from our axioms. All of our

* Received by the editors May 22, 1981, and in revised form September 10, 1982.

† Mathematical Sciences Department, IBM T. J. Watson Research Center, Yorktown Heights, New York 10598. On leave from the Department of Pure and Applied Mathematics, Stevens Institute of Technology, Hoboken, New Jersey 07030.

‡ Department of Pure and Applied Mathematics, Stevens Institute of Technology, Hoboken, New Jersey 07030.

formalizations use one-sorted algebras, as will become clear from the following summary.

In § 1, we start from some equational class $K = \mathrm{Mod}(\mathrm{EQ})$ of Σ-algebras (for some ranked set Σ). To each algebra A in K we adjoin a function $\kappa: A^4 \to A$ which satisfies

$$\kappa(x, y, u, v) = \begin{cases} u & \text{if } x = y, \\ v & \text{if } x \neq y, \end{cases}$$

obtaining a Σ^+-algebra A^+, where $\Sigma^+ = \Sigma \cup \{\kappa\}$. We then find a set of axioms Ax such that an equation between Σ^+-terms is valid in the class $\{A^+ : A \in K\}$ iff the equation is provable from Ax and the equations EQ defining K. (Sethi's algorithm [11] applies here only when Σ and EQ are empty.)

In § 2, we model the same form of "if-then-else", only this time we permit the underlying sets to contain an element modeling "undefined". More precisely, we let $K^+(\perp)$ be the class of all sets A containing a distinguished element $\perp$ and a function $\kappa: A^4 \to A$ which satisfies the condition:

$$\kappa(x, y, u, v) = \begin{cases} \perp & \text{if either } x \text{ or } y \text{ is } \perp, \\ u & \text{if } x = y, \\ v & \text{if } x \neq y. \end{cases}$$

(We have not allowed A to contain other functions aside from κ, since the resulting complication would lengthen an already over-long paper.) This version of "if-then-else" might be said to treat its inputs "call-by-need".

In § 3, we consider the class $\underline{K(3)}$ of those algebras A containing a distinguished constant T and a function $\kappa: A^3 \to A$ which satisfies the condition

$$\kappa(p, x, y) = \begin{cases} x & \text{if } p = \mathrm{T}, \\ y & \text{if } p \neq \mathrm{T}. \end{cases}$$

Thus T models "true", and any element other than "true" plays the role of "false".

In § 4, the class $K(3; \perp)$ is defined. Algebras A in this class contain three distinguished constants T, F, $\perp$ and a function $\kappa: A^3 \to A$ which satisfies the condition:

$$\kappa(p, x, y) = \begin{cases} x & \text{if } p = \mathrm{T}, \\ y & \text{if } p = \mathrm{F}, \\ \perp & \text{otherwise.} \end{cases}$$

Again, axioms are found for the identities valid in $K^+(\perp)$, $K(3)$ and $K(3; \perp)$.

The reasons we have treated just the four cases mentioned above are these. First, as already mentioned, in the literature both a three- and a four-argument function have been used to formalize "if-then-else". Second, for each choice of the number of arguments, there is the possibility of including an "undefined" element. The introduction of such an element raises enough new difficulties to deserve a separate treatment. Last, we found that any variation on our four versions, including a two sorted version (Boolean and "elements") of the three argument case, could be treated by methods very similar to the ones already considered. (It should be mentioned that we did not consider the question of whether the axioms we found are independent.)

There may be some interest in the method used here to prove the difficult part of the completeness theorems. In logic, it is usually the case that one shows that when a statement (of the appropriate form) is not provable from the axioms, one can then find a model of the axioms in which the statement is false. In contrast, here, completeness is proved directly by induction on the "size" of the statement.

Preliminaries. We begin with some notation. If $h: X \to Y$ is a function, the value of h on $x \in X$ will be written as xh or $h(x)$. The set of nonnegative integers is denoted ω. If $n \in \omega$, $[n]$ is $\{1, 2, \cdots, n\}$; thus $[0]$ is empty. The set $\{1, 2, \cdots\}$ is denoted $[\omega]$. For any set X, X^* is the set of all finite sequences ("words") of elements of X, including the empty sequence λ. For $\nu \in X^*$, the length of ν is denoted $|\nu|$, so that $|\lambda| = 0$. We will always identify an element x of X with a word in X^* of length one, so that $X \subseteq X^*$. For ν, ν' in X^*, the concatenation of ν and ν' is written $\nu\nu'$.

A *ranked alphabet* Σ (sometimes called a "signature") is the union $\Sigma = \cup\Sigma_n$ of the pairwise disjoint sets Σ_n, $n \in \omega$. Let V be a countably infinite set disjoint from Σ. The set T_Σ of "finite Σ-terms" built from V and Σ will be defined as follows. An element t of T_Σ is a partial function $t: [\omega]^* \to \Sigma \cup V$ which has the following properties (see [3], [7] or [12]).

(i) the domain of t, dom (t), is a finite, nonempty prefix-closed subset of $[\omega]^*$,

(ii) if $\nu \in \text{dom } t$ and $\nu t \in \Sigma_n$, then $\nu i \in \text{dom } (t)$ iff $i \in [n]$,

(iii) if $\nu \in \text{dom } t$ and $\nu t \in \Sigma_0 \cup V$, then $\nu i \notin \text{dom } (t)$, for any $i \in [\omega]$.

If $t \in T_\Sigma$ and $\nu \in \text{dom } (t)$, then we let t_ν denote the term which satisfies:

$$\mu t_\nu = (\nu\mu)t \quad \text{for all } \mu \in [\omega]^*.$$

The terms of the form t_ν, for $t \in T_\Sigma$, $\nu \in \text{dom } (t)$, are the *subterms* of t.

If Σ is a ranked alphabet, a Σ-*algebra* is a nonempty set A equipped with functions $\sigma_A: A^n \to A$, for each $\sigma \in \Sigma_n$, each nonnegative integer n. When $\sigma \in \Sigma_0$, $\sigma_A \in A$. (We will usually drop the subscript on σ.)

Elements of $\Sigma_0 \cup V$ are identified with the values of the partial functions in T_Σ defined only on λ.

If A and B are Σ-algebras, a homomorphism $h: A \to B$ is a function $A \to B$ such that for all $\sigma \in \Sigma_n$, $n \geqq 0$, $a_1, \cdots, a_n \in A$, $h(\sigma(a_1, \cdots, a_n)) = \sigma(h(a_1), \cdots h(a_n))$. The set T_Σ is a Σ-algebra in a natural way: given $t_1, \cdots, t_n \in T_\Sigma$, and $\sigma \in \Sigma_n$, $\sigma(t_1, \cdots, t_n)$ is the partial function $t: [\omega]^* \to \Sigma \cup V$ satisfying: $\lambda t = \sigma$ and $i\nu t = \nu t_i$, for each $i \in [n]$, $\nu \in [\omega]^*$. In fact, T_Σ is freely generated by V in the class of all Σ-algebras: i.e., for any Σ-algebra A and any function $h: V \to A$, there is a unique homomorphism $h^\#: T_\Sigma \to A$ extending h.

Finally, we review some facts on standard equational logic. An *equation* between Σ-terms is an expression of the form $t \approx t'$, for $t, t' \in T_\Sigma$. If A is a Σ-algebra and $h: T_\Sigma \to A$ is a homomorphism, we say h *satisfies* the equation $t \approx t'$, for t, t' in T_Σ, if $th = t'h$. The equation $t \approx t'$ is *true in* A if every homomorphism $T_\Sigma \to A$ satisfies it. For a set EQ of equations, Mod (EQ) is the class of all Σ-algebras A such that each equation in EQ is true in A. If K is a class of Σ-algebras, an equation is true (or valid) in K if it is true in A, for each $A \in K$.

If EQ is a set of Σ-equations, and $s, s' \in T_\Sigma$ we write

$$\text{EQ} \models s \approx s'$$

if Mod (EQ) $\subseteq$ Mod $(s \approx s')$, and we write

$$\text{EQ} \overset{\circ}{\models} s \approx s'$$

if, for every Σ-algebra A and every homomorphism $h: T_\Sigma \to A$, h satisfies $s \approx s'$ whenever h satisfies each equation in EQ.

We recall the following completeness theorems for equational logic. A reference for the $\models$ version is G. Gratzer, *Universal Algebra*, Van Nostrand, p. 170. The other version follows easily from the method used to prove this one.

THEOREM. *Let* EQ, s, s' *be as above. Then* $\mathrm{EQ} \models s \approx s'$ *iff the equation* $s \approx s'$ *is deducible from* EQ *using the rules* R1–R5 *below*; $\mathrm{EQ} \overset{\circ}{\models} s \approx s'$ *iff* $s \approx s'$ *is deducible from* EQ *using only* R1–R4.

R1. *infer* $x \approx x$, *for any* $x \in T_\Sigma$;

R2. *from* $x \approx y$, *infer* $y \approx x$, *for any* $x, y \in T_\Sigma$;

R3. *from* $x \approx y$, *and* $y \approx z$, *infer* $x \approx z$, *for any* $x, y, z \in T_\Sigma$;

R4. *from* $x_i \approx y_i$, $i \in [n]$, *infer* $\sigma(x_1, \cdots, x_n) \approx \sigma(y_1, \cdots, y_n)$ *for any* $x_i, y_i \in T_\Sigma$, $\sigma \in \Sigma_n$;

R5. *from* $x \approx y$, *infer* $xe \approx ye$, *for any* $x, y \in T_\Sigma$, *any endomorphism* $e: T_\Sigma \to T_\Sigma$.

We will write "$\mathrm{EQ} \vdash s \approx s'$" to mean "$s \approx s'$ is deducible from EQ using R1–R5". The five rules R1–R5 constitute "standard equational logic." In this notation, the first part of the completeness theorem may be stated: $\models = \vdash$.

Recall that a (formal) *deduction* of the equation $s \approx s'$ from the set EQ of equations is a finite sequence $\alpha_1, \cdots, \alpha_n$ of equations such that α_n is $s \approx s'$ and for each $i \in [n]$, either $\alpha_i \in \mathrm{EQ}$ or α_i follows from earlier α_j using one of the rules.

1. Axiomatization of K^+. Let Σ be a fixed ranked set. If A is any Σ-algebra, let $\kappa: A^4 \to A$ be the function defined as follows, for all x, y, u, v in A:

$$\kappa(x, y, u, v) = \begin{cases} u & \text{if } x = y, \\ v & \text{if } x \neq y. \end{cases} \tag{1.1}$$

Let A^+ denote the expansion of A obtained by adding κ to the other functions on A, so that A^+ is a Σ^+-algebra, where $\Sigma_4^+ = \Sigma_4 \cup \{\kappa\}$; $\Sigma_n^+ = \Sigma_n$ otherwise. If K is a class of Σ-algebras, we let K^+ denote the collection of all algebras A^+, for A in K.

In this section we will axiomatize all equations between elements of $T_{\Sigma+}$ which are valid in any class of the form K^+ when K is an equational class of Σ-algebras. In 1.2 below we will define a set Ax of equations and later we will prove (in 1.23) the following, (by induction on the "size" of $t \approx t'$).

COMPLETENESS THEOREM. *If* K *is an equational class of* Σ*-algebras* (*say* $A \in K$ *iff* EQ *is true in* A, *when* EQ *is a set of equations between elements of* T_Σ), *and if* t, t' *are any two* Σ^+*-terms, then* $(t \approx t')$ *is valid in* K^+ *iff* $Ax \cup \mathrm{EQ} \vdash (t \approx t')$.

The set Ax of "axioms" is written using the notation "$[x, y, u, v]$" in place of "$\kappa(x, y, u, v)$".

DEFINITION 1.2. Let a, b, x, y, z, u, v, w (sometimes with subscripts) be distinct variables in V. Then the following equations belong to Ax:

(1.2.1) (equal premise)

$$[x, x, u, v] \approx u,$$

(1.2.2) (equal conclusion)

$$[x, y, u, u] \approx u,$$

(1.2.3) (premise commutativity)

$$[x, y, u, v] \approx [y, x, u, v],$$

(1.2.4) (conclusion replacement)

$$[x, y, x, u] \approx [x, y, y, u],$$

(1.2.5) (premise replacement)

$$[x, y, [x, z, u, v], w] \approx [x, y, [y, z, u, v], w],$$

(1.2.6) (positive redundancy)

$$[x, y, [x, y, u, v], w] \approx [x, y, u, w],$$

(1.2.7) (negative redundancy)

$$[x, y, u, [x, y, v, w]] \approx [x, y, u, w],$$

(1.2.8) (premise interchange)

$$[x, y, [u, v, a, b], [u, v, z, w]] \approx [u, v, [x, y, a, z], [x, y, b, w]],$$

(1.2.9) (premise simplification)

$$[[x, y, u, v], a, b, z] \approx [x, y, [u, a, b, z], [v, a, b, z]],$$

(1.2.10) (Σ reduction): for each $\sigma \in \Sigma_n$, $n > 0$, each $i \in [n]$:

$$\sigma(x_1, \cdots, x_{i-1}, [u, v, a, b], x_{i+1}, \cdots, x_n)$$
$$\approx [u, v, \sigma(x_1, \cdots, x_{i-1}, a, x_{i+1}, \cdots, x_n), \sigma(x_1, \cdots, x_{i-1}, b, x_{i+1}, \cdots, x_n)],$$

(1.2.11) (congruence axioms). For each $\sigma \in \Sigma_n$, $n > 0$, each $i \in [n]$

(a)
$$[u, v, \sigma(x_1, \cdots, x_{i-1}, u, x_{i+1}, \cdots, x_n), y]$$
$$\approx [u, v, \sigma(x_1, \cdots, x_{i-1}, v, x_{i+1}, \cdots, x_n), y],$$

(b)
$$[u, v, [\sigma(x_1, \cdots, x_{i-1}, u, x_{i+1}, \cdots, x_n), y, a, b], w]$$
$$\approx [u, v, [\sigma(x_1, \cdots, x_{i-1}, v, x_{i+1}, \cdots, x_n), y, a, b], w].$$

Before beginning the proof, we make a preliminary remark which might motivate the long argument. Part of the problem in showing the completeness of any set of axioms for this version of "if-then-else" has to do with the logic of signed equations. For example, since

$$x \approx y, y \not\approx z \overset{\circ}{\models} x \not\approx z$$

it follows that

$$\models [x, y, [y, z, a, b], c] \approx [x, y, [y, z, a, [x, z, d, b]], c].$$

Thus we were led to the "insertion lemma" 1.14, one of our principal tools. Another idea was to prove a theorem apparently stronger than the completeness theorem itself (actually, the two are equivalent). In Theorem 1.23, we show that if M is any finite set of signed equations and M semantically entails $t \approx t'$, then it is provable from our axioms that

$$\tau_{\tilde{M}}(t) \approx \tau_{\tilde{M}}(t')$$

where $\tau_{\tilde{M}}(t)$ is an "if-then-else" term which behaves as t does: whenever the signed equations in M are satisfied. When M is the empty set, $\tau_{\tilde{M}}(t)$ is just t, so the completeness theorem follows.

It is convenient to note two immediate corollaries of the above axioms.

PROPOSITION 1.3. *For any Σ^+-terms u, v, x, y, a, b and c*

(i) $$\mathbf{Ax} \vdash [u, v, [x, y, a, b], c] \approx [x, y, [u, v, a, c], [u, v, b, c]]$$

and

(ii) $$\mathbf{Ax} \vdash [u, v, a, [x, y, b, c]] \approx [x, y, [u, v, a, b], [u, v, a, c]].$$

Proof. We prove only the first equation. First, by the equal conclusion axiom (1.2.2)

$$\mathrm{Ax} \vdash [x, y, c, c] \approx c,$$

so that

$$\mathrm{Ax} \vdash [u, v, [x, y, a, b], c] \approx [u, v, [x, y, a, b], [x, y, c, c]].$$

Now apply the premise interchange axiom (1.2.8) to the right-hand side:

$$\mathrm{Ax} \vdash [u, v, [x, y, a, b], [x, y, c, c]] \approx [x, y, [u, v, a, c], [u, v, b, c]].$$

Using rule R3 (transitivity), the proof is complete.

DEFINITION 1.4. A Σ^+-term t is a *simple premise* term if whenever $\kappa(x, y, u, v)$ is a subterm of t, then there are no occurrences of κ in x or y—i.e. x and y are Σ-terms.

Using induction and the axiom of premise simplification and (1.2.3) one easily proves

PROPOSITION 1.5. *For any Σ^+-term t there is a simple premise term t' such that* $\mathrm{Ax} \vdash t \approx t'$.

DEFINITION 1.6. A Σ^+-term t is *Σ-reduced* if whenever $\sigma \in \Sigma_n$, $n > 0$, and $\sigma(x_1, \cdots, x_n)$ is a subterm of t, then there is no occurrence of κ in any x_i, $i \in [n]$.

Using the axiom of Σ-reduction, one may prove

PROPOSITION 1.7. *For every Σ^+-term t there is a Σ-reduced term t' such that* $\mathrm{Ax} \vdash t \approx t'$.

From Propositions 1.5 and 1.7 it follows that every Σ^+-term is provably equal to a Σ-reduced, simple-premise term. We will identify such terms with certain partial functions in not quite the way sketched in the Preliminary section.

DEFINITION 1.8. A *reduced tree* (more fully, a Σ-reduced simple-premise Σ^+-tree) is a partial function $t: [4]^* \to \{\kappa\} \cup T_\Sigma$, whose domain is a nonempty, finite, prefix-closed subset of $[4]^*$, satisfying the following conditions.

(1.8.1) If $\nu \in [4]^*$ and $\nu t = \kappa$ then $\nu i t$ is defined for each $i \in [4]$, and moreover $\nu 1t$ and $\nu 2t$ belong to T_Σ;

(1.8.2) if $\nu \in [4]^*$ and $\nu t \in T_\Sigma$ then $\nu i t$ is undefined for each $i \in [4]$.

A reduced tree t represents (in the obvious way) a labeled tree whose *vertices* are the words in the domain of t. We will say "ν is a vertex of t" instead of "ν is in the domain of t". The *label* of the vertex $\nu \in [4]^*$ is νt. The *leaves* of the tree are labeled by elements of T_Σ; the vertices which are not leaves are labeled κ. For reduced trees t a *nonpremise vertex* is an element of $\{3, 4\}^*$ in the domain of t. We will write $[x, y, t_3, t_4]$ to denote the reduced tree t satisfying: $\lambda t = \kappa$; $1t = x$; $2t = y$; $i\mu t = \mu t_i$, for $i = 3, 4$, $\mu \in [4]^*$.

From now on we will be dealing only with such reduced trees.

A "signed Σ-equation" is an expression of the form $s \approx s'$ or $s \not\approx s'$, for $s, s' \in T_\Sigma$. The following notion plays an important role.

DEFINITION 1.9. Let t be a reduced tree and suppose ν is a nonpremise vertex of t. The set of signed equations $S(\nu, t)$ between Σ-terms (not Σ^+-terms!) is defined by induction on $|\nu|$.

(1.9.1) $S(\lambda, t) = \varnothing$, the empty set.

(1.9.2) If $t = [x, y, t_3, t_4]$, then if $\nu = i\mu$, $i \in \{3, 4\}$,

$$S(3\mu, t) = \{x \approx y\} \cup S(\mu, t_3), \text{ and}$$

$$S(4\mu, t) = \{(x \not\approx y)\} \cup S(\mu, t_4), \text{ where}$$

$x, y \in T_\Sigma$, and both t_3 and t_4 are reduced trees.

Informally, the set $S(\nu, t)$ is the set of signed equations which must be satisfied in order to "reach" the vertex ν. This is made precise in Definition 1.12 below.

Let A be a Σ^+-algebra and let $h: T_\Sigma \to A$ be a homomorphism. Suppose that t is a reduced tree.

DEFINITION 1.10. Path (h, t) is an element of $\{3, 4\}^*$ defined as follows:

if λt is a Σ-term, Path $(h, t) = \lambda$;

if $t = [x, y, t_3, t_4]$ then

$$\text{Path } (h, t) = \begin{cases} 3 \text{ Path } (h, t_3) & \text{if } xh = yh, \\ 4 \text{ Path } (h, t_4) & \text{if } xh \neq yh. \end{cases}$$

Here, 3 Path (h, t_3) is the word formed by concatenating 3 with the word Path (h, t_3). For example if u, v, a, b, c are in T_Σ and $uh \neq vh$, then when $t = [u, u, [v, u, a, b], c]$, Path $(h, t) = 34$.

Remark 1.11. Let t be a reduced tree and $h: T_{\Sigma^+} \to A^+$ be a homomorphism, where $A \in K$. Then th, the *value* of h on t, is $(\nu t)h$, where $\nu = \text{Path } (h, t)$.

Remark 1.12. For any reduced tree t, a homomorphism $h: T_{\Sigma^+} \to A^+$ will satisfy each equation or inequation in $S(\nu, t)$ iff ν is a prefix of Path (h, t).

We need some further notation.

DEFINITION 1.13. Suppose that S is a set of signed equations between Σ-terms and let α be some signed equation. Then we write $S \overset{\circ}{\models}_K \alpha$ (or just $S \overset{\circ}{\models} \alpha$ when being lazy) if for every Σ-algebra A in K and every homomorphism $h: T_\Sigma \to A$, h satisfies α whenever h satisfies each signed equation in S. Similarly, when t and t' are reduced trees, we write $S \overset{\circ}{\models}_K t \approx t'$ if $th = t'h$ for every homomorphism $h: T_{\Sigma^+} \to A^+$, $A \in K$, which satisfies S.

We recall the following convention: if ν is a node of the reduced tree t, then t_ν will denote the tree of descendents of ν in t (see the Preliminaries). If t' is any reduced tree, $t(\nu/t')$ denotes the tree obtained from t by replacing t_ν by t'. More precisely for any $\mu \in [4]^*$

$$\mu t(\nu/t') = \begin{cases} \mu t & \text{if } \nu \text{ is not a prefix of } \mu, \\ \mu' t' & \text{if } \mu = \nu\mu'. \end{cases}$$

A useful observation is that

$$S(\nu_1\nu_2, t_1(\nu_1/t_2)) = S(\nu_1, t_1) \cup S(\nu_2, t_2) \tag{1.13}$$

if ν_i is a nonpremise vertex of t_i, $i = 1, 2$.

The following lemma is the main tool used in the proof of the completeness of Ax $\cup$ EQ (recall: K is the class of all models of the set of equations EQ).

INSERTION LEMMA 1.14. *Let ν be a nonpremise node of the reduced tree t and suppose that x and y are Σ-terms. Then*

$$\text{if } S(\nu, t) \overset{\circ}{\models}_K (x \approx y) \text{ then } \text{Ax} \cup \text{EQ} \vdash t \approx t(\nu/[x, y, t_\nu, a]),$$

and

$$\text{if } S(\nu, t) \overset{\circ}{\models}_K (x \neq y) \text{ then } \text{Ax} \cup \text{EQ} \vdash t \approx t(\nu/[x, y, a, t_\nu])$$

where a is a "new variable," i.e. a variable which does not occur in x, y or t.

The proof of the insertion lemma will be postponed until the end of this section. We will first show how the completeness theorem follows from some corollaries of Lemma 1.14. The first corollary might be called the "deletion lemma."

COROLLARY 1.15. *Using the notation of the insertion lemma, if t_ν is the reduced tree $[x, y, t_3, t_4]$ and $S(\nu, t) \overset{\circ}{\models}_K x \approx y$ (respectively $S(\nu, t) \overset{\circ}{\models}_K x \neq y$) then* $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\nu/t_3)$ (*respectively*, $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\nu/t_4)$).

Proof. Apply the insertion lemma to the tree $t' = t(\nu/t_i)$, $i = 3$ or 4. Note that $S(\nu, t') = S(\nu, t)$.

COROLLARY 1.16. *With the notation of the insertion lemma, if $S(\nu, t) \overset{\circ}{\models}_K x \approx y$, then* $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\nu/x \mapsto y)$, *where $t(\nu/x \mapsto y)$ is the tree obtained from t by changing every occurrence of x in t_ν to y.*

Proof. We will show how to replace one occurrence of x in t_ν by y. The corollary then follows by induction. Suppose $\bar{\nu} = \nu\nu'$ is a leaf of t_ν labeled by a Σ-term, say $p(x)$, having x as a subterm. First, suppose that $\bar{\nu}$ is a *nonpremise* leaf. Then clearly $S(\bar{\nu}, t) \overset{\circ}{\models}_K p(x) \approx p(y)$, where $p(y)$ is the result of replacing every occurrence of x by y in $p(x)$. Then, by the insertion lemma, $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\bar{\nu}/[p(x), p(y), t_{\bar{\nu}}, a])$. But $t_{\bar{\nu}}$ is the Σ-term $p(x)$ and furthermore, by the conclusion replacement axiom (1.2.4), $\mathrm{Ax} \vdash [p(x), p(y), p(x), a] \approx [p(x), p(y), p(y), a]$. Thus $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\bar{\nu}/[p(x), p(y), p(y), a])$. Now apply Corollary 1.15.

In case $\bar{\nu}$ is a premise leaf, let μ be the immediate predecessor of $\bar{\nu}$ (so that $\bar{\nu}$ is $\mu 1$ or $\mu 2$). Then $S(\mu, t) \overset{\circ}{\models}_K p(x) \approx p(y)$, so that by the insertion lemma, $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\mu/[p(x), p(y), [p(x), u, t_3, t_4], a])$, if t_μ is $[p(x), u, t_3, t_4]$. We now apply the premise replacement axiom and Corollary 1.15, as above.

DEFINITION 1.17. A nonpremise vertex ν of the reduced tree t is *accessible* if for some homomorphism $h: T_{\Sigma^+} \to A^+$, $A^+ \in K^+$, ν is a prefix of Path (h, t). The tree t itself is accessible if every nonpremise vertex of t is accessible.

COROLLARY 1.18. *For every reduced tree t there is an accessible reduced tree t' such that* $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t'$.

Proof. If t is not accessible, let ν be a nonpremise vertex of shortest length which is not accessible. Clearly $\nu \neq \lambda$, so $\nu = \mu 3$ or $\mu 4$, for some $\mu \in \{3, 4\}^*$. Suppose that $t_\mu = [x, y, t_3, t_4]$. If $\nu = \mu 3$, then, since ν is inaccessible, $S(\mu, t) \overset{\circ}{\models}_K x \neq y$. So by Corollary 1.15, $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\mu/t_4)$. Similarly, if ν is $\mu 4$, $\mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\mu/t_3)$. In either case, t is provably equal to a tree with at least one fewer inaccessible vertices. The proof is completed by induction on the number of inaccessible vertices in t.

We turn now to the construction of a reduced tree $\tau_{\tilde{M}}$ which will code, in a certain sense, a finite sequence $\tilde{M}$ of signed equations between Σ-terms. Suppose we write the sequence $\tilde{M}$ in the form

$$(\alpha_1; j_1), \quad (\sigma_2; j_2), \quad \cdots, \quad (\alpha_k; j_k) \tag{1.19}$$

where $j_1, \cdots, j_k$ are elements in $\{3, 4\}$, and for each $i \in [k]$, α_i is an equation

$$s_i \approx s_i'. \tag{1.20}$$

If j_i is 3, the pair (α_i, j_i) codes the positive equation α_i; if j_i is 4, (α_i, j_i) codes the negation $s_i \not\approx s_i'$ of α_i. The tree corresponding to $\tilde{M}$ is defined by induction on the length k of $\tilde{M}$.

First, let $a_1, \cdots, a_k, b$ be $k+1$ distinct variables not occurring in any equation α_i, $i \in [k]$. Then, if $k = 0$, $\tau_{\tilde{M}}$ is the tree whose root is a leaf labeled b; if $k > 0$, write $\tilde{M}$ as $(\alpha_1, j_1), \tilde{M}'$. Then

$$\tau_{\tilde{M}} = \begin{cases} [s_1, s_1', \tau_{\tilde{M}'}, a_1] & \text{if } j_1 = 3, \\ [s_1, s_1', a_1, \tau_{\tilde{M}'}] & \text{if } j_1 = 4. \end{cases} \tag{1.21}$$

Now if M is a finite *set* of signed equations between Σ-terms we say that a finite sequence $\tilde{M}$ of the form (1.19) is an *arrangement of* M if an equation α belongs to

M iff for at least some $i \in [k]$, α_i is α and $j_i = 3$; and furthermore, a negated equation $\neg\alpha$ belongs to M iff for at least some $i \in [k]$, α_i is α and $j_i = 4$. (Thus an arrangement of M is a coded listing, perhaps with repetitions, of the signed equations in M.)

For the sequence $\tilde{M}$ of (1.19), let $\nu_{\tilde{M}}$ be the word $j_1 j_2 \cdots j_k$ in $\{3,4\}^*$. The following is the fundamental property of $\tau_{\tilde{M}}$.

FACT 1.22. *Let $\tilde{M}$ be any arrangement of the finite set M of signed equations. Then $S(\nu_{\tilde{M}}, \tau_{\tilde{M}}) = M$.*

Below we will write $\tau_{\tilde{M}}(t)$ instead of $\tau_{\tilde{M}}(\nu_{\tilde{M}}/t)$ to denote the result of replacing the variable b in $\tau_{\tilde{M}}$ by t.

The following theorem is the main result of this section. When M is empty, Theorem 1.23 is the completeness theorem.

THEOREM 1.23. *Suppose that M is a finite set of signed equations between Σ-terms. Let $\tilde{M}$ be any arrangement of M and let t and t' be reduced trees. If*

$$M \overset{\circ}{\models}_K t \approx t' \tag{1.24}$$

then

$$\mathrm{Ax} \cup \mathrm{EQ} \vdash \tau_{\tilde{M}}(t) \approx \tau_{\tilde{M}}(t'). \tag{1.25}$$

Note. From now on, we will write $\overset{\circ}{\models}$ instead of $\overset{\circ}{\models}_K$.

Proof. By induction on the number of occurrences of κ in t.

Basis step. t is a Σ-term. Let ν be any accessible nonpremise leaf of $\tau_{\tilde{M}}(t')$ of the form $\nu_{\tilde{M}}\nu'$ where ν' is a nonpremise leaf of t' (the definition of $\nu_{\tilde{M}}$ precedes 1.22). Suppose that ν is labeled by the Σ-term z. We claim that

$$S(\nu, \tau_{\tilde{M}}(t')) \overset{\circ}{\models} t \approx z.$$

Indeed $S(\nu, \tau_{\tilde{M}}(t')) = S(\nu_{\tilde{M}}, \tau_{\tilde{M}}) \cup S(\nu', t')$ by (1.13), and $S(\nu_{\tilde{M}}, \tau_{\tilde{M}}) = M$, by Fact 1.22. If $h: T_\Sigma \to A, A \in K$, is any homomorphism satisfying M, then $th = t'h$, by the assumption (1.24). But if h also satisfies $S(\nu', t')$, then $t'h = zh$ by Remark 1.11. Thus the claim is proved. Since ν was arbitrary, by Corollary 1.16 it follows that $\tau_{\tilde{M}}(t')$ is provably equal to a tree of the form $\tau_{\tilde{M}}(t'')$ such that every accessible nonpremise leaf of t'' in $\tau_{\tilde{M}}(t'')$ is labeled by the Σ-term t. But by Corollary 1.18 we may further assume that every nonpremise leaf of t'' in $\tau_{\tilde{M}}(t'')$ is accessible. Applying the equal conclusion axiom (1.2.2), we see that $\mathrm{Ax} \cup \mathrm{EQ} \vdash \tau_{\tilde{M}}(t) \approx \tau_{\tilde{M}}(t'')$, completing the proof of the basis step.

Induction step. Assume that t has the form $[x, y, t_3, t_4]$ for some Σ-terms x and y and some reduced trees t_3, t_4. We may further assume that t' also has the form $[x, y, t'_3, t'_4]$. (For if not, we replace t' by $[x, y, t', t']$.) Now if

$$M \overset{\circ}{\models} [x, y, t_3, t_4] = [x, y, t'_3, t'_4], \tag{1.26}$$

then

$$M \cup \{x \approx y\} \overset{\circ}{\models} t_3 \approx t'_3 \tag{1.27}$$

and

$$M \cup \{x \neq y\} \overset{\circ}{\models} t_4 \approx t'_4. \tag{1.28}$$

Write M_3 for $M \cup \{x \approx y\}$ and M_4 for $M \cup \{x \neq y\}$. By the induction hypothesis,

$$\mathrm{Ax} \cup \mathrm{EQ} \vdash \tau_{\tilde{M}_3}(t_3) \approx \tau_{\tilde{M}_3}(t'_3) \tag{1.29}$$

and

$$\mathrm{Ax} \cup \mathrm{EQ} \vdash \tau_{\tilde{M}_4}(t_4) \approx \tau_{\tilde{M}_4}(t'_4). \tag{1.30}$$

But the insertion lemma and Corollary 1.15 imply that

$$\text{Ax} \cup \text{EQ} \vdash \tau_{\tilde{M}}[x, y, t_3, t_4] \approx \tau_{\tilde{M}}[x, y, \tau_{\tilde{M}_3}(t_3), \tau_{\tilde{M}_4}(t_4)] \tag{1.31}$$

and

$$\text{Ax} \cup \text{EQ} \vdash \tau_{\tilde{M}}[x, y, t_3', t_4'] \approx \tau_{\tilde{M}}[x, y, \tau_{\tilde{M}_3}(t_3'), \tau_{\tilde{M}_4}(t_4')]. \tag{1.32}$$

The four facts (1.29)–(1.32) imply that

$$\text{Ax} \cup \text{EQ} \vdash \tau_{\tilde{M}}(t) \approx \tau_{\tilde{M}}(t'), \tag{1.33}$$

completing the proof.

Our only remaining task is to prove the insertion lemma, which for convenience we repeat here.

INSERTION LEMMA 1.14. *Let ν be a nonpremise node of the reduced tree t and suppose that x and y are Σ-terms. Then if $S(\nu, t) \overset{\circ}{\models} x \approx y$, $\text{Ax} \cup \text{EQ} \vdash t \approx t(\nu/[x, y, t_\nu, a])$; and if $S(\nu, t) \overset{\circ}{\models} x \not\approx y$, $\text{Ax} \cup \text{EQ} \vdash t \approx t(\nu/[x, y, a, t_\nu])$, where a is a variable not occurring in x, y or t.*

Before beginning the proof, we introduce some notation. Let $P(n)$ stand for the following statement:

(1.34) Let ν be any word in $\{3, 4\}^*$ of length at most n. Then for any Σ-terms x and y, and for any reduced tree t: if ν is a nonpremise node of t and $S(\nu, t) \overset{\circ}{\models} x \approx y$, then $\text{Ax} \cup \text{EQ} \vdash t \approx t(\nu/[x, y, t_\nu, a])$, for a "new variable" a. Let $Q(n)$ be the statement obtained from (1.34) by replacing "$x \approx y$" by "$x \not\approx y$" and replacing "$[x, y, t_\nu, a]$" by "$[x, y, a, t_\nu]$".

Our proof of the insertion lemma will go as follows: we first prove $P(0)$ and $Q(0)$; assuming $P(n)$ and $Q(n)$ we then first prove $P(n+1)$; assuming $P(n)$, $Q(n)$, $P(n+1)$ we prove $Q(n+1)$. Thus by induction, we will have proved $P(n)$ and $Q(n)$, for all nonnegative integers n, which is equivalent to proving the lemma.

Proof of $P(0)$. In this case $\nu = \lambda$ and $S(\lambda, t) = \varnothing$, for any reduced tree t. Hence if $\varnothing \overset{\circ}{\models} x \approx y$, $\text{EQ} \vdash x \approx y$, since K is Mod (EQ). Thus

$$\text{Ax} \cup \text{EQ} \vdash [x, y, t, a] \approx [x, x, t, a].$$

But clearly $\text{Ax} \cup \text{EQ} \vdash t \approx [x, x, t, a]$, by the equal premise axiom. Thus $P(0)$ is proved.

Proof of $Q(0)$. Since $S(\lambda, t) = \varnothing$, it is impossible that $\varnothing \overset{\circ}{\models} x \not\approx y$ (since K contains one-element algebras). Thus $Q(0)$ holds vacuously.

Induction step. Proof of $P(n+1)$ (assuming $P(n)$ and $Q(n)$ hold). Let $\nu = \mu i$, $\mu \in \{3, 4\}^*$, $|\mu| = n$, $i \in \{3, 4\}$. Assume ν is a nonpremise node of t. Write t_μ as $[u, v, t_3, t_4]$.

Case A. First we will assume that $S(\mu, t) \overset{\circ}{\models} x \approx y$.

Then if $i = 3$, we want to show

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\mu/[u, v, [x, y, t_3, a], t_4]). \tag{1.35}$$

Recalling Proposition 1.3(i), we have

$$\text{Ax} \cup \text{EQ} \vdash t(\mu/[u, v, [x, y, t_3, a], t_4]) \approx t(\mu/[x, y, [u, v, t_3, t_4], [u, v, a, t_4]]). \tag{1.36}$$

But using the induction hypothesis $P(n)$

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\mu/[x, y, [u, v, t_3, t_4], a]). \tag{1.37}$$

Now if we replace a in the right-hand tree in (1.37) by $[u, v, a, t_4]$, the proof of this case is complete.

The proof in the case $i = 4$ is similar and is omitted.

Case B. Now assume $S(\mu, t) \not\models (x \approx y)$, but $S(\nu, t) \models x \approx y$.

Subcase 1. $i = 3$, so that $S(\mu, t)$, $u \approx v \models x \approx y$.

Subcase 1.1. $S(\mu, t)$ together with $u \approx v$ is inconsistent. Then $S(\mu, t) \models u \not\approx v$. Then, by the induction hypothesis $Q(n)$,

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\mu/[u, v, a, [u, v, t_3, t_4]]) \tag{1.38}$$

where a is a new variable. Thus, if we substitute $[x, y, t_3, a]$ for a in the right-hand tree of (1.38), we obtain

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\mu/[u, v, [x, y, t_3, a], [u, v, t_3, t_4]]). \tag{1.39}$$

Now apply the negative redundancy axiom (1.2.7) to (1.39). We obtain

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\mu/[u, v, [x, y, t_3, a], t_4]) \tag{1.40}$$

which completes the proof of Subcase 1.1.

Subcase 1.2. $S(\mu, t)$, $u \approx v \models x \approx y$ and $S(\mu, t) \cup \{u \approx v\}$ is consistent. Then (see the Appendix), $S^+(\mu, t)$, $u \approx v \models x \approx y$, where $S^+(\mu, t)$ denotes the set of positive equations in $S(\mu, t)$. By the completeness theorem for $\models$ (see the Preliminaries) there is a proof of $x \approx y$ from $S^+(\mu, t) \cup \{u \approx v\} \cup \text{EQ}$ using the reflexivity, symmetry, transitivity and congruence rules (R1–R4 in the Preliminaries section). Let $R(k)$ denote the following statement:

> for any reduced tree $\bar{t}$ such that $S(\bar{t}, \nu) = S(t, \nu)$, for any Σ-terms α and β, if $\alpha \approx \beta$ is provable in at most k steps from $\text{EQ} \cup S^+(\bar{t}, \mu) \cup \{u \approx v\}$ using the reflexivity symmetry, transitivity and congruence rules, then
>
> $$\text{Ax} \cup \text{EQ} \vdash \bar{t} \approx \bar{t}(\nu/[\alpha, \beta, t_\nu, a]), \tag{1.41}$$
>
> where a is a new variable.

We will prove $R(k)$ for all k, by induction, which will complete the proof of Subcase 1 of $P(n+1)$.

If $k = 1$, then either $(\alpha \approx \beta)$ is $(u \approx v)$ or $S^+(\bar{t}, \mu) \models \alpha \approx \beta$. In the second case, we apply Case A. In the case α is u and β is v, we apply the axiom of positive redundancy.

Now assume $R(k)$, and that $\alpha \approx \beta$ is provable using the above-mentioned rules from EQ, $S^+(\mu, \bar{t})$, $u \approx v$ in $k+1$ steps. We consider the rule applied last. We may omit the reflexivity rule, by the case R(1).

Suppose the last rule applied was the symmetry rule. Then, by the induction assumption $R(k)$,

$$\text{Ax} \cup \text{EQ} \vdash \bar{t} \approx \bar{t}(\gamma/[\beta, \alpha, \bar{t}_\nu, a]). \tag{1.42}$$

We need only apply the axiom of premise commutativity to the right-hand tree of (1.42) to complete the proof.

Suppose $\alpha \approx \beta$ follows from $\alpha \approx \gamma$, $\gamma \approx \beta$ by the transitivity rule. Let t^* denote the tree $\bar{t}(\nu/[\alpha, \beta, \bar{t}_\nu, a'])$ where a'' is a new variable. Then $S(\nu, t^*) = S(\nu, \bar{t})$, so that by the induction hypothesis, we have both

$$\text{Ax} \cup \text{EQ} \vdash \bar{t}^* \approx \bar{t}^*(\nu/[\gamma, \beta, \bar{t}^*_\nu, a]) \tag{1.43}$$

and

$$\text{Ax} \cup \text{EQ} \vdash \bar{t}^*(\nu/[\gamma, \beta, t^*_\nu, a]) \approx \bar{t}^*(\nu/[\alpha, \gamma, [\gamma, \beta, t^*_\nu, a], a']) \tag{1.44}$$

where a and a' are distinct new variables.

One may show that

(1.45) $\quad \mathrm{Ax} \vdash [\alpha, \gamma, [\gamma, \beta, \bar{t}_\nu, a], a'] \approx [\alpha, \gamma, [\gamma, \beta, [\alpha, \beta, \bar{t}_\nu, a''], a], a']$.

But $[\alpha, \beta, \bar{t}_\nu, a''] = t^*_\nu$, and, by the induction hypothesis,

(1.46) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash \bar{t} \approx \bar{t}(\nu/[\alpha, \gamma, [\gamma, \beta, \bar{t}_\nu, a], a'])$.

Combining (1.43)–(1.46), we see that $\mathrm{Ax} \cup \mathrm{EQ} \vdash \bar{t} \approx t^*$, i.e.,

$$\mathrm{Ax} \cup \mathrm{EQ} \vdash \bar{t} \approx \bar{t}(\nu/[\alpha, \beta, \bar{t}_\nu, a''])$$

completing the proof of this case.

Lastly, suppose $(\alpha \approx \beta)$ follows from a congruence rule: say, for example, that $\sigma \in \Sigma_2$ and $\alpha \approx \sigma(\alpha_1, \alpha_2)$, $\beta \approx \sigma(\beta_1, \beta_2)$ and both $\alpha_1 \approx \beta_1$ and $\alpha_2 \approx \beta_2$ are provable from $S^+(\nu, \bar{t}) \cup \mathrm{EQ}$ in at most k steps.

Let $t' = \bar{t}(\nu/[\sigma(\alpha_1, \alpha_2), \sigma(\beta_1, \beta_2), \bar{t}_\nu, a])$. We must show $\mathrm{Ax} \cup \mathrm{EQ} \vdash \bar{t} \approx t'$. First we apply the induction assumption $R(k)$ to t'.

(1.47) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash t' \approx t'(\nu/[\alpha_1, \beta_1, [\sigma(\alpha_1, \alpha_2), \sigma(\beta_1, \beta_2), \bar{t}_\nu, a], a'])$.

By the congruence axiom (1.2.11)(b)

(1.48)
$$\begin{aligned} \mathrm{Ax} \vdash & [\alpha_1, \beta_1, [\sigma(\alpha_1, \alpha_2), \sigma(\beta_1, \beta_2), \bar{t}_\nu, a], a'] \\ & \approx [\alpha_1, \beta_1, [\sigma(\beta_1, \alpha_2), \sigma(\beta_1, \beta_2), \bar{t}_\nu, a], a']. \end{aligned}$$

Let $t'' = \bar{t}(\nu/[\sigma(\beta_1, \alpha_2), \sigma(\beta_1, \beta_2), \bar{t}_\nu, a])$. Since $S(\nu, t'') = S(\nu, \bar{t})$, the induction assumption $R(k)$ implies

(1.49) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash t'' \approx t''(\nu/[\alpha_1, \beta_1, t''_\nu, a'])$.

Thus, by (1.47), (1.48) and (1.49),

(1.50) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash t' \approx t''$.

We apply a similar argument to t'':

(1.51) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash t'' \approx t''(\nu/[\alpha_2, \beta_2, [\sigma(\beta_1, \alpha_2), \sigma(\beta_1, \beta_2), \bar{t}_\nu, a], a'])$.

Again by (1.2.11)(b), and (1.51),

(1.52) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash t'' \approx t''(\nu/[\alpha_2, \beta_2, [\sigma(\beta_1, \beta_2), \sigma(\beta_1, \beta_2), \bar{t}_\nu, a], a'])$.

Now apply the equal premise axiom to the right-hand tree in (1.52):

(1.53) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash t'' \approx t''(\nu/[\alpha_2, \beta_2, \bar{t}_\nu, a'])$.

But $t''(\nu/[\alpha_2, \beta_2, \bar{t}_\nu, a']) = \bar{t}(\nu/[\alpha_2, \beta_2, \bar{t}_\nu, a'])$ and by the induction hypothesis $R(k)$ applied to $\bar{t}$,

(1.54) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash \bar{t} \approx \bar{t}(\nu/[\alpha_2, \beta_2, \bar{t}_\nu, a], a')$.

By (1.50) and (1.54) we have shown $\mathrm{Ax} \cup \mathrm{EQ} \vdash \bar{t} \approx t'$, completing the induction proof. Thus $P(n+1)$, Subcase 1 is proved.

Subcase 2. $i = 4$. $S(\mu, t) \not\models x \approx y$ but $S(\mu 4, t) \models x \approx y$. Since $S(\mu 4, t)$ is $S(\mu, t) \cup \{u \neq v\}$, the only way both of the hypotheses can hold is (see the Appendix (A.2)) if

(1.55) $\quad S(\mu, t) \overset{\circ}{\models} u \approx v$.

We want to show

(1.56) $\quad \mathrm{Ax} \cup \mathrm{EQ} \vdash t \approx t(\mu/[u, v, t_3, [x, y, t_4, a]])$.

Let $t' = t(\mu/t_3)$. By the induction hypothesis $P(n)$,

$$\text{Ax} \cup \text{EQ} \vdash t' \approx t'(\mu/[u, v, t_3, a]). \tag{1.57}$$

First substitute t_4 for a in (1.57).

$$\text{Ax} \cup \text{EQ} \vdash t' \approx t. \tag{1.58}$$

Second, substitute $[x, y, t_4, a]$ for a in (1.57).

$$\text{Ax} \cup \text{EQ} \vdash t' \approx t'(\mu/[u, v, t_3, [x, y, t_4, a]]). \tag{1.59}$$

But $t'(\mu/[u, v, t_3, [x, y, t_4, a]])$ is $t(\mu/[u, v, t_3, [x, y, t_4, a]])$. Thus the proof of (1.56) is complete by (1.58).

The proof of $P(n+1)$ is complete.

Proof of $Q(n+1)$. We are assuming $\nu = \mu i$, $|\mu| = n$ and t is a reduced tree containing ν such that $S(\nu, t) \overset{\circ}{\models} x \not\approx y$. We want to prove

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\nu/[x, y, a, t_\nu]).$$

Recall that $t_\mu = [u, v, t_3, t_4]$.

Case A. $S(\mu, t) \overset{\circ}{\models} x \not\approx y$. This case is easy and uses the same arguments as Case A of $P(n+1)$.

Case B. $S(\mu, t) \overset{\circ}{\not\models} x \not\approx y$, but $S(\nu, t) \overset{\circ}{\models} (x \not\approx y)$.

Subcase 1. $i = 4$. Thus $S(\mu, t)$, $u \not\approx v \overset{\circ}{\models} x \not\approx y$. In this case, we must prove

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\mu/[u, v, t_3, [x, y, a, t_4]]). \tag{1.60}$$

Note that $S(\mu, t)$, $x \approx y \overset{\circ}{\models} u \approx v$. Let $\bar{t}$ be $t(\mu/[x, y, t_\mu, t_\mu])$.

By the induction assumption $P(n+1)$ applied to $\bar{t}$ and $\mu 3$.

$$\text{Ax} \cup \text{EQ} \vdash \bar{t} \approx \bar{t}(\mu 3/[u, v, t_\mu, a]). \tag{1.61}$$

But

$$\bar{t}(\mu 3/[u, v, t_\mu, a]) = t(\mu/[x, y, [u, v, t_\mu, a], t_\mu]).$$

Also $[u, v, t_\mu, a]$ is $[u, v, [u, v, t_3, t_4], a]$. By the axiom of positive redundancy,

$$\text{Ax} \cup \text{EQ} \vdash [u, v, t_\mu, a] \approx [u, v, t_3, a], \tag{1.62}$$

so that, by (1.61)

$$\text{Ax} \cup \text{EQ} \vdash \bar{t} \approx t(\mu/[x, y, [u, v, t_3, a], [u, v, t_3, t_4]]). \tag{1.63}$$

But, applying Proposition 1.3(ii), we have

$$\text{Ax} \cup \text{EQ} \vdash \bar{t} \approx t(\mu/[u, v, t_3, [x, y, a, t_4]]). \tag{1.64}$$

The proof of (1.60) is completed by the observation that

$$\text{Ax} \vdash t \approx \bar{t} \tag{1.65}$$

by the equal conclusion axiom.

Subcase 2. $i = 3$. In this case, $S(\mu, t)$, $u \approx v \overset{\circ}{\models} x \not\approx y$. Then (see the Appendix A.3) there is some negative equation $u' \not\approx v'$ in $S(\mu, t)$ such that

$$S^+(\mu, t), u \approx v, u' \not\approx v \overset{\circ}{\models} x \not\approx y. \tag{1.66}$$

Let μ_0 be the immediate predecessor of μ (which must exist) and *assume*

$$t_{\mu_0} \text{ is } [u', v', L, [u, v, t_3, t_4]] \tag{1.67}$$

(so that $\mu = \mu_0 4$ and $\nu = \mu_0 43$). We want to show

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\mu_0/[u', v', L, [u, v, [x, y, a, t_3], t_4]]). \tag{1.68}$$

First we apply Proposition 1.3(ii) to t_{μ_0}:

$$Ax \vdash t \approx t(\mu_0/[u, v, [u', v', L, t_3], [u', v', L, t_4]]). \tag{1.69}$$

By subcase 1 of Case B applied to the right-hand side of (1.69) and $\mu_0 34$,

$$\text{Ax} \cup \text{EQ} \vdash t \approx t(\mu_0/[u, v, [u', v', L, [x, y, a, t_3]], [u', v', L, t_4]]). \tag{1.70}$$

If we apply Proposition 1.3 and the equal conclusion axiom to the right-hand side of (1.70), we obtain the statement (1.68).

We indicate why we may make the assumption (1.67). Informally, if the negative equation $u' \not\approx v'$ used in (1.66) is not determined by labels on the premise leaves $\mu_0 1$ and $\mu_0 2$ (where $\mu = \mu_0 i$), then we may use the axiom of premise interchange to "put these labels there". More precisely, the following statement is true.

PROPOSITION 1.71. *Let $t = [u', v', L, R]$ be a reduced tree and suppose that ν is a nonpremise node of R. Write R_ν $(= t_{4\nu})$ as $[u, v, t_3, t_4]$. Then there is a reduced tree t' and a nonpremise vertex γ of t' with $|\gamma| = |4\nu|$ such that:*

a) $\text{Ax} \cup \text{EQ} \vdash t'_\gamma = t_{4\nu}$,
b) $S(\gamma, t') = S(4\nu, t)$,
c) $\gamma = \gamma' 4$, *for some* γ',
d) $t'_{\gamma'} = [u', v', t'_3, t'_\gamma]$.

The straightforward proof of Proposition 1.71 by induction on $|\nu|$ is omitted.

The proof of $Q(n+1)$ and hence of the entire insertion lemma is complete.

2. The class $K^+(\perp)$. Let $K(\perp)$ be the collection of all "pointed sets," i.e. all sets A having a distinguished element (always denoted $\perp$). Let $A \in K(\perp)$ and define $\kappa : A^4 \to A$ by:

$$\kappa(x, y, u, v) = \begin{cases} \perp & \text{if either } x = \perp \text{ or } y = \perp; \quad \text{otherwise} \\ u & \text{if } x = y, \\ v & \text{if } x \neq y. \end{cases}$$

Let A^+ be A augmented by the function κ and let $K^+(\perp) = \{A^+ : A \in K(\perp)\}$. Here $\perp$ models "undefined" and κ models the test "if x and y are both defined then u if $x = y$ and v otherwise; undefined otherwise." In this section we will axiomatize the identities valid in the class $K^+(\perp)$. The outline of the argument for this case is similar to that given in § 1, but it seems that many of the details are necessarily different and require new proofs. Note that we cannot use the argument of § 1, since at a crucial point, namely before (1.26), we used the equal conclusion axiom (1.2.2), which is not valid when an "undefined" element is present. (However, see Lemma 2.18.)

The set of terms $T^+(\perp)$ formed from the countably infinite set V of "variables", the constant symbol $\perp$ and the symbol κ (of rank 4) is defined in the standard way. Again, for convenience, the notation $[x, y, u, v]$ is used in place of $\kappa(x, y, u, v)$.

DEFINITION 2.1. For all $a, b, c, x, y, z, u, v, w$ in $T^+(\perp)$, the following equations belong to $\text{Ax}(\perp)$.

($\perp$-axioms):

(2.1.1) $[\perp, y, u, v] \approx \perp$,

(2.1.2) $[x, y, \perp, \perp] \approx \perp$,

(2.1.3) $[x, x, x, \perp] \approx x$,

(2.1.4) $[x, x, u, v] \approx [x, x, u, \perp]$,

(2.1.5) $[x, y, u, u] \approx [x, x, [y, y, u, \perp], \perp]$,

(2.1.6) $[x, y, u, v] \approx [x, y, u, [x, x, v, \perp]]$.

(2.1.7) The "common axioms", i.e., each of the axioms (1.2.3)–(1.2.9), with the present meaning of the letters $x, y, \cdots$, etc. (i.e. premise commutativity, conclusion replacement, premise replacement, the redundancy axioms, premise interchange and simplification).

We list some useful corollaries of these axioms.

PROPOSITION 2.2. *For all* $x, y, z, a, a_1, a_2, b, c, t_3, t_4$ *in* $T^+(\perp)$ *the following equations are provable (in standard equational logic) from* Ax ($\perp$).

(2.2.1) (*positive conclusion insertion*)

$$[x, y, [u, v, a, b], c] \approx [x, y, [u, v, [x, y, a, z], b], c],$$

(2.2.2) (*negative conclusion insertion*)

$$[x, y, [u, v, a, b], c] \approx [x, y, [u, v, a, [x, y, b, z]], c],$$

(2.2.3) (*positive alternative insertion*)

$$[x, y, a, [u, v, b, c]] \approx [x, y, a, [u, v, [x, y, z, b], c]],$$

(2.2.4) (*negative alternative insertion*)

$$[x, y, a, [u, v, b, c]] \approx [x, y, a, [u, v, b, [x, y, z, c]]],$$

(2.2.5) (*dual of* (2.1.6))

$$[x, y, u, v] \approx [x, y, [x, x, u, \perp], v],$$

(2.2.6) (*positive transitivity*)

$$[x, u, [y, v, [u, v, t_3, t_4], a_2], a_1] \approx [x, u, [y, v, [u, v, [x, y, t_3, a], t_4], a_2], a_1],$$

(2.2.7) (*negative transitivity*)

$$[x, u, a_1, [y, v, [u, v, t_3, t_4], a_2]] \approx [x, u, a_1, [y, v, [u, v, [x, y, a, t_3], t_4], a_2]].$$

Proof. We will prove only (2.2.1), (2.2.2) and (2.2.7).

Proof of (2.2.1). For any α in $T^+(\perp)$,

$$\text{Ax}\,(\perp) \vdash [x, y, [u, v, \alpha, b], c] \approx [x, y, [x, y, [u, v, \alpha, b], [u, v, \alpha, b]], c]$$

by the positive redundancy axiom. Let L denote the left-hand tree, and apply premise interchange to the right-hand tree.

$$\text{Ax}\,(\perp) \vdash L \approx [x, y, [u, v, [x, y, \alpha, \alpha], [x, y, b, b]], c].$$

Now let α be the tree $[x, y, a, z]$, and apply the positive and negative redundancy axioms:

$$\mathrm{Ax}\,(\bot)\vdash L \approx [x, y, [u, v, [x, y, \alpha, z], [x, y, b, b]], c].$$

Lastly, again use the premise interchange and positive redundancy axioms.

$$\mathrm{Ax}\,(\bot)\vdash L \approx [x, y, [u, v, a, b], c]. \tag{2.3}$$

Remembering that α in L is $[x, y, a, z]$, we may rewrite (2.3) as

$$\mathrm{Ax}\,(\bot)\vdash [x, y, [u, v, [x, y, a, z], b], c] \approx [x, y, [u, v, a, b], c]; \tag{2.4}$$

i.e. (2.2.1) is proved. (In essence, this argument is used in [10] to simplify McCarthy's axioms [9].)

The assertion (2.2.2) is proved using (2.2.1).

Proof of (2.2.2). By (2.2.1), we have

$$\mathrm{Ax}\,(\bot)\vdash [x, y, [u, v, a, [x, y, b, z]], c] \approx [x, y, [u, v, [x, y, a, z], [x, y, b, z]], c]. \tag{2.5}$$

Then (2.2.2) follows from using the premise interchange and the redundancy axioms on the right-hand tree.

Assertion (2.2.3) is proved similarly to (2.2.1), and (2.2.4) follows from (2.2.3) in the same way that (2.2.2) follows from (2.2.1). Using the premise replacement and positive redundancy axioms, we may easily prove (2.2.5).

The two assertions (2.2.6) and (2.2.7) are proved in a similar way, and we prove only the later one.

Proof of (2.2.7). First we observe that by positive alternative insertion, for any $\alpha \in T^+(\bot)$

$$\mathrm{Ax}\,(\bot)\vdash [x, u, a_1, [v, y, \alpha, a_2]] \approx [x, u, a_1, [v, y, [x, u, a_3, \alpha], a_2]]. \tag{2.6}$$

Also, by the same axiom,

$$\mathrm{Ax}\,(\bot)\vdash [x, u, a_3, [u, v, t_3, t_4]] \approx [x, u, a_3, [u, v, [x, u, a, t_3], t_4]]. \tag{2.7}$$

Now apply (2.6) when α is $[u, v, t_3, t_4]$:

$$\begin{aligned}\mathrm{Ax}\,(\bot)\vdash [x, u, a_1, [v, y, [u, v, t_3, t_4], a_2]] \\ \approx [x, u, a_1, [v, y, [x, u, a_3, [u, v, t_3, t_4]], a_2]],\end{aligned} \tag{2.8}$$

and by (2.7),

$$\begin{aligned}\mathrm{Ax}\,(\bot)\vdash [x, u, a_1, [v, y, [u, v, t_3, t_4], a_2]] \\ \approx [x, u, a_1, [v, y, [x, u, a_3, [u, v, [x, u, a, t_3], t_4]], a_2]].\end{aligned} \tag{2.9}$$

By several uses of premise replacement,

$$\begin{aligned}\mathrm{Ax}\,(\bot)\vdash [x, u, a_1, [v, y, [x, u, a_3, [u, v, [x, u, a, t_3], t_4]], a_2]] \\ \approx [x, u, a_1, [v, y, [x, u, a_3, [u, v, [x, y, a, t_3], t_4]], a_2]].\end{aligned} \tag{2.10}$$

Again, using positive alternative insertion, the right-hand term in (2.10) is provably equal to $[x, u, a_1, [v, y, [u, v, [x, y, a, t_3], t_4], a_2]]$, completing the proof.

Using the premise-simplification axiom, *we may henceforth assume that every term t in $T^+(\bot)$ is a simple-premise term*; i.e. for every subterm of t of the form $\kappa(x, y, u, v)$, there is no occurrence of κ in x or y. We identify simple-premise terms

in $T^+(\perp)$ with those partial functions

$$t:[4]^* \to V \cup \{\kappa, \perp\}$$

having a finite nonempty prefix-closed domain such that, for all $\nu \in [4]^*$, if $\nu t = \kappa$, then νit is defined for each $i \in [4]$; if $\nu t \in V \cup \{\perp\}$ then νit is not defined for any $i \in [4]$; furthermore, if $\nu t = \kappa$, then $\nu it \in V \cup \{\perp\}$ for $i = 1$ and 2. As before, for x, y in $V \cup \{\perp\}$, we write $[x, y, t_3, t_4]$ for the term (or "tree") t such that, $\lambda t = \kappa$, $1t = x$, $2t = y$ and $ivt = \nu t_i$, $i = 3, 4$, $\nu \in [4]^*$.

We may make one further preliminary simplification.

DEFINITION 2.11. A tree t in $T^+(\perp)$ is a *total premise* tree if for every subterm $[x, y, t_3, t_4]$ of t, neither x nor y is the symbol $\perp$.

PROPOSITION 2.12. *For every t in $T^+(\perp)$, there is a (simple and) total premise tree t' such that*

$$\mathrm{Ax} \vdash t \approx t'.$$

The proof is immediate from the axioms (2.1.1) and premise commutativity.

For total premise trees t, we may define the sets $S(\nu, t)$ as before.

DEFINITION 2.13. Suppose t is a total premise tree and $\nu \in \{3, 4\}^*$ is a nonpremise vertex of t. Then

$$S(\lambda, t) = \varnothing;$$

if $\nu = i\mu$, and $t = [x, y, t_3, t_4]$, then

$$S(3\mu, t) = \{x \approx y\} \cup S(\mu, t_3),$$

$$S(4\mu, t) = \{x \not\approx y\} \cup S(\mu, t_4).$$

We now define the appropriate version of Path (h, t).

DEFINITION 2.14. Let t be a total premise tree and let $A \in K^+(\perp)$. Suppose $h: T^+(\perp) \to A$ is a homomorphism. Then Path (h, t) is an element of $\{3, 4, \perp\}^*$ defined as follows:

if λt is in $V \cup \{\perp\}$, Path $(h, t) = \lambda$;

if t is $[x, y, t_3, t_4]$ then

$$\text{Path } (h, t) = \begin{cases} \perp & \text{if } xh \text{ or } yh \text{ is } \perp; \quad \text{otherwise} \\ 3 & \text{Path } (h, t_3) \text{ if } xh = yh, \\ 4 & \text{Path } (h, t_4) \text{ if } xh \neq yh. \end{cases}$$

For example, if $u, v, a, b, c \in V$ and $uh \neq \perp$, $vh = \perp$, then when t is $[u, u, [u, v, a, b], c]$, Path $(h, t) = 3\perp$.

Remark 2.15. Let t be a total premise tree and let $A \in K^+(\perp)$. Suppose that $h: T^+(\perp) \to A$ is a homomorphism. If $\nu = \text{Path } (h, t)$, then th, the value of h on t, is $(\nu t)h$ if $\nu \in \{3, 4\}^*$ (i.e. ν does *not* end with the symbol $\perp$) and is $\perp$ otherwise.

Remark 2.16. For any homomorphism $h: T^+(\perp) \to A$, $A \in K^+(\perp)$ and any total premise tree t, h will satisfy each signed equation in $S(\nu, t)$ iff ν is a prefix of Path (h, t).

Let α be a signed equation between variables in V, say α is $x \approx y$ or $x \not\approx y$. Then we let $\text{var } (\alpha) = \{x, y\}$. If X is a set of such signed equations, let $\text{var } X = \bigcup(\text{var } (\alpha): \alpha \in X)$.

DEFINITION 2.17. Suppose ν is a nonpremise node of the total premise tree t. The set $D(\nu, t)$ (of variables *defined* on the path to ν) is defined by

$$D(\nu, t) = \text{var } S(\nu, t).$$

The following two facts play the role of the insertion lemma of § 1. The meaning of t_ν and $t(\nu/t')$ is the same here as in § 1.

EQUAL PREMISE INSERTION LEMMA 2.18. *Let t be a total premise tree and let ν be a nonpremise node of t. If x is any variable in $D(\nu, t)$, then*

$$\mathrm{Ax}\,(\bot)\vdash t\approx t(\nu/[x, x, t_\nu, \bot]).$$

DISTINCT PREMISE INSERTION LEMMA 2.19. *Let t, ν be as in Lemma 2.18. Let x and y be distinct variables. Then*

$$\text{if } S(\nu, t)\overset{\circ}{\models} x\approx y, \text{ then } \mathrm{Ax}\,(\bot)\vdash t\approx t(\nu/[x, y, t_\nu, a])$$

and

$$\text{if } S(\nu, t)\overset{\circ}{\models} x\not\approx y, \text{ then } \mathrm{Ax}\,(\bot)\vdash t\approx t(\nu/[x, y, a, t_\nu])$$

where a is a variable distinct from x, y which does not occur in t.

The meaning of $X\overset{\circ}{\models}\alpha$, where X is a set of signed equations between variables and α is some signed equation, is as before: $X\overset{\circ}{\models}\alpha$ if for any $A\in K(\bot)$ and any function $h: V\to A$, h satisfies α whenever h satisfies every element of X. (Here the element $\bot$ plays no role.) The reason for making a distinction between the cases "x and y" and "x and y are distinct" is the following. For any set X of signed equations, and any variable x, $X\overset{\circ}{\models} x\approx x$. However, if t is the total premise tree $[u, v, a, b]$ and x does not occur in t, then $S(\lambda, t)=\varnothing\overset{\circ}{\models} x\approx x$ but it is not true that $\models t\approx[x, x, t, \bot]$.

We again postpone the proofs of the insertion lemmas. First we derive some corollaries of these lemmas, including the completeness of Ax $(\bot)$.

Note that we may reinterpret the insertion lemmas as "deletion lemmas" in certain cases. In the notation of Lemma 2.18, if t_ν is the tree $[x, y, t_3, t_4]$ and either x and y are distinct or $x\in D(\nu, t)$ and $S(\nu, t)\overset{\circ}{\models} x\approx y$, then $\mathrm{Ax}\,(\bot)\vdash t\approx t(\nu/t_3)$. Similarly, if $S(\nu, t)\overset{\circ}{\models} x\not\approx y$, then $\mathrm{Ax}\,(\bot)\vdash t\approx t(\nu/t_4)$.

DEFINITION 2.20. Let t be a total premise tree and suppose ν is a nonpremise vertex of t. Then ν is *accessible* in T if ν is a prefix of Path (h, t) for some homomorphism $h: T^+(\bot)\to A$, $A\in K^+(\bot)$. The tree t is accessible if every nonpremise vertex of t is accessible.

DEFINITION 2.21. Let ν be a nonpremise node of the total premise tree t. We say ν is *sufficiently accessible* if $\nu=\mu 4$, for some accessible nonpremise node μ of t and t_μ has the form $[x, x, t_3, \bot]$, for some t_3 and some variable x *not in* $D(\mu, t)$ (in particular, the label of ν is $\bot$). The tree t is *sufficiently accessible* if every nonpremise vertex of t is either accessible or sufficiently accessible.

The following fact follows from the two insertion lemmas.

PROPOSITION 2.22. *For any $t\in T^+(\bot)$ there is a sufficiently accessible (simple-) total-premise tree t' such that*

$$\mathrm{Ax}\,(\bot)\vdash t\approx t'.$$

Proof. Let t be a fixed total premise tree. Let ν be an inaccessible nonpremise vertex of shortest length. Then $\nu=\mu i$ for some $i\in\{3, 4\}$ and $\mu\in\{3, 4\}^*$. Write t_μ as $[x, y, t_3, t_4]$. There are three possibilities:

(P1) $x=y$;

(P2) x and y are distinct and $\nu=\mu 3$ (so $S(\mu, t)\overset{\circ}{\models} x\not\approx y$);

(P3) x and y are distinct and $\nu=\mu 4$ (so $S(\mu, t)\overset{\circ}{\models} x\approx y$).

In the case (P1), ν is necessarily $\mu 4$ (since μ and hence $\mu 3$ are accessible). First we apply axiom 2.1.4 to remove t_4: $\mathrm{Ax}\vdash[x, x, t_3, t_4]\approx[x, x, t_3, \bot]$. Now, either $x\in D(\mu, t)$ or not. If not, there is nothing left to do. If $x\in D(\mu, t)$, let $t'=t(\mu/t_3)$. Then, by the

equal premise insertion lemma,

$$\mathrm{Ax} \vdash t' \approx t(\mu/[x, x, t_3, \bot]).$$

Thus, we have shown how to remove from t one inaccessible vertex violating Definition 2.21.

In cases (P2) and (P3) we argue as in § 1 to show how, using the insertion lemma 2.19, at least one inaccessible vertex is removable from t. By induction on the number of such vertices, the proof is complete.

We now describe a construction analogous to that of $\tau_{\tilde{M}}$ in § 1. Let $\tilde{\Delta} = v_1, \cdots, v_m$ be any finite sequence of variables. Let b be a variable not occurring in $\tilde{\Delta}$. We define $\tau_{\tilde{\Delta}}$ to be the total premise tree

$$\tau_{\tilde{\Delta}} = [v_1, v_1, [v_2, v_2, \cdots, [v_m, v_m, b, \bot], \bot], \cdots, \bot].$$

Thus, in the case $m = 2$,

$$\tau_{\tilde{\Delta}} = [v_1, v_1, [v_2, v_2, b, \bot], \bot].$$

When $m = 0$, $\tau_{\tilde{\Delta}}$ is just the variable b.

Let $\tilde{M}$ be any finite sequence of the form (1.19) (where now each α_i is an equation $s_i \approx s_i'$ between *variables* in V). Then $\tau_{\tilde{M}}$ was defined in (1.21).

Lastly, given $\tilde{\Delta}$ and $\tilde{M}$ as above, we write $\tau_{\tilde{\Delta},\tilde{M}}$ for the tree obtained from $\tau_{\tilde{\Delta}}$ by replacing the variable b by $\tau_{\tilde{M}}$. For example, if $\tilde{\Delta} = v_1, v_2$ and $\tilde{M}$ is $(s_1 \approx s_1'; 3), (s_2 \approx s_2'; 4)$, then

$$\tau_{\tilde{\Delta},\tilde{M}} = [v_1, v_1, [v_2, v_2, [s_1, s_1', [s_2, s_2', a_2, b], a_1], \bot], \bot].$$

With $\nu_{\tilde{M}}$ defined as in (1.22) and with $\nu_{\tilde{\Delta}}$ defined as the vertex $3^m = 33 \cdots 3$ (m times, when $\tilde{\Delta}$ has length m) we note that the variable b in $\tau_{\tilde{\Delta},\tilde{M}}$ is the label of $\nu_{\tilde{\Delta}}\nu_{\tilde{M}}$. We write "$\tau_{\tilde{\Delta},\tilde{M}}(t)$" instead of $\tau_{\tilde{\Delta},\tilde{M}}(\nu_{\tilde{\Delta}}\nu_{\tilde{M}}/t)$, the result of replacing b by t in $\tau_{\tilde{\Delta},\tilde{M}}$.

For any finite set Δ of variables, an *arrangement* of Δ is any finite sequence $\tilde{\Delta}$ listing, perhaps with repetitions, all of the elements of Δ.

Let Δ be a finite set of variables and let M be a finite set of signed equations between variables. For terms t, t' in $T^+(\bot)$, we write

$$\Delta, M \overset{\circ}{\models} t \approx t' \tag{2.23}$$

if for every $A \in K^+(\bot)$ and every homomorphism $h: T^+(\bot) \to A$, $th = t'h$ whenever $vh \neq \bot$, all $v \in \Delta$ and h satisfies each signed equation in M.

The following theorem plays the role of Theorem 1.23 in this context.

THEOREM 2.24. *Suppose that Δ is a finite set of variables and M is a finite set of signed equations between variables. Let t and t' be total premise trees. Let $\tilde{\Delta}$ and $\tilde{M}$ be any arrangements of Δ and M respectively. If*

$$\Delta, M \overset{\circ}{\models} t \approx t' \tag{2.25}$$

then

$$\mathrm{Ax}\,(\bot) \vdash \tau_{\tilde{\Delta},\tilde{M}}(t) \approx \tau_{\tilde{\Delta},\tilde{M}}(t'). \tag{2.26}$$

Note that when Δ and M are both empty, Theorem 2.24 reduces to the completeness theorem for Ax $(\bot)$. The proof of Theorem 2.24 is by induction on the number of occurrences of κ in t. Since the basis step is very similar to that of Theorem 1.23, we will give only the:

Proof of the induction step. Write t as $[x, y, t_3, t_4]$. First we will show why (2.25) implies that

$$\text{Ax}(\bot) \vdash t' \approx [x, y, t'_3, t'_4] \tag{2.27}$$

for some total-premise trees t'_3 and t'_4.

Let ν be any accessible nonpremise leaf of $\tau_{\Delta,\tilde{M}}(t')$ of the form $\nu_{\Delta}\nu_{\tilde{M}}\nu'$, where ν' is a leaf of t'. Suppose the label of ν is z. We claim that both x and y belong to $X = D(\nu, \tau_{\Delta,\tilde{M}}(t')) \cup \{z\}$. Indeed suppose that x doesn't belong to X. Since ν is accessible, there is some homomorphism $h: T^+(\bot) \to A$, $A \in K^+(\bot)$ such that $\nu = \text{Path}(h, \tau_{\Delta,\tilde{M}}(t'))$. We may further assume that $vh \neq \bot$ for every $v \in X$. Let h' agree with h except perhaps on the variable x, and let $xh' = \bot$. Then both h and h' satisfy $\Delta \cup M$, since $\nu_{\Delta}\nu_{\tilde{M}}$ is an initial segment of ν; and $t'h = t'h' = zh$, by construction. But $th' = [x, y, t_3, t_4]h' = \bot$, since $xh' = \bot$. This contradicts the hypothesis (2.25).

Now, by the equal premise insertion lemma,

$$\text{Ax}(\bot) \vdash \tau_{\Delta,\tilde{M}}(t') \approx \tau_{\Delta,\tilde{M}}(t')(\nu/[x, x, [y, y, z, \bot], \bot]).$$

But, by axiom (2.1.5)

$$\text{Ax}(\bot) \vdash [x, x, [y, y, z, \bot], \bot] \approx [x, y, z, z].$$

Thus, we have shown that any accessible nonpremise leaf of t' in $\tau_{\Delta,\tilde{M}}(t')$ labeled z may be replaced by $[x, y, z, z]$. Any inaccessible nonpremise leaf of t' in $\tau_{\Delta,\tilde{M}}(t')$ is labeled $\bot$, since $\tau_{\Delta,\tilde{M}}(t')$ is sufficiently accessible. By axiom (2.1.2),

$$\text{Ax}(\bot) \vdash \bot \approx [x, y, \bot, \bot].$$

Hence we may now assume that every nonpremise leaf ν of t' in $\tau_{\Delta,\tilde{M}}(t')$ has been replaced by $[x, y, z, z]$, where z is the label of ν. We now apply the axiom of premise interchange as often as necessary to show

$$\text{Ax}(\bot) \vdash \tau_{\Delta,\tilde{M}}(t') \approx \tau_{\Delta,\tilde{M}}(t''),$$

where $t'' = [x, y, t'_3, t'_4]$, for some t'_3, t'_4.

The proof from this stage is short. We must show that $\tau_{\Delta,\tilde{M}}[x, y, t_3, t_4] \approx \tau_{\Delta,\tilde{M}}[x, y, t'_3, t'_4]$ is provable from Ax $(\bot)$, assuming

$$\Delta, M \overset{\circ}{\models} [x, y, t_3, t_4] \approx [x, y, t'_3, t'_4]. \tag{2.28}$$

Now (2.28) implies that

$$\Delta \cup \{x, y\}, M \cup \{x \approx y\} \overset{\circ}{\models} t_3 \approx t'_3 \tag{2.29}$$

and

$$\Delta \cup \{x, y\}, M \cup \{x \not\approx y\} \overset{\circ}{\models} t_4 \approx t'_4. \tag{2.30}$$

Let $\Delta_1 = \Delta \cup \{x, y\}$, $M_1 = M \cup \{x \approx y\}$, $M_2 = M \cup \{x \not\approx y\}$. Then, by the induction hypothesis.

$$\text{Ax}(\bot) \vdash \tau_{\Delta_1,\tilde{M}_1}(t_3) \approx \tau_{\Delta_1,\tilde{M}_1}(t'_3) \tag{2.31}$$

and

$$\text{Ax}(\bot) \vdash \tau_{\Delta_1,\tilde{M}_2}(t_4) \approx \tau_{\Delta_1,\tilde{M}_2}(t'_4). \tag{2.32}$$

Now, by the insertion lemmas,

$$\text{Ax}(\bot) \vdash \tau_{\Delta,\tilde{M}}[x, y, t_3, t_4] \approx \tau_{\Delta,\tilde{M}}[x, y, \tau_{\Delta_1,\tilde{M}_1}(t_3), \tau_{\Delta_1,\tilde{M}_2}(t_4)]. \tag{2.33}$$

The same statement is true with t_i' replacing t_i, $i = 3, 4$. Hence,

$$\text{Ax}\,(\bot) \vdash \tau_{\tilde{\Delta},\tilde{M}}[x, y, t_3, t_4] \approx \tau_{\tilde{\Delta},\tilde{M}}[x, y, t_3', \tau_4'], \tag{2.34}$$

completing the proof.

To complete this section, we prove the insertion lemmas.

EQUAL PREMISE INSERTION LEMMA. *Let t be a total premise tree and let ν be a nonpremise node of t. If x is a variable in $D(\nu, t)$, then* $\text{Ax}\,(\bot) \vdash t \approx t(\nu/[x, x, t_\nu, \bot])$.

Proof. By induction on $|\nu|$. When $|\nu| = 0$, $D(\nu, t)$ is empty, so there is nothing to prove. Now assume that ν is μi, for some $\mu \in \{3, 4\}^*$, and some $i \in \{3, 4\}$. Write t_μ as $[u, v, t_3, t_4]$.

Case 1. $x \in D(\mu, t)$. By the induction hypothesis,

$$\text{Ax}\,(\bot) \vdash t \approx t(\mu/[x, x, [u, v, t_3, t_4], \bot]), \tag{2.35}$$

if i is 3, by (2.2.1),

$$\text{Ax}\,(\bot) \vdash t \approx t(\mu/[x, x, [u, v, [x, x, t_3, \bot], t_4], \bot]), \tag{2.36}$$

and by the induction assumption (in its "deletion" version), the right-hand tree in (2.36) is provably equal to $t(\mu/[u, v, [x, x, t_3, \bot], t_4])$, which is $t(\nu/[x, x, t_\nu, \bot])$ when $\nu = \mu 3$. When $i = 4$, we use negative conclusion insertion (2.2.2) in place of (2.1.1), and argue in the same way.

Case 2. $x \notin D(\mu, t)$. In this case either u or v is x. We may assume, by the axiom of premise commutativity that u is x, so that t_μ is $[x, v, t_3, t_4]$.

First suppose $\nu = \mu 3$. Then, by (2.2.5),

$$\text{Ax}\,(\bot) \vdash t \approx t(\mu/[x, v, [x, x, t_3, \bot], t_4]). \tag{2.37}$$

When ν is $\mu 4$, we use axiom (2.1.6) to show

$$\text{Ax}\,(\bot) \vdash t \approx t(\mu/[x, v, t_3, [x, x, t_4, \bot]]). \tag{2.38}$$

But both (2.37) and (2.38) show $\text{Ax}\,(\bot) \vdash t \approx t(\nu/[x, x, t_\nu, \bot])$, completing the proof.

DISTINCT PREMISE INSERTION LEMMA. *Let x and y be distinct variables and let t, ν be as in Lemma* 2.18. *If*

$$S(\nu, t) \overset{\circ}{\models} x \approx y, \text{ then } \text{Ax}\,(\bot) \vdash t \approx t(\nu/[x, y, t_\nu, a])$$

and if

$$S(\nu, t) \overset{\circ}{\models} x \not\approx y, \text{ then } \text{Ax}\,(\bot) \vdash t \approx t(\nu/[x, y, a, t_\nu])$$

where a is a variable distinct from x and y not occurring in t.

Proof. We argue as we did for the insertion lemma in § 1. Let $P(n)$ denote the statement:

(2.39) For all variables x, y, for all total premise trees t, for all $\nu \in \{3, 4\}^*$, if $|\nu| = n$ and if ν is a nonpremise node of t and if x and y are distinct variables such that $S(\nu, t) \overset{\circ}{\models} x \approx y$, then $\text{Ax}\,(\bot) \vdash t \approx t(\nu/[x, y, t_\nu, a])$.

Let $Q(n)$ be the statement obtained from $P(n)$ by replacing "$x \approx y$" by "$x \not\approx y$" and "$[x, y, t_\nu, a]$" by "$[x, y, a, t_\nu]$". Again we prove $P(n)$ and $Q(n)$ for all nonnegative integers n by induction on n.

Basis step. $n = 0$. In this case, since $S(\nu, t) = \varnothing$ and x and y are distinct, it is impossible that $S(\nu, t) \overset{\circ}{\models} x \approx y$ or $S(\nu, t) \overset{\circ}{\models} x \not\approx y$. Indeed, when x and y are distinct, if $S(\nu, t)$ is consistent and either $S(\nu, t) \overset{\circ}{\models} x \approx y$ or $S(\nu, t) \overset{\circ}{\models} x \not\approx y$, then both x and y belong to $D(\nu, t)$ (see the Appendix, A.4(ii)). Thus both $P(0)$ and $Q(0)$ are vacuously true.

Induction step. Assume $P(n)$ and $Q(n)$. First we prove $P(n+1)$. Thus suppose $\nu = \mu i$, where $\mu \in \{3, 4\}^*$, $i \in \{3, 4\}$ and t_μ is $[u, v, t_3, t_4]$.

Case A. $S(\mu, t) \overset{\circ}{\models} x \approx y$. Then, by the assumption $P(n)$,

$$\mathrm{Ax}\,(\bot) \vdash t \approx t(\mu/[x, y, [u, v, t_3, t_4], a]).$$

Then, we use either (2.2.1) (if $\nu = \mu 3$) or (2.2.2) (if $\nu = \mu 4$) to show, for $i = 3$ or $i = 4$:

$$\mathrm{Ax}\,(\bot) \vdash t \approx t(\mu/[x, y, t_\mu, (i/[x, y, t_i, a']), a]), \tag{2.40}$$

where a and a' are distinct new variables. But again by the assumption $P(n)$, the right-hand tree in (2.40) is provably equal to $t(\nu/[x, y, t_\nu, a'])$, completing the proof of Case A.

Case B. $S(\mu, t) \overset{\circ}{\not\models} x \approx y$, but $S(\nu, t) \overset{\circ}{\models} x \approx y$.

Subcase 1. $\nu = \mu 3$, so that $S(\nu, t) = S(\mu, t) \cup \{u \approx v\}$.

There are two possibilities. Either $S(\nu, t)$ is inconsistent (in which case $S(\mu, t) \overset{\circ}{\models} u \not\approx v$) or not. If so, we argue exactly as in Case B, Subcase 1.1 in the proof of Lemma 1.14, using $Q(n)$. Otherwise, as is proved in the Appendix,

$$S(\mu, t) \overset{\circ}{\models} x \approx u \quad \text{and} \quad S(\mu, t) \overset{\circ}{\models} y \approx v$$

(or "$y \approx u$" and "$x \approx v$" follow from $S(\mu, t)$). By the induction assumption $P(n)$,

$$\mathrm{Ax}\,(\bot) \vdash t \approx t(\mu/[x, u, [y, v, [u, v, t_3, t_4], a_2], a_1]). \tag{2.41}$$

We now apply positive transitivity (2.2.6), to show

$$\mathrm{Ax}\,(\bot) \vdash t \approx t(\mu/[x, u, [y, v, [u, v, [x, y, t_3, a], t_4], a_2], a_1]). \tag{2.42}$$

But, again assuming $P(n)$, the right-hand tree in (2.42) is provably equal to $t(\mu/[u, v, [x, y, t_3, a], t_4])$, completing the proof of Subcase 1.

Subcase 2. ν is $\mu 4$, so that $S(\nu, t) = S(\mu, t) \cup \{u \not\approx v\}$. There are two possibilities. Either $S(\nu, t)$ is inconsistent (in which case $S(\mu, t) \overset{\circ}{\models} u \approx v$) or not. If so, we argue similarly to the above. If not, $S(\mu, t) \overset{\circ}{\models} x \approx y$, contradicting the hypothesis of Case B. This completes the proof of $P(n+1)$.

Proof of $Q(n+1)$. We are assuming $\nu = \mu i$ is a nonpremise node of t such that $S(\nu, t) \overset{\circ}{\models} x \not\approx y$.

Case A. $S(\mu, t) \overset{\circ}{\models} x \not\approx y$. This is similar to Case A of $P(n+1)$. We omit the details.

Case B. $S(\mu, t) \overset{\circ}{\not\models} x \not\approx y$, but $S(\nu, t) \overset{\circ}{\models} x \not\approx y$.

Subcase 1. $\nu = \mu 3$, so $S(\nu, t) = S(\mu, t) \cup \{u \approx v\}$. If $S(\nu, t)$ is inconsistent, we argue in the usual way. Otherwise, as is shown in the Appendix A.5, (up to a notational permutation)

$$S(\mu, t) \overset{\circ}{\models} x \not\approx u \quad \text{and} \quad S(\mu, t) \overset{\circ}{\models} y \approx v. \tag{2.43}$$

Then, by the assumptions $Q(n)$ and $P(n)$,

$$\mathrm{Ax}\,(\bot) \vdash t \approx t(\mu/[x, u, a_1, [y, v, [u, v, t_3, t_4], a_2]]). \tag{2.44}$$

By negative transitivity (2.2.7),

$$\mathrm{Ax}\,(\bot) \vdash t \approx t(\mu/[x, u, a_1, [y, v, [u, v, [x, y, a, t_3], t_4], a_2]]). \tag{2.45}$$

Again, assuming $P(n)$ and $Q(n)$, the right-hand tree in (2.45) is provably equal to $t(\nu/[x, y, a, t_3])$, completing the proof of Subcase 1.

Subcase 2. $\nu = \mu 4$: so $S(\nu, t) = S(\mu, t) \cup \{u \not\approx v\}$. We will omit the usual argument for the case that $S(\nu, t)$ is inconsistent. Assuming $S(\nu, t)$ is consistent, then, as shown

in the Appendix A.4 (up to a notational permutation)

$$S(\mu, t) \overset{\circ}{\models} x \approx u \quad \text{and} \quad S(\mu, t) \overset{\circ}{\models} v \approx y. \tag{2.46}$$

Thus, by the induction assumption $P(n)$

$$\mathrm{Ax}(\bot) \vdash t \approx t(\mu/[x, u, [y, v, [u, v, t_3, t_4], a_2], a_1]).$$

But one may prove similarly to the proof of (2.2.5) that

$$\begin{aligned} \mathrm{Ax}(\bot) \vdash & [x, u, [y, v, [u, v, t_3, t_4], a_2], a_1] \\ & \approx [x, u, [y, v, [u, v, t_3, [x, y, a, t_4]], a_2], a_1]. \end{aligned} \tag{2.47}$$

The rest of the argument is clear, and is omitted. The proof of $Q(n+1)$ and hence the lemma is complete.

3. Axiomatization of the class $\underline{K(3)}$. An algebra (A, κ, T) belongs to $\underline{K(3)}$ iff T is a distinguished element of A, and $\kappa: A^3 \to A$ is a function which satisfies the following condition: for all p, x, y in A

$$\kappa(p, x, y) = \begin{cases} x & \text{if } p = \mathrm{T}, \\ y & \text{if } p \neq \mathrm{T}. \end{cases} \tag{3.1}$$

Thus if p is "true", $\kappa(p, x, y)$ is x, and is y otherwise.

We let $\underline{T(3)}$ denote the algebra of terms built from a countably infinite set V of variables, the constant symbol T and the function symbol κ. As usual, we identify an element t of $T(3)$ with a partial function: this time a partial function $t: [3]^* \to V \cup \{\mathrm{T}, \kappa\}$ (see the Preliminaries). We write "$[t_1, t_2, t_3]$" instead of "$\kappa(t_1, t_2, t_3)$" for the partial function satisfying:

$$\lambda[t_1, t_2, t_3] = \kappa: \quad i\mu[t_1, t_2, t_3] = \mu t_i, \, i \in [3], \, \mu \in [3]^*.$$

The equations valid in $K(3)$ will be axiomatized by the following set of equations.

DEFINITION 3.2. Ax (3) consists of the following equations, for all p, q, x, y, z in $\underline{T(3)}$:

(3.2.1) (equal conclusion)

$$[p, x, x] \approx x;$$

(T-axioms)

(3.2.2) $[\mathrm{T}, x, y] \approx x$,

(3.2.3) $[p, p, x] \approx [p, \mathrm{T}, x]$.

(Common axioms)

(3.2.4) (premise interchange)

$$[p, [q, x, y], [q, u, v]] \approx [q, [p, x, u], [p, y, v]],$$

(3.2.5) (positive redundancy)

$$[p, [p, x, y], z] \approx [p, x, z],$$

(3.2.6) (negative redundancy)

$$[p, x, [p, y, z]] \approx [p, x, z],$$

(3.2.7) (premise simplification)

$$[[p, x, y], u, v] \approx [p, [x, u, v], [y, u, v]].$$

DEFINITION 3.3. A tree t is a simple premise tree if whenever $\nu t = \kappa$, then $t_{\nu 1} \in V$, i.e. any subterm of t of the form $[t_1, t_2, t_3]$ has the property that t_1 is a *variable*.

PROPOSITION 3.4. *For any t in $\underline{T(3)}$ there is a simple premise tree t' in $\underline{T(3)}$ such that* Ax $(3) \vdash t \approx t'$.

This fact follows easily using axioms (3.2.2) and (3.2.7).

DEFINITION 3.5. Let $A \in \underline{K(3)}$ and suppose that $h: \underline{T(3)} \to A$ is a homomorphism. Further, let t be a simple premise tree. Then Path (h, t) is the following word in $\{2, 3\}^*$:

if $t \in V \cup \{\mathrm{T}\}$, Path $(h, t) = \lambda$;

if $t = [p, t_2, t_3]$, then

$$\text{Path } (h, t) = \begin{cases} 2 \text{ Path } (h, t_2) & \text{if } ph = \mathrm{T}, \\ 3 \text{ Path } (h, t_3) & \text{if } ph \neq \mathrm{T}. \end{cases}$$

Remark 3.6. Let t be a simple-premise tree and $h: \underline{T(3)} \to A$, $A \in \underline{K(3)}$ a homomorphism. Then if $\nu = \text{Path } (h, t)$, $th = (\nu t)h$.

DEFINITION 3.7. Let t be a simple-premise tree and let ν be a nonpremise vertex of t (i.e. $\nu \in \{2, 3\}^*$ and νt is defined). Say ν is *accessible* if ν is an initial segment of Path (h, t), for some homomorphism $h: \underline{T(3)} \to A$ and some $A \in \underline{K(3)}$. The tree t itself is *accessible* if every nonpremise vertex of t is accessible.

We want to show every simple tree is provably equal to an accessible simple tree. In order to do so, we need to prove an appropriate insertion lemma. We need the following definitions:

Let $\bar{V}$ be a set in bijective correspondence with, but disjoint from the set of variables V. We write $\bar{v}$ for the element of $\bar{V}$ corresponding to $v \in V$. Let $L = V \cup \bar{V}$; the elements of L will be called "literals".

A *truth function* is a function $h: L \to \{0, 1\}$ such that for all $v \in V$,

$$\bar{v}h = 1 - vh$$

i.e., if $vh = 0$, $\bar{v}h = 1$ and if $vh = 1$, $\bar{v}h = 0$.

If $\alpha \in L$ and h is a truth function, say h *satisfies* α if $\alpha h = 1$.

Let $X \subseteq L$ and $\alpha \in L$. We write

$$X \overset{\circ}{\models} \alpha$$

if for every truth function h, $\alpha h = 1$ whenever h satisfies every member of X.

For each simple premise tree t and each nonpremise vertex ν in t, we define a set $S(\nu, t)$ of literals as follows:

DEFINITION 3.8. If $\nu = \lambda$, $S(\lambda, t) = \varnothing$;

if $t = [p, t_2, t_3]$, and $\nu = i\mu$, $\mu \in \{2, 3\}^*$,

$$S(\nu, t) = \begin{cases} \{p\} \cup S(\mu, t_2) & \text{if } i = 2, \\ \{\bar{p}\} \cup S(\mu, t_3) & \text{if } i = 3. \end{cases}$$

In the present context, a useful version of the insertion lemma is given in Lemma 3.9. The fact that only the "common axioms" are involved enables us to use this lemma in the next section as well.

INSERTION LEMMA 3.9 FOR $\underline{K(3)}$. *Suppose that ν is a nonpremise vertex of the simple premise tree t and let $p \in V$. If $S(\nu, t) \overset{\circ}{\models} p$ (respectively, $S(\nu, t) \overset{\circ}{\models} \bar{p}$) then $C \vdash t \approx t(\nu/[p, t_\nu, a])$ (respectively, $C \vdash t \approx t(\nu/[p, a, t_\nu])$) where C is the set of "common axioms" (3.2.4)–(3.2.7) and where a is a variable distinct from p and which does not occur in t.*

Proof. Before beginning the proof, which proceeds by induction on $|\nu|$ as in §§ 1 and 2, we note that $S(\nu, t) \overset{\circ}{\models} p$ (or $\bar{p}$) iff $S(\nu, t)$ is inconsistent or p (or $\bar{p}$) belongs to $S(\nu, t)$.

Basis step. $|\nu| = 0$. Since $S(\lambda, t)$ is empty, the hypothesis $S(\nu, t) \overset{\circ}{\models} p$ (or $\bar{p}$) is false, so there is nothing to prove.

Induction step. Suppose ν is μi, for $\mu \in \{2, 3\}^*$, $i = 2$ or 3 and let t_μ have the form $[q, t_2, t_3]$. We give the argument only for the case $S(\nu, t) \overset{\circ}{\models} p$, since the case $S(\nu, t) \overset{\circ}{\models} \bar{p}$ is entirely similar.

Case A. $S(\mu, t) \overset{\circ}{\models} p$. By the induction hypothesis. $C \vdash t \approx t(\mu/[p, [q, t_2, t_3], a])$. Now we show that for all p, q, t_2, t_3, a in $\underline{T(3)}$.

$$C \vdash [p, [q, t_2, t_3], a] \approx [p, [q, [p, t_2, a], t_3], a] \tag{3.9.1}$$

and

$$C \vdash [p, [q, t_2, t_3], a] \approx [p, [q, t_2, [p, t_3, a]], a]. \tag{3.9.2}$$

We will give the argument for (3.9.1) as a pseudo-equational deduction, writing the justification for each inference on the right.

$$\begin{aligned}
[p, [q, t_2, t_3], a] &\approx [p, [p, [q, t_2, t_3], [q, a, t_3]], a] && (3.2.5)\\
&\approx [p, [q, [p, t_2, a], [p, t_3, t_3]], a] && (3.2.4)\\
&\approx [p, [q, [p, [p, t_2, a], [p, t_2, a]], [p, t_3, t_3]], a] && (3.2.5)\\
&\approx [p, [p, [q, [p, t_2, a], t_3], [q, [p, t_2, a], t_3]], a] && (3.2.4)\\
&\approx [p, [q, [p, t_2, a], t_3], a] && (3.2.5)
\end{aligned}$$

which completes the proof of (3.9.1). The proof of (3.9.2) uses (3.9.1):

$$\begin{aligned}
[p, [q, t_2, [p, t_3, a]], a] &\approx [p, [q, [p, t_2, a], [p, t_3, a]], a] && (3.9.1)\\
&\approx [p, [p, [q, t_2, t_3], [q, a, a]], a] && (3.2.4)\\
&\approx [p, [q, t_2, t_3], a] && (3.2.5).
\end{aligned}$$

Now, using (3.9.1), we have that

$$C \vdash t \approx t(\mu/[p, [q, [p, t_2, a], t_3], a])$$

and, thus by the induction hypothesis, $C \vdash t \approx t(\mu/[q, [p, t_2, a], t_3])$; i.e. when ν is $\mu 2$, $C \vdash t \approx t(\nu/[p, t_2, a])$. The case $\nu = \mu 3$ uses (3.9.2).

Case B. $S(\mu, t) \overset{\circ}{\not\models} p$, but $S(\nu, t) \overset{\circ}{\models} p$.

Case 1. $S(\nu, t)$ is inconsistent. When ν is $\mu 2$, $S(\mu, t) \overset{\circ}{\models} \bar{q}$. In this case, by the induction hypothesis, $C \vdash t \approx t(\mu/[q, a, [q, t_2, t_3]])$. Thus, $C \vdash t \approx t(\mu/[q, a, [q, [p, t_2, a], t_3]])$, using the axiom of negative redundancy. So when ν is $\mu 2$, we have shown $C \vdash t \approx t(\nu/[p, t_2, a])$. The case ν is $\mu 3$ is similar, since $S(\mu, t) \overset{\circ}{\models} q$.

Case 2. $S(\nu, t)$ is consistent. Then if $S(\nu, t) \overset{\circ}{\models} p$, p belongs to $S(\nu, t)$. Since $S(\mu, t) \overset{\circ}{\not\models} p$, the variable p must be the variable q. So that $C \vdash [p, t_2, t_3] \approx [p, [p, t_2, a], t_3]$ and $C \vdash [p, t_2, t_3] \approx [p, t_2, [p, a, t_3]]$ by the redundancy axioms. This completes the proof.

We now obtain several useful corollaries of the insertion lemma.

COROLLARY 3.10. *For every simple premise tree t in $\underline{T(3)}$ there is an accessible simple premise tree t' such that $C \vdash t \approx t'$.*

The proof of Corollary 3.10 is similar to the corresponding result in the previous section.

A "p-subtree" of a tree t in $\underline{T(3)}$ is a subtree of t of the form $[p, u, v]$, for some u, v in $\underline{T(3)}$.

COROLLARY 3.11. *Let ν be a nonpremise node of the accessible simple premise tree t in $T(3)$ and suppose $S(\nu, t) \overset{\circ}{\models} \alpha$, where α is p or $\bar{p}$, for $p \in V$. Then t_ν has no p-subtrees. In particular, if t is $[p, t_2, t_3]$, then neither t_2 nor t_3 have any p-subtrees.*

The proof of Corollary 3.11 uses the insertion lemma to show that if t_ν contains a p-subtree, then t would contain an inaccessible nonpremise vertex.

We may slightly refine one aspect of Corollary 3.11.

COROLLARY 3.12. *With ν, t, p as in* 3.11, *if $S(\nu, t) \overset{\circ}{\models} p$, then $C + (\text{T-axioms}) \vdash t \approx t(\nu/t')$, where t' contains no leaf labeled p. In particular, if t is $[p, t_2, t_3]$, $C + (\text{T-axioms}) \vdash t \approx t[p, t_2', t_3]$, where t_2' has no occurrence of p.*

Proof. Let $\nu\nu'$ be a leaf of t_ν labeled p. By Corollary 3.11, we know $\nu\nu'$ is a nonpremise leaf. Then $S(\nu\nu', t) \overset{\circ}{\models} p$, so that $C \vdash t \approx t(\nu\nu'/[p, t_{\nu\nu'}, a])$. But $t_{\nu\nu'}$ is p and, so, by Axiom (3.2.3), $C + (\text{T-axioms}) \vdash t \approx t(\nu\nu'/[p, \text{T}, a])$. Thus, by the insertion lemma, $C + (\text{T-axioms}) \vdash t \approx t(\nu\nu'/\text{T})$. This shows how to replace any occurrence of p in t_ν by T. By induction on the number of such occurrences, the proof is complete.

For t, t' in $\underline{T(3)}$, we write $\models t \approx t'$ if for all $A \in \underline{K(3)}$ and all homomorphisms $h: \underline{T(3)} \to A$, $th = t'h$. It should be clear that $\models t \approx t'$ whenever $\text{Ax}(3) \vdash t \approx t'$. Before proving the converse, we need some further facts.

PROPOSITION 3.13. *Suppose t is an accessible simple premise tree of the form $[p, t_2, t_3]$. If $\models t \approx \text{T}$, then $\models t_i \approx \text{T}$, for $i = 2, 3$.*

Proof. First we show $\models t_2 \approx \text{T}$. Let $h: \underline{T(3)} \to A$, $A \in \underline{K(3)}$ be a homomorphism. Then, since p does not occur in t_2, $t_2 h = t_2 h^p_{\text{T}} = [p, t_2, t_3] h^p_{\text{T}} = \text{T}$, so that $\models t_2 \approx \text{T}$. ($h^p_{\text{T}}$ is the homomorphism agreeing with h on all variables $\neq p$; $p h^p_{\text{T}} = \text{T}$.)

Next, in order to obtain a contradiction, suppose h is some homomorphism such that $t_3 h \neq \text{T}$. Let $\nu = \text{Path}(h, t_3)$ and $z = \nu t_3$. Since neither p nor $\bar{p}$ occurs in $S(\nu, t_3)$, there is a homomorphism h' such that $ph' = zh' \neq \text{T}$, and for all $v \in V$ such that either v or $\bar{v}$ is in $S(\nu, t_3)$, $vh' = \text{T}$ iff $vh = \text{T}$.

But then

$$\begin{aligned} [p, t_2, t_3]h' &= t_3 h' \quad (\text{since } ph' \neq \text{T}) \\ &= zh' \quad (\text{since } h' \text{ satisfies } S(\nu, t_3)) \\ &\neq \text{T} \end{aligned}$$

contradicting the assumption that $\models [p, t_2, t_3] \approx \text{T}$.

PROPOSITION 3.14. *Suppose that $\models [p, t_2, t_3] \approx [p, t_2', t_3']$, where t_i, t_i', are simple premise trees in $\underline{T(3)}$. Then $\models t_i \approx t_i'$, $i = 2, 3$.*

Proof. By Corollaries 3.11 and 3.12 we may assume that neither t_2 nor t_2' contain any occurrence of p, and neither t_3 nor t_3' contain any p-subtrees. Then, for any homomorphism $h: \underline{T(3)} \to A$, $A \in \underline{K(3)}$,

$$t_2 h = t_2 h^p_{\text{T}} = [p, t_2, t_3] h^p_{\text{T}} = [p, t_2', t_3'] h^p_{\text{T}} = t_2' h^p_{\text{T}} = t_2' h,$$

showing that $\models t_2 \approx t_2'$.

We prove $\models t_3 \approx t_3'$ by contradiction. If $t_3 h \neq t_3' h$ for some homomorphism h, let $\nu = \text{Path}(h, t_3)$, $\nu' = \text{Path}(h, t_3')$, and let $z = \nu t_3$, $z' = \nu' t_3'$. One of $\{z, z'\}$ is not T, say z. Since neither p nor $\bar{p}$ occurs in $S(\nu, t_3) \cup S(\nu', t_3')$, there is a homomorphism h' such that

(a) $\text{Path}(h', t_3) = \text{Path}(h, t_3)$ and $\text{Path}(h', t_3') = \text{Path}(h, t_3')$,

(b) $ph' \neq \text{T} \neq zh' \neq z'h'$.

But then it easily follows that

$$[p, t_2, t_3]h' \neq [p, t_2', t_3']h'$$

contradicting the hypothesis that

$$\models [p, t_2, t_3] \approx [p, t_2', t_3'].$$

Thus Proposition 3.14 is proved.

We may now prove the

COMPLETENESS THEOREM: *If* $\models t \approx t'$, *then* $\mathrm{Ax}(3) \vdash t \approx t'$.

Proof. By induction on $\kappa(t)$, the number of occurrences of κ in t. If $\kappa(t) = 0$, then t is T or a variable. When t is T, if $\kappa(t') = 0$, $t' = \mathrm{T}$, so $\mathrm{Ax}(3) \vdash t \approx t'$ trivially. If $\kappa(t') > 0$, write t' as $t' = [p, t_2', t_3']$. Then, by Proposition 3.13, $\models \mathrm{T} \approx t_i'$, $i = 2, 3$. By the induction hypothesis, $\mathrm{Ax}(3) \vdash \mathrm{T} \approx t_i'$, $i = 2, 3$, so that $\mathrm{Ax}(3) \vdash t' \approx [p, \mathrm{T}, \mathrm{T}]$. Using the equal-conclusion axiom, $\mathrm{Ax}(3) \vdash t' \approx \mathrm{T}$, i.e. $\mathrm{Ax}(3) \vdash t \approx t'$.

Now assume t is a variable, say $t = p$. If $\kappa(t') = 0$ then clearly $t' = p$ also. Hence assume $\kappa(t') > 0$. Since $\mathrm{Ax}(3) \vdash t' = [p, t', t']$, we may assume that t' has the form $[p, t_2', t_3']$. But since $\models p \approx [p, \mathrm{T}, p]$, we have $\models [p, \mathrm{T}, p] \approx [p, t_2', t_3']$. Thus, by 3.14, $\models \mathrm{T} \approx t_2'$ and $\models p \approx t_3'$. By the above and the induction hypothesis, $\mathrm{Ax}(3) \vdash \mathrm{T} \approx t_2'$ and $\mathrm{Ax}(3) \vdash p \approx t_3'$. Hence $\mathrm{Ax}(3) \vdash [p, t_2', t_3'] \approx [p, \mathrm{T}, p]$. The basis step is completed by (3.2.3).

Now assume t is $[p, t_2, t_3]$. Again, we may assume t' is $[p, t_2', t_3']$. Thus, by Proposition 3.14, $\models t_i \approx t_i'$, $i = 2, 3$. Hence, by the induction hypothesis, $\mathrm{Ax}(3) \vdash t_i \approx t_i'$ and hence $\mathrm{Ax}(3) \vdash t \approx t'$, completing the proof.

Remark 3.15. Let A_3 be a three-element algebra in $\underline{K(3)}$. (Any two such algebras are isomorphic.) Then for all t, t' in $\underline{T(3)}$, $\models t \approx t'$ iff for all $h: \underline{T(3)} \to A_3$, $th = t'h$. Note that in any two-element algebra A_2 in $\underline{K(3)}$, the following equation is true:

$$[p, a, [q, a, q]] \approx [p, a, [q, a, p]].$$

But this equation is not valid in $\underline{K(3)}$.

4. Axiomatization of the class $K(3; \perp)$. This class is a modification of the class $\underline{K(3)}$. An algebra $(A, \kappa, \mathrm{T}, \mathrm{F}, \perp)$ belongs to $K(3; \perp)$ iff T, F, and $\perp$ are distinguished constants in the set A, and $\kappa: A^3 \to A$ is a function which satisfies

$$\kappa(p, x, y) = \begin{cases} x & \text{if } p = \mathrm{T}, \\ y & \text{if } p = \mathrm{F}, \\ \perp & \text{otherwise.} \end{cases} \tag{4.1}$$

Here $\perp$ models "undefined", T and F model "true" and "false" respectively.

Note that if $A \in K(3; \perp)$ and either $\mathrm{T} = \mathrm{F}$ or $\mathrm{T} = \perp$ or $\mathrm{F} = \perp$, then A is a singleton set.

We let $T(3; \mathrm{T})$ denote the algebra of terms built from the countably infinite set V, the constant symbols T, F and $\perp$, and the function symbol κ. As before, we write $[p, x, y]$ instead of $\kappa(p, x, y)$, and identify elements of $T(3; \perp)$ with certain partial functions $t: [3]^* \to V \cup \{\mathrm{T}, \mathrm{F}, \perp, \kappa\}$.

The equations valid in $K(3; \perp)$ will be axiomatized by the set of axioms $\mathrm{Ax}(3; \perp)$ below. Note that $\mathrm{Ax}(3; \perp)$ is obtained by adding to the T-axioms and common axioms in $\mathrm{Ax}(3)$ only axioms which involve the new constant symbols F and $\perp$. Further we must necessarily delete the equal conclusion axiom $[p, x, x] = x$, which is clearly not valid in $K(3; \perp)$. Thus the argument used to prove the completeness theorem in the previous section cannot be used.

DEFINITION 4.2. Ax $(3; \perp)$ consists of the following equations, for all p, x, y in $T(3; \perp)$

(4.2.1) (the T-axioms): (3.2.2) and (3.2.3).

(the F-axioms)

(4.2.2) $[\mathrm{F}, x, y] \approx y$,

(4.2.3) $[p, x, p] \approx [p, x, \mathrm{F}]$.

(the $\perp$-axioms)

(4.2.4) $[\perp, x, y] \approx \perp$,

(4.2.5) $[p, \perp, \perp] \approx \perp$,

(4.2.6) (the common axioms), (3.2.4)–(3.2.7).

Of course, when using an axiom in Ax (3) as an axiom in Ax $(3; \perp)$, e.g. $[p, p, x] \approx [p, \mathrm{T}, x]$, we assume that the letters p, x, etc. range over $T(3; \perp)$, not just $\underline{T(3)}$.

A tree t in $T(3; \perp)$ is a *simple premise* tree if whenever $\nu t = \kappa$, for ν in $[3]^*$, then $\nu 1 t \in V$. The proof of Fact 4.3 uses the premise reduction and the T-, F- and $\perp$-axioms.

FACT 4.3. *For any t in $T(3; \perp)$ there is a simple premise tree t' such that* Ax $(3; \perp) \vdash t \approx t'$.

For nonpremise nodes ν of a simple premise tree t in $T(3; \perp)$, the set $S(\nu, t)$ of literals is defined here exactly as in Definition 3.8. We may make use of the insertion lemma 3.9, since C, the set of common axioms, is a subset of Ax $(3; \perp)$.

We must define Path (h, t), for a homomorphism $h: T(3; \perp) \to A$, $A \in K(3; \perp)$ and t in $T(3; \perp)$, slightly differently from Definition 3.5, since ph need not be T or F in A, for some variables p.

DEFINITION 4.4. Let $h: T(3; \perp) \to A$, $A \in K(3; \perp)$, be a homomorphism and suppose t is a simple premise tree in $T(3; \perp)$. Path (h, t) is the following word in $\{2, 3, \perp\}^*$:

(4.4.1) if t is in $V \cup \{\mathrm{T}, \mathrm{F}, \perp\}$,

$$\text{Path } (h, t) = \lambda, \text{ the empty word.}$$

(4.4.2) if t is $[p, t_2, t_3]$ then

$$\text{Path } (h, t) = \begin{cases} 2 \text{ Path } (h, t_2) & \text{if } ph = \mathrm{T}, \\ 3 \text{ Path } (h, t_3) & \text{if } ph = \mathrm{F}, \\ \perp & \text{otherwise.} \end{cases}$$

The definition of an *accessible* (nonpremise vertex of a) simple premise tree t is formally the same as Definition 3.7, even though Path (h, t) may end with the symbol $\perp$.

PROPOSITION 4.5. *For every simple premise tree t in $T(3; \perp)$ there is an accessible simple premise tree t' in $T(3; \perp)$ such that* Ax $(3; \perp) \vdash t \approx t'$.

The proof is the same as that of Corollary 3.10. In fact, we may use "C" in place of "Ax $(3; \perp)$" in the statement of Proposition 4.5, where C denotes the set of common axioms. However, we will not need this refinement.

We now state a proposition which combines Corollaries 3.11, 3.12 and one additional fact. First, it is convenient to introduce some terminology.

DEFINITION 4.6. Let t be a simple premise tree in $T(3; \perp)$. We say t is *reduced* if t is accessible and whenever t has a subtree of the form $[p, t_2, t_3]$ then the variable p is not the label of any leaf of t_2 or t_3.

PROPOSITION 4.7. *For any simple premise tree t there is a reduced tree t' such that* Ax $(3; \perp) \vdash t \approx t'$.

Proof. By Corollaries 3.11 and 3.12, it is necessary to show only that p may be eliminated from nonpremise leaves of t_3, in any accessible tree of the form $[p, t_2, t_3]$. But this fact is proved using the F-axioms and the insertion lemma 3.9.

The following fact plays an important part in the proof of the completeness theorem.

PROPOSITION 4.8. *Let* $[p, t_2, t_3]$ *and* $[q, t_2', t_3']$ *be reduced trees such that* $\models [p, t_2, t_3] \approx [q, t_2', t_3']$. *Then if* p *is distinct from* q, *either* p *or* $\bar{p}$ *belongs to* $S(\nu, t_i')$, *for each nonpremise leaf* ν *of* t_i', $i = 2, 3$.

Proof. Suppose ν is a nonpremise leaf of t_2', say, such that neither p nor $\bar{p}$ occurs in $S(\nu, t_2')$. Let $z = \nu t_2'$, the label of ν in t_2'. Assume z is a variable. Since $[q, t_2', t_3']$ is reduced, neither z, q, nor $\bar{q}$ occurs in $S(\nu, t_2')$ either. Thus there is a homomorphism $h: T(3; \perp) \to A_4$, where A_4 is a four element algebra in $K(3; \perp)$, $A_4 = \{\mathrm{T}, \mathrm{F}, \perp, a_0\}$ such that $ph = zh = a_0$, $qh = \mathrm{T}$ and Path $(h, t_2) = \nu$. But then $\perp = [p, t_2, t_3]h \neq [q, t_2', t_3']h = t_2'h = zh = a_0$, contradicting the hypothesis.

The cases when z is a constant symbol, or when ν is a nonpremise leaf in t_3' are handled similarly. The proof is complete.

We will now prove the

COMPLETENESS THEOREM. *Let* t, t' *be trees in* $T(3; \perp)$, *such that* $\models t \approx t'$. *Then* $\mathrm{Ax}(3; \perp) \vdash t \approx t'$.

Proof. By induction on $|t|$, the number of vertices in t. First, by Proposition 4.7, we may assume that t and t' are reduced.

Now if $|t| = 1$ and t is a variable, or one of the constants T or F, then the only reduced tree t' satisfying $\models t \approx t'$ is t itself. (This fact is easily proved by induction on $|t'|$.) Thus in these cases, there is nothing to prove. If t is the constant $\perp$ and $|t'| = 1$, then t' is also $\perp$. Now if t' is $[q, t_2', t_3']$ and $\models \perp \approx [q, t_2', t_3']$ then $\models \perp \approx t_i'$, $i = 2, 3$. So, by induction in $|t'|$, $\mathrm{Ax}(3; \perp) \vdash \perp \approx t_i'$ and hence $\mathrm{Ax}(3; \perp) \vdash t' \approx [q, \perp, \perp]$. The proof of this case is completed by the $\perp$-axiom (4.2.5).

Now suppose t is $[p, t_2, t_3]$ and t' is $[q, t_2', t_3']$. (We may clearly assume $|t'| > 1$.)

First assume p is q. Then $\models t_i \approx t_i'$, $i = 2, 3$. Indeed, when $i = 2$ (the case $i = 3$ is entirely similar), for any homomorphism $h: T(3; \perp) \to A$, $A \in K(3; \perp)$,

$$\begin{aligned} t_2h &= t_2h_{\mathrm{T}}^p \quad \text{(since } [p, t_2, t_3] \text{ is reduced)} \\ &= [p, t_2, t_3]h_{\mathrm{T}}^p \\ &= [p, t_2', t_3']h_{\mathrm{T}}^p \quad \text{(since } \models [p, t_2, t_3] = [p, t_2', t_3']) \\ &= t_2'h_{\mathrm{T}}^p = t_2'h \quad \text{(since } [p, t_2', t_3'] \text{ is reduced).} \end{aligned}$$

Now assume p and q are distinct. By Proposition 4.8, either p or $\bar{p}$ occurs in $S(\nu, t_i')$ for every nonpremise leaf ν in t_i', $i = 2, 3$. Hence, by the insertion lemma 3.9, if z is the label of such a leaf, we may replace z by a tree of the form $[p, z, z_2]$. Then we apply the premise interchange axiom sufficiently many times to the resulting tree, so that $\mathrm{Ax}(3; \perp) \vdash [q, t_2', t_3'] \approx [p, t_2'', t_3'']$, for some t_2'', t_3''. We may now apply the first argument to deduce $\mathrm{Ax}(3; \perp) \vdash t \approx t'$ in this case, completing the induction step and the proof of the theorem.

Remark 4.9. Let A_4 be a four-element algebra in $K(3; \perp)$. Then, for any t, t' in $T(3; \perp)$, $\models t \approx t'$ iff for all homomorphisms $h: T(3; \perp) \to A_4$, $th = t'h$.

Appendix. Some facts about the logic of signed equations. Let Σ be a ranked alphabet. A *positive* Σ*-equation* (respectively, a *negative* Σ-equation) is an expression of the form $t \approx t'$ (respectively, $t \not\approx t'$), for t, t' in T_Σ, the set of Σ-terms. A *signed* Σ-equation is either a positive or negative Σ-equation.

If A is a Σ-algebra and $h: T_\Sigma \to A$ is a homomorphism, we say h *satisfies* the equation $t \approx t'$ if $th = t'h$, and h satisfies $t \not\approx t'$ otherwise. A signed equation α is true in A if every homomorphism $T_\Sigma \to A$ satisfies α.

Let M_Σ denote the set of all signed Σ-equations. For $S \subseteq M_\Sigma$, $\alpha \in M_\Sigma$, we write

$$S \models \alpha$$

if for every Σ-algebra A, α is true in A whenever each signed equation in S is true in A. We will write

$$S \models^{\circ} \alpha$$

if for every homomorphism $h: T_\Sigma \to A$, h satisfies α whenever h satisfies every element of S.

A set $S \subseteq M_\Sigma$ is $\models$-inconsistent (respectively $\models^{\circ}$-inconsistent) if $S \models x \neq x$, (respectively $S \models^{\circ} x \neq x$) for some variable x. Clearly, if $S \models^{\circ} x \neq x$, then $S \models x \neq x$, and more generally,

(A.1) if $S \models^{\circ} \alpha$ then $S \models \alpha$.

For $S \subseteq M_\Sigma$, S^+ denotes the set of positive equations in S, and $S^- = S - S^+$.

THEOREM A2. *Suppose $S \subseteq M_\Sigma$, $t, t' \in T_\Sigma$.*

i) *If S is $\models$-consistent and $S \models t \approx t'$, then $S^+ \models t \approx t'$.*

ii) *The same as (i), with $\models^{\circ}$ in place of $\models$.*

Proof sketch. (i) Define the relation $\sim$ on T_Σ as follows: for s, s' in T_Σ, $s \sim s'$ if $S^+ \models s \approx s'$. Then $\sim$ is a congruence relation on T_Σ and since S is $\models$-consistent, each signed equation in S is true in the quotient algebra $T_\Sigma/\sim$. Thus $t \approx t'$ must be true in $T_\Sigma/\sim$, so in particular $t \sim t'$; i.e. $S^+ \models t \approx t'$.

The proof of (ii) is similar.

THEOREM A.3. *Suppose $S \subseteq M_\Sigma$, $t, t' \in T_\Sigma$.*

i) *If $S \models t \neq t'$, there is one negative equation $s \neq s'$ in S such that $S^+ \cup \{s \neq s'\} \models t \neq t'$.*

ii) *Same as* i), *with $\models^{\circ}$ in place of $\models$.*

Proof of (i). Suppose that for each negative equation n in S^-, $S^+ \cup \{n\} \not\models t \neq t'$. Then there is some Σ-algebra A_n such that $S^+ \cup \{n\}$ is true in A_n but $t \neq t'$ is *not* true in A_n. Let $A = \Pi(A_n : n \in S^-)$. It is easily seen that S is true in A, but $t \neq t'$ is not true in A, contradicting the assumption that $S \models t \neq t'$.

The proof of (ii) is similar.

The remaining theorems treat the case that Σ is empty, so that T_Σ is just V, the set of variables and a Σ-algebra is just a nonempty set. We write M instead of M_Σ in this case. Since the relation $\models$ is rather uninteresting when $\Sigma = \varnothing$, we consider only $\models^{\circ}$.

Given a subset S of M, let $G(S)$ denote the following labeled undirected graph.

The vertices of $G(S)$ are the elements of V.

For each positive equation $x \approx y$ in S, there is an edge between x and y labeled $\oplus$; for each negative equation $x \neq y$ in S there is an edge between x and y labeled $\ominus$. There are no other edges in $G(S)$.

Call $G(S)$ inconsistent iff there is a cycle in $G(S)$ containing exactly one edge labeled $\ominus$.

THEOREM A.4. i) *S is $\models^{\circ}$-inconsistent iff $G(S)$ is inconsistent.*

ii) *If S is $\models^{\circ}$-consistent, then $S \models^{\circ} x \approx y$ iff $x = y$ or there is a path in $G(S)$ between x and y all of whose edges are labeled $\oplus$.*

iii) *If S is $\models^{\circ}$-consistent, then $S \models^{\circ} x \neq y$ iff there is a path in $G(S)$ between x and y with exactly one edge labeled $\ominus$.*

Proof sketch. (i) Clearly if $G(S)$ is inconsistent, so is S. Now suppose $G(S)$ is consistent. We show there is a set A and a function $h: V \to A$ such that h satisfies each signed equation in S, showing S is $\models^{\circ}$-consistent.

Define the relation $\sim$ on V by: $v \sim v'$ iff $v = v'$ or there is a path in $G(S)$ between v and v' all of whose edges are labeled $\oplus$. Then clearly $\sim$ is an equivalence relation on V, and if $x \approx y \in S$, $x \sim y$. Also, if $x \not\approx y \in S$, it cannot happen that $x \sim y$, or else $G(S)$ will be inconsistent. Hence, we may let h be the canonical map $v \mapsto v/\sim$ from V to $A = V/\sim$, and h will satisfy S.

The proof of (ii) is similar.

The fact that $S \overset{\circ}{\models} x \not\approx y$ if there is a path with exactly one edge labeled $\ominus$ between x and y is clear. So suppose, in order to prove the converse that S is $\overset{\circ}{\models}$-consistent, and $S \overset{\circ}{\models} x \not\approx y$. Let S_1 be $S \cup \{x \approx y\}$. $G(S_1)$ differs from $G(S)$ in that there is one new edge, labeled $\oplus$, between x and y. Since S_1 is $\overset{\circ}{\models}$-inconsistent, so is $G(S_1)$, by (i). But then there must be a path between x and y containing exactly one edge labeled $\ominus$, completing the proof.

COROLLARY A.5. *Suppose* $S \subseteq M, x, y, u, v \in V$. *If* $S \overset{\circ}{\not\models} x \not\approx y$, *but* $S \cup \{u \approx v\} \overset{\circ}{\models} x \not\approx y$, *then if* $S \cup \{u \approx v\}$ *is consistent,* $S \overset{\circ}{\models} x \not\approx u$ *and* $S \overset{\circ}{\models} y \approx v$ (*or* $S \overset{\circ}{\models} x \approx u$ *and* $S \overset{\circ}{\models} y \not\approx v$).

This fact follows from Theorem A.4iii).

Acknowledgments. The authors are grateful to Irene Guessarian who carefully read a draft of this paper and corrected a large number of misprints and minor errors. The referees also provided helpful comments and suggestions, and found some further mistakes. Each of us will assure you that any remaining errors are the fault of the other.

REFERENCES

[1] A. ARNOLD, *Semantique algebrique de l'appel par valeur*, Theoret. Comput. Sci., 12 (1978), pp. 69–82.

[2] J. W. BACKUS, *Can programming be liberated from the von Neumann style?*, Comm. ACM, 21 (1978), pp. 613–639.

[3] S. L. BLOOM AND R. TINDELL, *Compatible orderings on the metric theory of trees*, this Journal, 11 (1980), pp. 683–691.

[4] B. COURCELLE, *Infinite trees in normal form and recursion equations having a unique solution*, Math. Systems Theory, 13 (1979), pp. 131–180.

[5] B. COURCELLE AND M. NIVAT, *Algebraic families of interpretations*, 17th Symposium on Foundations of Computer Science, Houston, Texas, 1976.

[6] G. COUSINEAU AND M. NIVAT, *On rational expressions representing infinite rational trees: applications to the study of flowcharts*, manuscript.

[7] S. GORN, *Explicit definitions and linguistic dominoes*, Systems and Computer Science, J. Hart and S. Takasu, eds., Univ. Toronto Press, Toronto, 1965.

[8] Z. MANNA AND J. VUILLEMIN, *Fixpoint approach to the theory of computation*, Comm. ACM, 15 (1972), pp. 528–536.

[9] J. MCCARTHY, *A basis for a mathematical theory of computation*, in Computer Programming and Formal Systems, Braffort and Hirschberg, eds., North-Holland, Amsterdam, 1963.

[10] H. SAMET, *A canonical form algorithm for proving equivalence of conditional forms*, Inform. Proc. Letters, 7 (1978), pp. 103–106.

[11] R. SETHI, *Conditional expressions with equality tests*, J. Assoc. Comput. Mach., 25 (1978), pp. 667–674.

[12] E. WAGNER, J. THATCHER AND J. WRIGHT, *Programming languages as mathematical objects*, Proc. Math. Found. Comp. Sci. 1978, Lecture Notes in Computer Science 64, Springer-Verlag, New York, 1978, pp. 84–101.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0008 $01.25/0

ON THE SPACE AND TIME COMPLEXITY OF FUNCTIONS COMPUTABLE BY SIMPLE PROGRAMS*

TAT-HUNG CHAN† AND OSCAR H. IBARRA‡

Abstract. We study the space and time complexity of functions computable by simple loop-free programs operating on integers. In particular, we show that any function $f(x_1, \cdots, x_k)$ computable by a program using only comparison-based conditional forward branching instructions and the arithmetic operations $+$, $-$, and truncating division by integer constants (such programs compute exactly the functions definable in Presburger arithmetic) can be computed by an off-line Turing machine in space $s(n)$ and time $n^2/s(n)$ for any reasonable space bound $s(n)$ between $\log n$ and n. Moreover, the space-time trade-off is optimal.

Key words. space complexity, time complexity, loop-free programs

1. Introduction. Space-efficient algorithms for some specific problems are known. For example, we know that context-free languages are recognizable on an off-line Turing machine in space $O(\log^2 n)$ [14]. For deterministic context-free languages, a space-time trade-off with $s(n)$ for space and $n^2/s(n)$ for time for any reasonable $\log^2 n \leqq s(n) \leqq n$ is known [15] (see also [3]). Algorithms of space complexity $O(\log n)$ and $O(\log^2 n)$ for some matrix and graph problems can be found in [12]. Our results, which concern functions computable by simple loop-free programs, are similar in nature. The classes of programs we consider are the following:

K —Set of all programs over integer inputs (Z) using only constructs of the form $x \leftarrow 0$, $x \leftarrow x+1$, $x \leftarrow y+z$, $x \leftarrow y-z$, $x \leftarrow y/k$, **skip** l, **if** $x = 0$ **then skip** l, **if** $x > 0$ **then skip** l, **halt**. (Here, k is a positive integer and x/k denotes integer division with truncation[1]; l is a nonnegative integer and **skip** l causes the $(l+1)st$ instruction following the current instruction to be executed next.)

K^+ —Similar to the class K except that $x \leftarrow y-z$ is replaced by $x \leftarrow y \dot{-} z$ (proper subtraction[2]), and the inputs range over N, the set of nonnegative integers.

L —Add to the constructs in K nonscalar multiplication and division, i.e., $x \leftarrow y*z$ and $x \leftarrow y/z$.

M —Add to the constructs in L the instruction $x \leftarrow \lfloor \sqrt[k]{y} \rfloor$, where k denotes a positive integer constant.

Each program has a distinguished set of input variables and one output variable. By convention, the value of the output variable is undefined on inputs which cause the program to divide by 0 or compute the kth-root of a negative number.

Notation. For a class C of programs, $\mathcal{F}(C)$ denotes the class of functions computable by programs in C. For a positive integer i, $C(i)$ denotes the programs in C with exactly i *input* variables (but with an unrestricted number of intermediate variables).

Let P be the set of all Presburger functions over the integers (i.e., functions over the integers definable in Presburger arithmetic) and P^+ be the set of all Presburger functions over the nonnegative integers. Then the following theorem follows from results in [10].

* Received by the editors October 27, 1981, and in revised form October 11, 1982.

† Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455.

‡ Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455. The research of this author was supported in part by National Science Foundation grant MCS8102853.

[1] e.g., $5/2 = 2$, $-5/2 = -2$.

[2] $y \dot{-} z = y - z$ if $y \geqq z$, 0 otherwise.

THEOREM 1.1. *$\mathcal{F}(K) = P$ and $\mathcal{F}(K^+) = P^+$.*

In §§ 2–4, we shall investigate the space and time complexity of functions in $\mathcal{F}(K)$, $\mathcal{F}(L(1))$, $\mathcal{F}(L)$, and $\mathcal{F}(M)$. The model of computation is a (deterministic) two-way transducer. Its finite-state control is attached to a two-way read-only input tape (with endmarkers), a one-way write-only output tape, and a two-way read-write work tape (Fig. 1). See [9] for a formal definition.

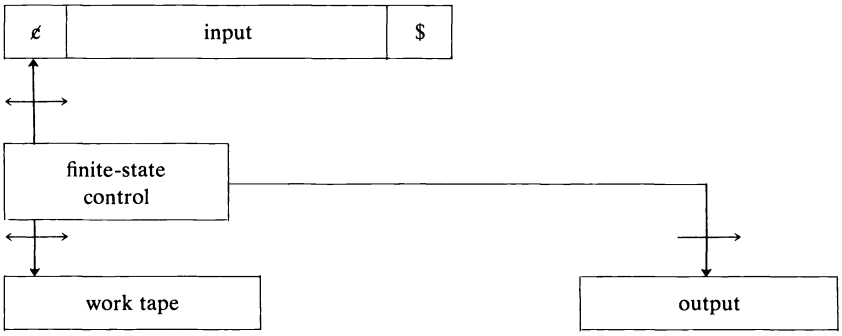


FIG. 1. *A two-way transducer.*

A function $f(x_1, \cdots, x_k)$ over the integers is computable in space $s(n)$ if there is a two-way transducer which when given any input $x_1 \# x_2 \cdots \# x_k$, each x_i in binary (with sign attached), outputs $f(x_1, \cdots, x_k)$ in binary using at most $s(n)$ cells on its work tape. The time complexity $t(n)$ is defined similarly. By convention, on inputs for which f is undefined, the transducer goes into an infinite loop.

2. The class $\mathcal{F}(K)$.

THEOREM 2.1. *Every function in $\mathcal{F}(K)$ can be computed by a two-way transducer simultaneously in space $s(n)$ and time $t(n) = n^2/s(n)$ for any $s(n)$ such that*

(1) $\log n \leqq s(n) \leqq n$,

(2) *$s(n)$ is constructible simultaneously in $s(n)$ space and $n^2/s(n)$ time. (See [9] for a definition of constructible.)*

Moreover, the time complexity $n^2/s(n)$ is optimal in $s(n)$ space, and the $\log n$ lower bound for the space complexity is also optimal in that there are functions in $\mathcal{F}(K)$ not computable in asymptotically smaller space.

Proof. We first consider the class $\mathcal{F}(K^+)$. By Theorem 1.1, $\mathcal{F}(K^+) = P^+$ equals the class of Presburger functions (over N). Let $f: N^k \to N$ be a function in P^+. Hence the graph of f, i.e., the set $\{(x_1, \cdots, x_k, x_{k+1}) | f(x_1, \cdots, x_k) = x_{k+1}\}$, is a semilinear set [5], say

$$\bigcup_{i=1}^{m} \left\{ v_0^i + \sum_{j=1}^{r_i} t_j v_j^i \,|\, t_j \geqq 0 \right\}$$

where $v_j^i \in N^{k+1}$. Following the strategy of [6], we can compute f as follows. Given $x_1, \cdots, x_k \in N$, find an $i \in \{1, \cdots, m\}$ and $t_1, \cdots, t_{r_i} \in N$ such that $x_1, \cdots, x_k$ are the first k components of $w = v_0^i + \sum_{j=1}^{r_i} t_j v_j^i$. Then $f(x_1, \cdots, x_k)$ is the last component of w. 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 [7].

Let $A\bar{y} = \bar{b}$ be a system of linear equations, where A is an $m \times n$ integral matrix, $\bar{y} = (y, \cdots, y_n)^T$ 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 $L \subseteq \{l_1, \cdots, l_r\} \subseteq \{1, \cdots, n\}$, $\hat{y}_i < \Psi$ for each $i \notin L$. 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 graph is associated a finite set of nonsingular systems each of which arises by "predetermining" some of the t_j's. For each $\bar{x} = (x_1, \cdots, x_k)^T \in N^k$, one need only try each such system to see if the remaining t_j's are solvable in nonnegative integers. By definition, we know that this will happen with at least one system, and the solution $t_1, \cdots, t_{r_i}$ is used to compute the last component of the corresponding vector $v_0^i + \sum_{j=1}^{r_i} t_j v_j^i$ to yield the desired function value $f(x_1, \cdots, x_k)$.

The solution of each of the nonsingular systems can be effected by applying Cramér's rule to a square nonsingular subsystem. Thus each "nonpredetermined" t_j can be written as

$$t_j = \frac{\sum a_p y_p - \sum b_q z_q + c}{\Delta} \tag{1}$$

where y_p, z_q are components of $\bar{x}$, a_p, b_q and Δ are positive integers, and c is an integer. Here a_p, b_q and Δ depend only on the vectors defining the underlying linear 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.

As a first step, for each t_j to be solved, we need to check that the value given by (1) is a nonnegative integer. The integrality test takes linear time and no auxiliary space—a finite automaton scanning the input once and computing mod Δ suffices. To test nonnegativity, assume that the machine has laid out $s(n)$ cells for use on the work tape at the start of the computation. Then the machine can generate successive segments of $s(n)$ bits of $\sum a_p y_p$ and $\sum b_q z_q$ on its work tape (with c or $-c$ added to the appropriate sum) and compare them, proceeding from low-order bits to high-order bits. Thus for the ith segment, the machine copies bits $(i-1)s(n)$ to $i \cdot s(n) - 1$ of the arguments y_p, z_q onto its work tape, and then in one pass (using carries from the $(i-1)$st segments and any relevant bits from c) compute bits $(i-1)s(n)$ to $i \cdot s(n) - 1$ of the two sums. Using the result of the comparisons of the first $(i-1)$ segments, the machine can decide how the first i segments compare, and it has also generated the carries into the $(i+1)$st segments of the sums.

We now analyze the space and time requirements of the nonnegativity tests. Because the coefficients are all independent of the input, the carries are bounded by a constant depending only on f and hence can be stored in finite memory. Also, since $s(n) \geqq \log n$, there is enough space to maintain binary pointers to input positions. This pointer is 0 to begin with. Inductively, after the ith segments have been processed, it has value $i \cdot s(n)$. Starting from bit 0 of an input argument, the input head moves to the next more significant bit for each decrement of 1 from the binary count. By the analysis in [4], it takes $2i \cdot s(n)$ Turing machine steps to decrement the binary count from $i \cdot s(n)$ to 0. Then the machine copies the next $s(n)$ bits onto the work tape, and the input head is on bit $(i+1)s(n)$ of the argument. At this point, the binary count $(i+1)s(n)$ can be generated on the work tape by moving the input head back to bit 0 while incrementing an initially zero binary count; this takes $2(i+1)s(n)$ time. The same process is used to copy the $(i+1)$st segments of all relevant arguments onto the work tape, and since the input has length n and $i+1 \leqq \lceil n/s(n) \rceil$, it takes $O(n)$ time to position the input head, copy the bits and generate the binary count $(i+1)s(n)$

on the work tape. The subsequent additions and comparison take $O(s(n)) \leqq O(n)$ time. Hence one segment requires $O(n)$ time. Since there are altogether $O(n/s(n))$ segments, it takes $O(n^2/s(n))$ time to test each system of equations (1).

Having found a system whose chosen square subsystem yields nonnegative integral t_j's, we then have to check that the t_j's (including the predetermined ones) in fact also satisfy the $m-r$ equations that were deleted in forming the square subsystem. If these equations are satisfied then we know we have the right system from which to generate the function value. Both steps involve computing nonnegative affine combinations of the t_j's, the results then being compared with appropriate input arguments or written on the output tape.

We shall again carry out the computations in segments of $s(n)$ bits. Since we already saw how to copy successive segments of the input arguments onto the work tape using $O(n)$ time for each segment, we shall concentrate on the computation of affine combinations of t_j's. Clearly if we can compute and write down the successive segments of the t_j's on the work tape then it is an easy matter to compute the corresponding segments of the affine combinations. The computation of the t_j's differ from the nonnegativity test in only one respect—we need to perform subtractions and scalar divisions. Subtractions can be performed in segments in much the same way as additions. Examining the long division of a binary number x by a divisor d, we see that to obtain the least significant l bits of the quotient, we need not only the least significant l bits of x but also the remainder $\lfloor x/2^l \rfloor \bmod d$. Fortunately, for a fixed divisor d, this remainder can be obtained by a finite automaton computation.

Hence to obtain the ith segment of some t_j, we first compute the ith segment of the numerator ν in (1); this computation is the same as the nonnegativity test described above, except that comparisons are replaced by subtractions. At the same time, the remainders of $\lfloor y_p/2^{i\cdot s(n)} \rfloor$ and $\lfloor z_q/2^{i\cdot s(n)} \rfloor \bmod \Delta$ can be obtained in the same scan of the input arguments. These are combined with c to form $\lfloor \nu/2^{i\cdot s(n)} \rfloor \bmod \Delta$. Using this information, the machine can compute the ith segment of ν/Δ on the work tape from high-order to low-order bits. This division takes $s(n)$ time, which is dominated by the linear time used to compute the ith segment of the numerator and the remainder of $\lfloor \nu/2^{i\cdot s(n)} \rfloor \bmod \Delta$. The ith segment of the affine combination of t_j's is then generated by additions of the ith segments of the t_j's and the carry (of bounded size) from the $(i-1)$st segment, while the carry into the $(i+1)$st segment is also generated. This again takes $O(s(n))$ time. The result can then be compared against a segment of an input argument already copied onto the work tape or written out from low-order to high-order bits as appropriate. In either case, one segment takes $O(n)$ time, and the total time is $O(n^2/s(n))$ as before.

To suppress the output of "leading zeroes", the machine can go through the computation of the answer twice. The first time through no output is written; instead the last nonzero segment, i.e., the highest order segment that is not all zeroes, is determined. The second time through the segments up to the last nonzero segment (without its leading zeroes) are written out. The time complexity is not affected.

The time complexity $n^2/s(n)$ for computing P^+ is optimal in $s(n)$ space. To see this, we note that the recognition of the language

$$E = \{w \neq w : w \in \{0, 1\}^*\}$$

is essentially the computation of the 0–1 Presburger function

$$f(x, y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise.} \end{cases}$$

If the 2-way transducer can compute f simultaneously in $s(n)$ space and $t(n)$ time, then we can easily construct a 1-tape Turing machine M that recognizes E in time $t(n)s(n)$. Since E requires $O(n^2)$ time for recognition by a 1-tape Turing machine (see [9]), we must have $t(n)s(n) \geqq n^2$, i.e., $t(n) \geqq n^2/s(n)$. Furthermore, the recognition of E by a 2-tape Turing machine (which can be viewed as a 2-way transducer that gives only 0–1 outputs) requires $\log n$ space, so the $\log n$ lower bound for $s(n)$ is also optimal for the entire class P^+.

The above results can be extended to P. A program over Z with variables $x_1, \cdots, x_v$ can be simulated by a program over N with variables $x_1^+, x_1^-, \cdots, x_v^+, x_v^-$, t^+, t^-. Here it is intended that at appropriate corresponding points in the two programs, $x_i = x_i^+ - x_i^-$ with at least one of x_i^+, x_i^- being zero. Instructions are simulated as follows:

$$x \leftarrow 0 \qquad \begin{cases} x^+ \leftarrow 0 \\ x^- \leftarrow 0 \end{cases}$$

$$x \leftarrow y + z \qquad \begin{cases} t^+ \leftarrow y^+ + z^+ \\ t^- \leftarrow y^- + z^- \\ x^+ \leftarrow t^+ \dot{-} t^- \\ x^- \leftarrow t^- \dot{-} t^+ \end{cases}$$

$$x \leftarrow y - z \qquad \begin{cases} t^+ \leftarrow y^+ + z^- \\ t^- \leftarrow y^- + z^+ \\ x^+ \leftarrow t^+ \dot{-} t^- \\ x^- \leftarrow t^- \dot{-} t^+ \end{cases}$$

$x \leftarrow x + 1$ Similar to $x \leftarrow y + z$

$$x \leftarrow y/k \qquad \begin{cases} x^+ \leftarrow y^+/k \\ x^- \leftarrow y^-/k \end{cases}$$

skip l **skip** l'

$$\textbf{if } x = 0 \textbf{ then skip } l \qquad \begin{cases} \textbf{if } x^+ > 0 \textbf{ then skip } 1 \\ \textbf{if } x^- = 0 \textbf{ then skip } l' \end{cases}$$

if $x > 0$ **then skip** l **if** $x^+ > 0$ **then skip** l'

where $l' = l$ adjusted. An input of $\hat{x}_j \geqq 0$ for x_j in the original program is taken as an input of $\hat{x}_j$ for x_j^+ and 0 for x_j^- in the equivalent program, whereas if $\hat{x}_j < 0$ then the inputs are 0 for x_j^+ and $-\hat{x}_j$ for x_j^-. If the output variable is x_j in the original program then a 2-way transducer can compute both x_j^+ and x_j^- using the equivalent K^+ program, and output the appropriate value with the correct sign, being guaranteed that at least one of x_j^+ and x_j^- is zero. The same complexity results hold.

Finally we remark that the problem is no easier if we compute the answer from high-order to low-order bits. In this case, division is performed in the "natural" direction but addition and subtraction are done in the "wrong" direction, and we are again forced to "look ahead" to determine the carries or borrows from the lower-order segments. □

3. The class $\mathscr{F}(L(1))$. It is known that the sum, difference, and product of two n-bit numbers can be computed in $O(\log n)$ space, and hence also in polynomial time. For example, the following algorithm given in [12] computes the mth bit of the product

of two n-bit numbers x and y (the algorithm can easily be implemented on a two-way transducer in $O(\log n)$ space).

```
procedure Mult (x, y, m)
  ||returns the mth bit of x*y||
  carry ← 0
  do col = 0 to m
    i ← col; sum ← 0
    do j = 0 to col
      sum ← sum + x(i) * y(j)
      i ← i − 1
    end
    sum ← sum + carry
    carry ← sum/2
  end
  return least-significant-bit-of (sum)
end
```

In the algorithm, $x(i)$ and $y(j)$ are the ith and jth (least significant) bits of x and y, respectively. For instance if x has n bits then the binary representation of x would be $x(n-1), \cdots, x(1), x(0)$. Before we discuss the bit-wise space requirement of this algorithm we need the following notation.

Notation. $s(x)$ will denote the amount of space needed to obtain a bit of x. If $w(x_1, x_2, \cdots, x_m)$ is an m-ary (m fixed) operation, then $s(w)$ will denote the space needed to obtain a bit of $w(x_1, \cdots, x_m)$ as a function of $n = \max_{1 \leq i \leq m} \{|x_i|\}$, where x_i denotes the binary representation of x_i (i.e., the input has $O(n)$ bits).

For example, if we consider the binary operation of multiplication, it is clear from an analysis of the previous algorithm that for $O(n)$-bit integers x and y, $s(x*y) = c\log n + \max\{s(x), s(y)\}$ for some constant c. By including the space necessary to obtain a bit of the input as part of the space needed to perform an operation we are allowing the possibility of the operation being performed in the context of an L-program. Similarly the space needed to add or subtract two $O(n)$-bit numbers x and y is $c \log n + \max\{s(x), s(y)\}$ as was shown in [12]. The next lemma considers the composition of these operations for a function computable by an L-program that does not contain a division instruction.

LEMMA 3.1. *Any function with arguments $x_1, \cdots, x_k$ computable by an L-program P that does not contain a division instruction (except those which are division by a constant), can be computed in $c_P \log n + \max_{1 \leq i \leq k} \{s(x_i)\}$ space where $n = \max_{1 \leq i \leq k} \{|x_i|\}$ and c_P is a constant depending only on P.*

Proof. It was noted earlier that the sum, difference, and product of two n-bit numbers x, y can be computed in $\log n + \max\{s(x), s(y)\}$ space for some constant c. It is also easy to show that the integer division of an n-bit number x by a constant k can be computed in $d \log n + s(x)$ space where d is a constant (depending only on k). We also observe that the predicates "**if** $x = 0$ **then skip** l" and "**if** $x > 0$ **then skip** l" are easily simulated by a two-way transducer in $c' \log n + s(x)$ space (provided of course that x has $O(n)$ bits). Since P is a loop-free program, $O(n)$ bits are sufficient to represent the value of any variable during the execution of P. The proof of the lemma is now an easy induction on the length of P. □

We do not know whether the quotient of two arbitrary n-bit numbers can be computed in $O(\log n)$ space, although it is fairly well known that the quotient can be obtained in $O(\log^2 n)$ space. For functions computable by programs in $L(1)$ (these

are programs in L with one input variable), we can show that transducers need only $O(\log n)$ space. The proof is based on the following lemma which was recently shown in [11].

LEMMA 3.2. *Let P be a program in $L(1)$ and x be the input variable. Assume without loss of generality that x does not appear on the left-hand side of any instruction in P. Let y be any variable in P (possibly x). We can effectively construct finite sets Z and $S(y)$ whose elements are of the form T and $(p(x), T)$, respectively, where T is a finite set of pairs (m, n) of integers with $0 \leqq m < n$ and $p(x)$ is a polynomial in x with rational coefficients, such that for some computable positive integer d the following holds.*

Let x_0 be any input such that $x_0 \geqq d$. Then

(1) *A division by 0 occurs during the execution of the program if and only if there is a T in Z such that $x_0 \bmod n = m$ for all (m, n) in T.*

(2) *If the value of y is defined at the end of the program (i.e., no division by 0 occurs during the computation), then there is a unique element $(p(x), T)$ in $S(y)$ such that $x_0 \bmod n = m$ for all (m, n) in T, and the value of y is $p(x_0)$.*

From Lemmas 3.1 and 3.2, we have

THEOREM 3.1. *Functions in $\mathcal{F}(L(1))$ are computable in $O(\log n)$ space and polynomial time.*

COROLLARY 3.1. *Sums, differences, products, and quotients of polynomials in 1 variable with integer coefficients and compositions of these operations can be evaluated in $O(\log n)$ space and polynomial time.*

4. The classes $\mathcal{F}(L)$ and $\mathcal{F}(M)$. It is fairly well known that integer division with truncation can be computed in $O(\log^2 n)$ space. To see this, suppose we wish to compute x/y, where x has at most kn bits and y has exactly $n \geqq 2$ bits. Let α be a kn-bit approximation to the fraction $1/y$. Then $\hat{q} = \lfloor \alpha * x \rfloor$ is a good approximation to the quotient x/y, and $\hat{q}$ can be corrected slightly to obtain the correct quotient q satisfying $0 \leqq x - q*y < y$.

The approximation α can be found using Newton's method (see [1], [8], [13]). Applying Newton's formula

$$\alpha_{i+1} = \alpha_i - \frac{f(\alpha_i)}{f'(\alpha_i)}$$

to the function $f(v) = 1/v - y$ yields the iteration formula $\alpha_{i+1} = 2\alpha_i - y\alpha_i^2$ for obtaining the $(i+1)$st approximation to $1/y$ in terms of the ith approximation. The iteration formula converges quadratically. Hence, the digits of α can be obtained in $O(\log n)$ iterations. The iteration formula $\alpha_{i+1} = 2\alpha_i - y\alpha_i^2$ can now be used to write a recursive procedure ALPHA (y, i, j) which computes the jth bit of α_i (see [1], [13] for similar procedures). Note that y is global and only i and j are passed from one level of recursion to the next. The body of the procedure makes recursive calls of the form ALPHA$(y, i-1, j)$ and uses the routines for computing the digits of the sum, difference, and product of numbers as discussed in § 3. (Variables are used to store the location of the binary point.) Clearly, the recursive procedure can be written so that each variable occurring in the body of the program needs only $O(\log n)$ bits of storage. Since the maximum depth of recursion is $i = O(\log n)$, the program can be implemented on a Turing machine with a pushdown store of size $O(\log n) * O(\log n) = O(\log^2 n)$.

It follows from the above discussion and Lemma 3.1 that the digits of α and therefore also of x/y can be computed in $O(\log^2 n)$ space. Thus, functions in $\mathcal{F}(L)$ are computable in $O(\log^2 n)$ space. This result can be generalized to functions compu-

table by programs which use instructions of the form $x \leftarrow \lfloor \sqrt[k]{y} \rfloor$. To compute $\lfloor \sqrt[k]{y} \rfloor$, where k is a positive integer $\geqq 2$ and y is a positive integer with $n \geqq 2$ bits, we first compute an approximation α to $1/\sqrt[k]{y}$ using the iteration formula[3]

$$\alpha_{i+1} = \left(1 + \frac{1}{k}\right)\alpha_i - \frac{1}{k} y \alpha_i^{k+1}.$$

Clearly, the digits of α can be found in $O(\log^2 n)$ space. Let β be an approximation to $1/\alpha$. Then β can be obtained from α in $O(\log^2 n)$ space.[4] The integral part $\hat{p}$ of β is a good approximation to $\lfloor \sqrt[k]{y} \rfloor$. As before, $\hat{p}$ can be corrected slightly to obtain the correct integer p such that p is the largest integer satisfying $p^k \leqq y$. We omit the details.

THEOREM 4.1. *Functions in $\mathcal{F}(M)$ are computable in $O(\log^2 n)$ space.*

COROLLARY 4.1. *For each positive integer k, let $A_k = \{x^k \mid x \text{ in } N\}$. (Thus, A_2 is the set of perfect squares.) Then A_k is recognizable by a deterministic Turing machine in $O(\log^2 n)$ space.*

Open problems.

(1) Can x/y be computed in less than $O(\log^2 n)$ space?

(2) Can A_2 be recognized in less than $O(\log^2 n)$ space?

Finally, we note that if time is the measure of complexity (instead of space), then the operations of squaring, division, and finding kth roots are computationally time-equivalent to multiplication [1], [2]. Hence, every function in $\mathcal{F}(M)$ has time complexity $O(g(n))$, where $g(n)$ is the complexity of multiplication.

Acknowledgments. We would like to thank Janos Simon and Lou Rosier for helpful comments concerning the results of the paper. We learned after submitting the paper to this journal that a technique similar to the one we used for the class $\mathcal{F}(M)$ has also been used in [16] to show an $O(\log^2 n)$ bound on the delay complexity of square rooting in combinational logic circuits. We thank Al Borodin for bringing this to our attention.

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] H. ALT, *Functions equivalent to integer multiplication*, Automata, Languages and Programming Proceedings, J. de Bakker and J. van Leeuwen, eds. Lecture Notes in Computer Science, 85, Springer, New York, 1980, pp. 30–37.

[3] S. A. COOK, *Deterministic CFL's are accepted simultaneously in polynomial time and log squared space*, Proc. 11th Annual ACM Symposium on Theory of Computing, 1979, pp. 338–345.

[4] P. C. FISCHER, A. R. MEYER AND A. L. ROSENBERG, *Counter machines and counter languages*, Math. Systems Theory, 2 (1968), pp. 265–283.

[5] S. GINSBURG AND E. SPANIER, *Semigroups, Presburger formulas, and languages*, Pacific J. Math., 16 (1966), pp. 285–296.

[6] E. M. GURARI AND O. H. IBARRA, *The complexity of the equivalence problem for two characterizations of Presburger sets*, Theoret. Comput. Sci., 13 (1981), pp. 295–314.

[3] Obtained by applying Newton's method to the function $f(v) = 1/v^k - y$.

[4] An approximation β to $\sqrt[k]{y}$ could be obtained directly using the formula

$$\beta_{i+1} = \left(1 - \frac{1}{k}\right)\beta_i + \frac{y}{k\beta_i^{k-1}}.$$

However, this formula would require more than $O(\log^2 n)$ space for computing the digits of β.

[7] ———, *An NP-complete number-theoretic problem*, J. Assoc. Comput. Mach., 26 (1979), pp. 567–581.
[8] J. HARTMANIS AND J. SIMON, *On the power of multiplication in random access machines*, Proc. 15th SWAT Symposium, 1974, pp. 13–23.
[9] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.
[10] O. H. IBARRA AND B. S. LEININGER, *Characterizations of Presburger functions*, this Journal, 10 (1981), pp. 22–39.
[11] ———, *Straight-line programs with one input variable*, this Journal, 11 (1982), pp. 1–14.
[12] J. JA' JA' AND J. SIMON, *Some space-efficient algorithms*, Proc. 17th Allerton Conference, 1979, pp. 677–684.
[13] D. E. KNUTH, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.
[14] P. M. LEWIS, R. E. STEARNS AND J. HARTMANIS, *Memory bounds for recognition of context-free and context-sensitive languages*, IEEE Conference on Recognition and Switching Theory, 1965, pp. 191–202.
[15] B. VON BRAUNMÜHL AND R. VERBEEK, *A recognition algorithm for deterministic CFLs optimal in time and space*, IEEE 21st Annual Symposium on Foundations of Computer Science, 1980, pp. 411–420.
[16] H. YASUURA AND S. YAJIMA, *On the delay complexity of square rooting in combinational logic circuits*, Tech. Rep. No. AL79-29, Institute of Electronics and Communication Engineering of Japan, 1979, pp. 27–35.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0009 $01.25/0

AN ANALYSIS OF BACKTRACKING WITH SEARCH REARRANGEMENT*

PAUL WALTON PURDOM, JR.† AND CYNTHIA A. BROWN‡

Abstract. The search rearrangement backtracking algorithm of Bitner and Reingold [Comm. ACM, 18 (1975), pp. 651–655] introduces at each level of the backtrack tree a variable with a minimal number of remaining values; search order may differ on different branches. For conjunctive normal form formulas with v variables, s literals per term ($s \geqq 3$), and v^α terms ($(s/2) < \alpha < s$), the average number of nodes in a search rearrangement backtrack tree is $\exp[\Theta(v^{(s-\alpha-1)/(s-2)})]$ (i.e., for some positive constants a_1, a_2, and v_0, when $v \geqq v_0$ the number of nodes is between $\exp(a_1 v^{(s-\alpha-1)/(s-2)})$ and $\exp(a_2^{(s-\alpha-1)/(s-2)})$. For $1 < \alpha \leqq s/2$ the average number of nodes is between $\exp[\Theta(v^{(s-\alpha-1)/(s-2)})]$ and $\exp[\Theta((\ln v)^{(s-1)/(s-2)} v^{(s-\alpha-1)/(s-2)})]$. This compares with $\exp[\Theta(v^{(s-\alpha)/(s-1)})]$ for ordinary backtracking. For $1 < \alpha < s$, simple search rearrangement has approximately the same effect on speeding up backtracking as does reducing the problem complexity by decreasing the number of literals per term by one. Thus simple search rearrangement backtracking leads to a dramatic improvement in the expected running time.

Key words. analysis of algorithms, average time analysis, backtracking, NP-complete, search algorithms

1. Introduction.

Many problems can be regarded as a search for all the solutions to an equation of the form $\mathbf{P}(x_1, \cdots, x_v) = \text{true}$, where $\mathbf{P}$ is a v-ary predicate over an ordered set of variables $S = \{x_i\}_{1 \leqq i \leqq v}$, and each x_i has finitely many possible values. A straightforward, but exponentially costly, way to solve such a problem is to generate and test each combination of values of the variables. For most problems backtracking can reduce the amount of time required to find the solutions.

To perform backtracking it is necessary to have, in addition to the problem predicate $\mathbf{P}$, an *intermediate predicate* $\mathbf{P}_A$ for each subset A of S. The predicate $\mathbf{P}_A$ must have the value true for any assignment of values to the variables in A that can be extended to a solution to $\mathbf{P}$. An intermediate predicate is powerful if it is false for most other assignments of values. Powerful intermediate predicates that can be calculated efficiently eliminate false starts quickly, making backtracking feasible for large problems.

In backtracking, each variable starting with the first is set to its initial value. As each variable is set, the appropriate intermediate predicate is tested. If it is true, the next variable is set. If it is false, the current variable is reset to its next value; if the current variable has no more values, it is removed from the set of variables with values, and the previous variable is set to its next value. Knuth [4] gives a good introduction to backtracking.

Bitner and Reingold [1] consider a modification of backtracking which we call (simple) search rearrangement backtracking. (For other modifications, see [6].) Instead of setting the variables in a fixed order, the unset variables are tested at each step to select one that produces the fewest true values for the corresponding intermediate predicate. Bitner and Reingold showed experimentally that this search rearrangement process improves the efficiency of backtracking. In this paper we study search rearrangement backtracking analytically, and compare its performance with that of ordinary backtracking, which we analyzed in [2].

* Received by the editors December 23, 1980, and in final revised form September 23, 1982. This research was supported in part by the National Science Foundation under grant MCS 79-06110.

† Computer Science Department, Indiana University, Bloomington, Indiana 47405.

‡ Computer Science Department, Indiana University, Bloomington, Indiana 47405 (on leave) and GTE Laboratories, Waltham, Massachusetts 02254.

2. Model and notation. To compare backtracking methods, we analyze their average performance on the problem of finding all solutions to conjunctive normal form formulas having t terms and s literals per term over a set S of v variables. Duplication is permitted in the terms for a predicate and in the literals within a term, so there are $(2v)^{st}$ predicates in the set. For each predicate $\mathbf{P}$ and set of variables $A \subseteq S$, the intermediate predicate $\mathbf{P}_A$ is the conjunction of those terms of $\mathbf{P}$ for which all the variables are in the set A. (If there are no such terms, then $\mathbf{P}_A$ is by definition true.) These intermediate predicates are natural and efficiently calculable. The set of predicates is NP complete (for $s \geqq 3$, $t > v$, and v increasing), so some problems in the set are hard. We find that the backtrack trees for most of the problems in this set are similar to those encountered in realistic problems, and this set of problems is suitable for analysis. We therefore believe these problems are a good model for our study of backtracking.

For purposes of illustration we use the predicate $E = T_1 \wedge T_2 \wedge T_3 \wedge \cdots \wedge T_{11}$, where $T_1 = (v_1 \vee v_1 \vee \neg v_2)$, $T_2 = (v_1 \vee v_1 \vee \neg v_5)$, $T_3 = (v_1 \vee \neg v_3 \vee v_5)$, $T_4 = (v_1 \vee v_2 \vee \neg v_6)$, $T_5 = (\neg v_1 \vee \neg v_1 \vee v_2)$, $T_6 = (\neg v_1 \vee \neg v_1 \vee v_2)$, $T_7 = (\neg v_1 \vee v_3 \vee v_4)$, $T_8 = (\neg v_1 \vee v_3 \vee v_5)$, $T_9 = (\neg v_1 \vee \neg v_4 \vee \neg v_5)$, $T_{10} = (\neg v_1 \vee \neg v_3 \vee v_5)$, and $T_{11} = (\neg v_1 \vee v_4 \vee v_6)$. Notice the duplication in the literals within a clause (as in $(v_1 \vee v_1 \vee \neg v_2)$) and in the clauses in the predicate ($T_5 = T_6$). Intermediate predicate $E_{\{1,2,5\}}$ is $T_1 \wedge T_2 \wedge T_5 \wedge T_6$; intermediate predicate $E_{\{1,3\}}$ is the empty predicate, which is identically true.

Simple search rearrangement backtracking is done as follows. Let S' be the set of variables without values (the unset variables) and S'' the set of variables with values. Let w range over the problem variables; denote the value of problem variable w by Value $[w]$ for $1 \leqq w \leqq v$. The set S'' is maintained as a stack.

SIMPLE SEARCH REARRANGEMENT.

Step 1. (Initialize.) Set S'' to empty and S' to S.

Step 2. (Solution?) If S' is not empty, go to Step 3. Otherwise, the current values in Value constitute a solution. Go to Step 6.

Step 3. (Find best variable.) For each variable w in S' do the rest of this step. (The order in which the variables are tested is immaterial for our purposes.) For both Value $[w] \leftarrow$ false and Value $[w] \leftarrow$ true, compute $\mathbf{P}_{S'' \cup \{w\}}$. If the result is false in both cases, exit the loop for w and go to Step 6. If it is false in one and true in the other, remember the value of w that gives true and exit the loop for w and go to Step 5.

Step 4. (Binary node.) Let w be the smallest element of S'. Set $S'' \leftarrow S'' \cup \{w\}$ and $S' \leftarrow S' - \{w\}$. Set Value $[w] \leftarrow$ false, mark w as binary, and go to Step 2.

Step 5. (Unary node.) Set Value $[w]$ to the unique value that makes $\mathbf{P}_{S'' \cup \{w\}}$ true. (This value is remembered from Step 3.) Set $S'' \leftarrow S'' \cup \{w\}$ and $S' \leftarrow S' - \{w\}$. Mark w as unary, and go to Step 2.

Step 6. (Next value.) If S'' is empty, stop. Otherwise, set $w \leftarrow$ top (S''). If w is marked as binary and Value $[w]$ = false, set Value $[w] \leftarrow$ true and go to Step 2.

Step 7. (Backtrack.) Set $S'' \leftarrow S'' - \{w\}$, $S' \leftarrow S' \cup \{w\}$, and go to Step 6.

Figure 1 shows the backtrack tree obtained by applying the simple search rearrangement algorithm to E, with the convention that unset variables are tested in numerical order according to their subscripts. Initially, no variables are set. The first time Step 3 is executed it does not find a variable with zero or one values, so in Step 4 v_1 is selected and set to its first value (false). Returning to Step 3 by way of Step 2, testing $E_{\{1,2\}} = T_1 \wedge T_5 \wedge T_6$ shows that setting v_2 to true makes the predicate false.

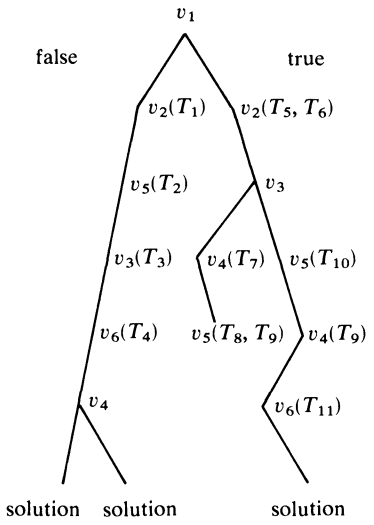


FIG. 1. *The backtrack tree obtained by applying a simple search rearrangement backtrack algorithm to predicate E. Each interior node is labelled with the name of the variable that is set at that node. At nodes where a variable is forced to take on a particular value, the clause or clauses that cause the forcing are shown in parentheses. At the leaf nodes where a variable with no values is discovered, the clauses that eliminate both values are shown.*

Therefore v_2 is selected and set to false in Step 5. In a similar way, v_5, v_3, and v_6 are forced to assume the value false. The only remaining variable, v_4, is not constrained and so is not selected in Step 3; it is selected as a binary variable in Step 4, leading to two solutions. The remainder of the backtrack tree is produced in a similar way.

We will concentrate on the number of binary nodes in the average search rearrangement backtrack tree. Let $N_{\mathbf{P}}$ be the number of binary nodes in the backtrack tree for predicate **P**. The total number of nodes in the tree is between $N_{\mathbf{P}}$ and $(2v-1)N_{\mathbf{P}}$. Assuming that for each $A \subseteq S$, $\mathbf{P}_A$ can be evaluated in constant time, the total running time of the algorithm is between $c_1 N_{\mathbf{P}}$ and $c_2 v^2 N_{\mathbf{P}}$ for some constants c_1 and c_2. We compute N, the average value of $N_{\mathbf{P}}$. The average values for both the total number of nodes and the total running time are N times some polynomially bounded function of v.

In [3] we give the results of a statistical study of the ratio of the number of binary nodes to the number of unary nodes and to the number of predicate evaluations.

3. The average number of binary nodes. We now derive an exact formula for N, the average number of binary nodes. The exact formula has an exponential number of terms, so we also derive upper and lower limits with a polynomial number of terms.

Our method is to consider each possible binary node and to count the predicates that have that node in their backtrack trees. Summing over all binary nodes and dividing by the number of predicates ($(2v)$st) gives the average number of binary nodes in a tree.

We begin by counting the predicates for which a particular variable w occurs as a binary node with all the other variables in S' set to false. (Later we multiply by an appropriate factor to allow for other settings of the variables.) For this to occur, the algorithm must reach Step 4 with every variable less than w set to false, with the value of w unconstrained, and with none, some, or all of the variables greater than w set to false. Any variable $y > w$ which is set to false must have been set in Step 5:

it must have been forced to assume the value false by a clause made up of literals with truth value false, the literal $\neg y$, and nothing else. The variables less than w may have been forced to false in Step 5 or may correspond to a binary node where the false branch is being explored; the step at which these variables received their values does not affect the analysis.

Let q_0 equal the number of variables that precede w (recall that they are all set to false). Let q_1 equal q_0 plus the number of variables whose values are forced to false when just the first q_0 variables are set. Let q_i be q_{i-1} plus the number of variables whose values are forced to false by the q_{i-1} variables. Let m $(0 \leqq m < v)$ be the smallest value of i such that $q_{i+1} = q_i$. We call m the number of rounds of forcing.

To illustrate, consider the binary node for v_4 on the left branch of the backtrack tree in Fig. 1. Variables v_1, v_2, and v_3 precede v_4 in the ordered set of variables, so $q_0 = 3$. With these variables set to false, v_5 is forced to false by T_2 and v_6 is forced to false by T_4. Thus $q_1 = 5$. Setting v_5 and v_6 to false does not force any more variables, so $m = 1$. (A second round of forcing would have occurred if the predicate had contained a term such as $(v_5 \vee v_6 \vee \neg v_7)$.) Notice that the rounds of forcing used in the analyses do not necessarily correspond to the order in which variables are set by the algorithm. They are a device to facilitate counting the predicates that have a particular binary node. The analysis starts with the assumption that a node such as v_4 is binary and counts the number of ways in which this can happen.

Assume that the variables forced during rounds 1 through m are consecutive variables. (We later multiply by a factor to account for the number of ways of choosing these variables.) For the rest of this section let the variables be renumbered so that the variable for the binary node is q_0, so that the forced variables start with $q_0 + 1$. Recall that each predicate in the class we are considering has t clauses. For purposes of counting we divide these clauses into two classes. Given m and $q_0, q_1, \cdots, q_m$, the predicate must contain clauses that force the variables from $q_0 + 1$ to q_m to the value false. We call these *required* clauses. For $1 \leqq i \leqq m$ and $1 \leqq x \leqq q_i - q_{i-1}$, let $j_{ix} \geqq 1$ be the number of clauses that force variable $q_{i-1} + x$ to false, given only that the first q_{i-1} variables are set to false. Notice that for $(i, x) \neq (i', x')$ the clauses counted by j_{ix} are disjoint from those counted by $j_{i'x'}$.

Remaining clauses in the predicate are called *permitted* clauses. Permitted clauses are any clauses other than required clauses that are compatible with the assumptions we make about the node we are considering. Whether a clause is permitted or required depends on the binary node being considered. The same clause might be a permitted clause for one node and a required clause for another. In predicate E, when v_1 is set to false and $w = v_4$ the variables v_2, v_3, v_5 and v_6 are forced. Since v_2 and v_3 precede v_4 we do not care whether they were forced or set, and so the clauses that force them are not required clauses. Variable v_5 is forced by T_2, and v_6 by T_4, so $j_{11} = 1$ and $j_{12} = 1$. The rest of the clauses are permitted clauses.

The initial counting arguments are summarized in Table 1. First consider the required clauses for round i of forcing. Variable $q_{i-1} + x$ $(1 \leqq x \leqq q_i - q_{i-1})$ is forced to false by clauses containing some of the first q_{i-1} variables, all unnegated, and the negated literal for the forced variable. Since clauses containing only the first q_{i-1} variables (and not the forced variable) are not suitable, there are $Q_{i-1} = (q_{i-1} + 1)^s - {q_{i-1}}^s$ such clauses. The Q_{i-2} of these that contain only the first q_{i-2} variables and the forced variable are also unsuitable, since they would force the variable on an earlier round than the ith. Therefore there are $R_{i-1} = Q_{i-1} - Q_{i-2}$ clauses that force variable $q_{i-1} + x$ on round i. (Use 0 for Q_{-1}.) For each variable $q_{i-1} + x$ forced on round i, the predicate contains j_{ix} clauses chosen (with replacement) from a set of

R_{i-1} clauses; the sets corresponding to distinct values of x are disjoint. The total number of required clauses in the predicate is $\sum_{i,x} j_{ix}$.

The remaining clauses in the predicate (the permitted clauses) can be selected from any of the $(2v)^s$ clauses except the following. There are q_m^s clauses made up entirely of false (unnegated) literals for the first q_m variables. A predicate containing such a clause would be false and would not reach Step 4 under our assumptions. For each of the $(v-q_m)$ unset variables, there are Q_m clauses that would force it to true on an $(m+1)$st round of forcing. This accounts for $(v-q_m)Q_m$ clauses. Immediately after round i $(0 \leqq i \leqq m)$ there are $(v-q_i)$ unset variables and R_i clauses that would force them to false on round $i+1$. (We exclude the required clauses from the class of permitted clauses, along with unwanted forcing clauses. The required clauses are counted explicitly by the other factors of the formula.) This accounts for $\sum_{0 \leqq i \leqq m} (v-q_i)R_i$ clauses.

Much of the complexity of our analysis results from dividing the forcing into rounds. This division is necessary in order to count the required and permitted clauses correctly. A variable can only be forced by being in a clause where all variables for the other literals have already received values. One of the approximate formulas we present later was obtained by dropping the requirement that variables be forced in a legal order.

The total number of predicates that satisfy our current assumptions is

$$P^{t-\sum_{i,x} j_{ix}} \prod_{1 \leqq i \leqq m} R_{i-1}^{\sum_x j_{ix}},$$

where

$$\begin{aligned} P &= (2v)^s - q_m^s - (v-q_m)Q_m - \sum_{0 \leqq i \leqq m} (v-q_i)R_i \\ &= (2v)^s - q_m^s - 2(v-q_m)Q_m - \sum_{0 \leqq i \leqq m} (q_{i+1}-q_i)Q_i. \end{aligned}$$

The total number of binary nodes in all the $(2v)^{st}$ backtrack trees, which is $(2v)^{st}N$, can be obtained by multiplying this formula by the appropriate factors and summing. The result is

$$\begin{aligned} (2v)^{st}N = \sum_{0 \leqq m < v} \sum_{\substack{0 \leqq q_0 < q_1 < \cdots \\ < q_m < v}} \sum_{\substack{j_{11} \neq 0, \cdots \\ j_{m,q_m - q_{m-1}} \neq 0}} & 2^{q_m} \binom{t}{j_{11}, \cdots, j_{m,q_m-q_{m-1}}, t - \sum_{i,x} j_{ix}} \\ & \cdot \binom{v-q_0-1}{q_1-q_0, \cdots, q_m-q_{m-1}, v-q_m-1} P^{t-\sum_{i,x} j_{ix}} \prod_{1 \leqq i \leqq m} R_{i-1}^{\sum_x j_{ix}}, \end{aligned}$$

where the factor 2^{q_m} accounts for the number of ways to assign values to the q_m set variables, the first multinomial accounts for the number of ways to order the terms ($t_1 \wedge t_2$ is different from $t_2 \wedge t_1$, for $t_1 \neq t_2$), and the second multinomial accounts for the number of ways to select which variables are forced on each of the m rounds. The initial q_0 variables and the variable for the binary node at Step 4 are not available for selection. The sums over j require that there is at least one term to force each forced variable. The sums over the q_i require that at least one variable be forced on each round.

We adopt the convention that limits on a summation variable are not shown explicitly when we intend that the sum be taken over all values of the variable that result in a nonzero value of the summand. If, for example, the variable appears as

the bottom of a binomial coefficient, the implied range is between zero and the value of the top of the binomial.

The sums over the j_{ix} can be done using the binomial theorem and subtracting the $j_{ix}=0$ term. The sum over all the j_{1x} for $1\leqq x\leqq q_1-q_0$ is

$$(2v)^{st}N=\sum_{0\leqq m<v}\sum_{\substack{0\leqq q_0<q_1<\\ \cdots<q_m<v}}\sum_{\substack{j_{21}\neq 0\cdots\\ j_{m,q_m-q_{m-1}}\neq 0}}\sum_{i_1}2^{q_m}\binom{q_1-q_0}{i_1}(-1)^{q_1-q_0+i_1}$$
$$\cdot\binom{t}{j_{21},\cdots,j_{m,q_m-q_{m-1}},t-\sum_{i>1,x}j_{ix}}$$
$$\cdot\binom{v-q_0-1}{q_1-q_0,\cdots,q_m-q_{m-1},v-q_m-1}(P+i_1R_0)^{t-\sum_{i>1,x}j_{ix}}\prod_{2\leqq i\leqq m}R_{i-1}^{\sum_x j_{ix}}.$$

Summing over all j_{ix} and combining the binomials with the second multinomial gives

$$(2v)^{st}N=\sum_{0\leqq m<v}\sum_{\substack{0\leqq q_0<q_1<\\ \cdots<q_m<v}}\sum_{i_1,\cdots,i_m}2^{q_m}(-1)^{q_m-q_0+\sum_n i_n}$$
$$\cdot\binom{v-q_0-1}{q_1-q_0-i_1,\cdots,q_m-q_{m-1}-i_m,i_1,\cdots,i_m,v-q_m-1}\tag{1}$$
$$\cdot\left(P+\sum_{1\leqq n\leqq m}i_nR_{n-1}\right)^t.$$

The sum $\sum_{1\leqq n\leqq m}i_nR_{n-1}$ equals $\sum_{1\leqq n\leqq m}(i_n-i_{n+1})Q_{n-1}$, where $i_{m+1}\equiv 0$.

4. Lower and upper limits. Formula (1) is not very useful for $v\gg 10$ because the number of terms increases exponentially with increasing v. We obtain a lower limit on its value by noticing that, for each fixed value of m, the corresponding partial sum is positive. (It equals the number of binary nodes that are immediately preceded by m rounds of forcing.) The *k-sum lower limit* is obtained by replacing $\sum_{0\leqq m<v}$ in formula (1) by $\sum_{0\leqq m<k}$. In particular, the one-sum lower limit is

$$(2v)^{st}N\geqq\sum_{0\leqq q_0<v}2^{q_0}P_0^t,\tag{2}$$

where $P_0=(2v)^s-q_0^s-2(v-q_0)Q_0$, and the two-sum lower limit is

$$(2v)^{st}N\geqq\sum_{0<q_0\leqq q_1<v}\sum_{i_1}2^{q_1}(-1)^{q_1-q_0+i_1}\binom{v-q_0-1}{q_1-q_0-i_1,i_1,v-q_1-1}(P_1+i_1Q_0)^t.\tag{3}$$

where $P_1=(2v)^s-q_1^s-2(v-q_1)Q_1-(q_1-q_0)Q_0$. The $m=0$ and $m=1$ sums have been combined.

Upper limits can be obtained by allowing "sloppy counting" of required and permitted clauses. If some permitted clauses are misclassified as required clauses, the total number of predicates that meet the criteria increases: all the original predicates are still counted (though some of their clauses are put in a different class) and some additional, spurious predicates are also included. If some clauses are classified as both permitted and required, the number of spurious predicates increases even more. Another simplification that increases the size of the sum is to disregard the necessity for legal rounds of forcing. We combine these approaches to obtain upper limits with polynomially many terms.

The *one-sum upper limit* is obtained by ignoring altogether the idea of required vs. permitted terms and the idea of rounds of forcing. All terms that are not false and that do not force unset variables are used, and all variables except the one for the y node are available for forcing. Once the variable for the binary node is chosen (and choosing it accounts for the factor of v in Formula 4) there are $v-1$ variables available to be set or forced, or left unset (choosing these accounts for the factor of $\binom{v-1}{q_0}$). If q_0 variables are being set or forced, there are 2^{q_0} branches. Finally, a factor P_0 is needed to count the number of legal clauses under these assumptions. The derivation of P_0 is summarized in Table 2; $P_0=(2v)^s-q_0^s-2(v-q_0)Q_0$, and P_0^t is the number of predicates made up of such clauses. The final formula is

$$(2v)^{st}N \leqq \sum_{0\leqq q_0<v} v2^{q_0}\binom{v-1}{q_0}P_0^t. \tag{4}$$

Notice that the factor of v was needed here because the q_0 variables that are set or forced are chosen arbitrarily and do not necessarily precede the variable for the binary node, nor does the value of q_0 determine which variable is the one for the binary node. In all our other formulas variable q_0+1 (before renumbering) is the variable for the binary node and so the factor of v is not needed.

In the *two-sum upper limit* we have q_1-q_0 sets of required clauses, each with $q_1^s-(q_1-1)^s$ elements (and zero elements when $q_1=0$). Here we ignore the different rounds of forcing: each set has all the clauses that can force a variable if it is the last one to be forced (that is, the clauses contain literals of the other forced variables). Thus we over-count the actual number of forcing clauses available. These required sets are not disjoint; they contain $q_1^s-q_0^s$ terms in all. Table 3 summarizes the initial analysis. The final formula is

$$(2v)^{st}N \leqq \sum_{0\leqq q_0\leqq q_1<v}\sum_{i_1} 2^{q_1}(-1)^{q_1-q_0+i_1}\binom{v-q_0-1}{q_1-q_0-i_1,\, i_1,\, v-q_1-1} \cdot [P_u+i_1[q_1^s-(q_1-1)^s]]^t, \tag{5}$$

where $P_u=(2v)^s-2(v-q_1)Q_1-2q_1^s+q_0^s$. In this formula q_1 plays a role similar to that of q_m in Formula (1); the role played by q_0 is similar in both formulas.

Another upper limit can be obtained from the analysis in Table 4. It leads to the following formula:

$$(2v)^{st}N \leqq \sum_{0\leqq q_0\leqq q_1<v}\ \sum_{\substack{0\leqq i_1\leqq 1,\cdots,\\ 0\leqq i_{q_1-q_0}\leqq 1}} 2^{q_1}(-1)^{q_1-q_0+\sum_n i_n}\frac{(v-q_0-1)!}{(v-q_1-1)!} \cdot [(2v)^s-2(v-q_1)Q_1-2q_1^s+q_0^s + \sum_{1\leqq n\leqq q_1-q_0} i_n[(q_0+i_n)^s-(q_0+i_n-1)^s]]^t.$$

We do not, however, analyze the asymptotic behavior of this formula.

The two-sum upper limit can be improved in various ways. For example, one can treat the first few rounds of forcing exactly and the remaining rounds approximately. Another approach would have an approximate treatment for forcing the first half of the forced variables, followed by an approximate treatment for the second half. We have not investigated which of these methods gives the most precise answer for a fixed amount of computation.

5. Asymptotic analysis. We now consider the asymptotic behavior of the formulas for the one- and two-sum lower and upper limits (four cases), holding s fixed, letting $t = v^{\alpha}$ for a fixed α, and allowing v to become large. We find only the leading term in the exponential dependence of the result.

In each of the four formulas (once the sum over i_1 is done, if necessary) the sums contain no more than v^2 terms, where each term is positive. Therefore, to the required accuracy (ignoring polynomial factors), each sum is equal to the largest term. Most details of the analysis are omitted; they are similar to those in [2]. We assume $s \geqq 3$ and $1 < \alpha < s-1$.

Briefly, the procedure is: (1) sum over i_1 (if necessary); (2) expand multinomials into factorials and use Stirling's approximation for the factorials; (3) take the logarithm of the summand; (4) take the derivatives of the logarithm (with respect to q_0 and to q_1) and set them to zero; (5) solve the equations asymptotically to find the values of q_0 and q_1 that maximize the summand; (6) substitute the values of q_0 and q_1 into the log of the summand to find the log of the maximum term; and (7) obtain the final value by increasing the error term to $\log v$ (if necessary) to allow for missing polynomial factors and exponentiate the result. Do 40 pages of calculations without error and obtain the results given below. (Appendix 2 outlines the necessary steps.)

The one-sum lower limit summand is maximized with

$$q_0 = \left(\frac{2^{s-1} \ln 2}{s(s-1)}\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)} + \Theta(v^{(s-2\alpha)/(s-2)}) + \Theta(1).$$

This gives

$$N \geqq \exp\left[\left(\frac{2 \ln 2}{s(s-1)}\right)^{1/(s-2)} \frac{2(s-2)\ln 2}{s-1} v^{(s-\alpha-1)/(s-2)} + \Theta(v^{(s-2\alpha)/(s-2)}) + \Theta(1)\right]. \tag{6}$$

The one-sum upper limit summand is maximized with

$$q_0 = 2\left(\frac{2 \ln F(v)}{s(s-1)(s-2)}\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)} + \Theta((\ln F(v))^{2/(s-2)} v^{(s-2\alpha)/(s-2)}) + \Theta(1),$$

where $F(v)$ is the solution to the equation

$$F(v) = \frac{a(v)}{\ln F(v)} \quad \text{with } a(v) = \frac{s(s-1)(s-2)}{2} v^{\alpha-1}.$$

For any $\varepsilon > 0$ we have for large v

$$\frac{a(v)}{\ln a(v)} < F(v) < \frac{a(v)}{\ln a(v)}\left(1 + (1+\varepsilon)\frac{\ln \ln a(v)}{\ln a(v)}\right),$$

so $\ln F(v) \sim (s-1) \ln v$, where $x(v) \sim y(v)$ means $\lim_{v\to\infty} x(v)/y(v) = 1$. This gives

$$\begin{aligned} N \leqq \exp\Bigg[& \frac{2}{s-1}\left(\frac{2}{s(s-1)(s-2)}\right)^{1/(s-2)} (\ln F(v))^{(s-1)/(s-2)} v^{(s-\alpha-1)/(s-2)} \\ & + 2\left(\frac{2}{s(s-1)(s-2)}(\ln F(v))\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)} \\ & + \Theta((\ln F(v))^{s/(s-2)} v^{(s-2\alpha)/(s-2)}) + \Theta(\ln v)\Bigg]. \end{aligned} \tag{7}$$

The leading term in the exponent for the one-sum upper limit is larger than the corresponding term in the one-sum lower limit by the slowly increasing factor $(\ln F(v))^{(s-1)/(s-2)}$.

The two-sum cases require approximating a sum of the form

$$\sum_i \binom{a}{i}(-1)^i(1+ic)^t = (-1)^a \sum_j \binom{t}{j}\left\{{j \atop a}\right\} a!c^j,$$

where a, c, and t are functions of v, q_0, and q_1 ($t = v^\alpha$), and where $\{{j \atop a}\}$ is a Stirling number of the second kind [4]. Since each term on the right side is positive, a lower limit is the first nonzero term ($j = a$). The first nonzero term is also a good approximation for the total sum if the quantity act approaches zero as v becomes large (in our application this happens when $\alpha > s/2$). For the two-sum lower limit we use

$$(-1)^a \sum_i \binom{a}{i}(-1)^i(1+ic)^t > \binom{t}{a} a!c^a.$$

The two-sum lower limit summand is (approximately) maximized with

$$q_0 = \frac{2(s-1)}{y+s-1-\ln 2}\left[\frac{2y}{s(s-1)}\right]^{1/(s-2)} v^{(s-\alpha-1)/(s-2)},$$

$$q_1 = 2\left[\frac{2y}{s(s-1)}\right]^{1/(s-2)} v^{(s-\alpha-1)/(s-2)},$$

where y is the solution of the equation

$$y + \ln\left(1-\frac{\ln 2}{y}\right) + (s-2)\ln\left(1+\frac{v-\ln 2}{s-1}\right) = 0.$$

The value of y is between $\ln 2$ and $\ln 3$ for all $s \geqq 3$; $y \simeq 1.0106135875$ for $s = 3$, and $y \simeq 0.9965516271$ for $s = 4$. This gives $y \to \ln 2$. This gives

$$N \geqq \exp\left[2\left[\frac{2y}{s(s-1)}\right]^{1/(s-2)}\left[\ln 2 + \frac{y-\ln 2}{y+s-1-\ln 2}(2-y\ln 2) - \frac{y}{(s-1)}\right]\right.$$
$$\left. \cdot\, v^{(s-\alpha-1)/(s-2)} + \Theta(v^{(s-2\alpha)/(s-2)}) + \Theta(\ln v)\right]. \tag{8}$$

It is difficult to obtain an asymptotic upper limit from limit (5) due to the problem of approximating the sum over i, for values of $q_1 \geqq v^{(s-\alpha)/s}$. Combining the derivation of limits (4) and (5), however, gives

$$(2v)^{st} N \leqq \sum_{0 \leqq q_0 \leqq q_1 < q_*} \sum_{i_1} 2^{q_1}(-1)^{q_1-q_0+i_1} \binom{v-q_0-1}{q_1-q_0-i_1,\, i_1,\, v-q_1-1}$$
$$\cdot\, [P_u + i_1[q_1^s - (q_1-1)^s]]^t + v \sum_{q_* \leqq q_0 < v} 2^{q_0}\binom{v-1}{q_0} P_0^t. \tag{9}$$

Using $i' = q_1 - q_0 - i_1$ and approximating the sum over i' gives

$$(2v)^{st} N \leqq \sum_{0 \leqq q_0 \leqq q_1 < q_*} 2^{q_1} \frac{(v-q_0-1)!}{(v-q_1-1)!}\binom{t}{q_1-q_0}[q_1^s - (q_1-1)^s]^{q_1-q_0}$$
$$\cdot\, [P_u + (q_1-q_0)[q_1^s - (q_1-1)^s]]^{t-q_1+q_0}\left(1 + O\left(\frac{q_1^s t}{v^s}\right)\right) \tag{10}$$
$$+ v \sum_{q_* \leqq q_0 < v} 2^{q_0}\binom{v-1}{q_0} P_0^t,$$

provided $q_1^s t/v^s$ decreases as v increases. For $q_* = v^{(s-\alpha)/s-\varepsilon}$, $\varepsilon > 0$ and small, and $\alpha > s/2$, the approximation is valid. Also under these conditions the second summation will be small enough to be absorbed in the final error term. The first sum is maximized with

$$q_0 = 2\left(1 - \frac{\ln 2}{s-1}\right)\left(\frac{4 \ln 2}{s(s-1)}\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)} + \Theta(1),$$

$$q_1 = 2\left(\frac{4 \ln 2}{s(s-1)}\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)} + \Theta(1).$$

This gives

$$N \leqq \exp\left[\frac{2(s-2)\ln 2}{s-1}\left(\frac{4 \ln 2}{s(s-1)}\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)} + \Theta(\ln v)\right]. \tag{11}$$

for $s/2 < \alpha < s-1$. Unfortunately, we do not have a good approximation for $1 < \alpha \leqq s/2$.

6. Conclusion. For $s/2 < \alpha < s-1$ and $s \geqq 3$ we have shown that

$$N = \exp\left[\Theta(v^{(s-\alpha-1)/(s-2)})\right],$$

while for $1 < \alpha \leqq s/2$ and $s \geqq 3$ we have

$$\exp\left[\Theta(v^{(s-\alpha-1)/(s-2)})\right] \leqq N \leqq \exp\left[\Theta\{(v^{(s-\alpha-1)/(s-2)})(\ln v)^{(s-1)/(s-2)}\}\right].$$

The coefficient of the leading term of the two-sum upper limit is exactly $2^{1/(s-2)}$ times the coefficient of the leading term of the one-sum lower limit, so the upper and lower bounds on the coefficient are close for large s. Table 5 shows the value of the leading coefficients for formulas (6), (8) and (11) for small values of s. The two-sum lower limit gives only a small improvement over the one-sum lower limit.

The time for simple backtracking on these problems [2] is

$$N = \exp\left[\Theta(v^{(s-\alpha)/(s-1)})\right],$$

so the speed-up from using simple search rearrangement is comparable to that obtained by switching to simpler problems with s reduced by one. (For $s/2 < \alpha < s$ the leading coefficient is also somewhat smaller for the search rearrangement algorithm.) Since this change is in the exponent the improvement in expected running time obtained by using simple search rearrangement is dramatic.

The values of q_0 and q_1 at the maximum in the two-sum approximations suggest that in search rearrangement backtracking there is typically a large number of unary nodes preceding each binary node. This conclusion is confirmed by our experimental measurements.

An interesting question for future research is whether there are backtracking algorithms (such as perhaps the ones reported in [6]) that use average time $\exp\left[\Theta(v^{(s-\alpha-k+1)/(s-k)})\right]$ for each fixed $k \leqq s-1$. Experimental results show that the algorithms in [6] perform better than simple search rearrangement on very large problems, but they have not been analyzed as yet.

TABLE 1
A summary of the analysis for the exact formula.

Required terms: $j_{ix} \geqq 1$ for $1 \leqq i \leqq m$, $1 \leqq x \leqq q_i - q_{i-1}$

Size of set	Reason
$R_{i-1} = Q_{i-1} - Q_{i-2}$, where $Q_i = (q_i+1)^s - q_i^s$, $Q_{-1} = 0$	Term must have false literals combined with literals for forced variables. The forced variable cannot be forced on earlier round.

Permitted terms: $t - \sum_x j_{ix}$

Size of set	Reason
$(2v)^s$	Total number of terms, minus:
$-q_m^s$	terms made of false literals,
$-(v-q_m)Q_m$	terms that force variables to true, and
$-\sum_{0 \leqq i \leqq m} (v-q_i)R_i$	terms that force variables to false.

TABLE 2
A summary of the analysis for the one-sum upper limit formula.

Required terms: None

Permitted terms: t

Size of set	Reason
$(2v)^s$	Total number of terms, minus:
$-q_0^s$	terms made of false literals, and
$-2(v-q_0)Q_0$	terms that force unforced variables.

TABLE 3
A summary of the analysis for the two-sum upper limit formula.

Required terms: $j_{1x} \geqq 1$ for $1 \leqq x \leqq q_1 - q_0$

Size of set	Reason
$q_1^s - (q_1-1)^s$	Each forced variable must occur in a term with all other literals false.

Permitted terms: $t - \sum_x j_{1x}$

Size of set	Reason
$(2v)^s$	Total number of terms, minus:
$-q_1^s$	terms made of false literals,
$-2(v-q_1)Q_1$	terms that force unforced variables, and
$-(q_1^s - q_0^s)$	total number of required terms.

TABLE 4
A summary of an alternate two-sum upper limit.

Required terms: $j_x = 1$ for $q_0 < x \leqq q_1$

Size of set	Reason
$x^s - (x-1)^s$	Each forced variable must occur in a term with all other literals false, where all other variables are already set.

Permitted terms: Same as in Table 3

TABLE 5
Coefficients c for $N \sim \exp cv^{(s-\alpha-1)/(s-2)}$.

s	one-sum lower	two-sum lower	two-sum upper
3	0.1602	0.2465	0.3203
4	0.3141	0.3922	0.4442
5	0.4271	0.5226	0.5381
6	0.5142	0.5760	0.6115
7	0.5840	0.6398	0.6708
8	0.6415	0.6924	0.7200
9	0.6899	0.7367	0.7617
10	0.7314	0.7748	0.7976
100	1.2534	1.2595	1.2624
1000	1.3663	1.3670	1.3672
∞	$2 \ln 2 \approx 1.3863$	$2 \ln 2 \approx 1.3863$	$2 \ln 2 \approx 1.3863$

Appendix 1. A number of numerical checks were performed to verify the results in this paper.

Equation (1) was verified by a computer program that performed direct enumeration for $s = 1$, $1 \leqq t \leqq 4$, $1 \leqq v \leqq 8$; $s = 2$, $1 \leqq t \leqq 2$, $1 \leqq v \leqq 8$; $s = 2$, $t = 3$, $1 \leqq v \leqq 6$; $s = 3$, $t = 1$, $1 \leqq v \leqq 8$; and $s = 3$, $t = 2$, $1 \leqq v \leqq 4$. A statistical check with one percent accuracy was done for $s = 3$, $t = v^{3/2}$, $v = 4$ and 9.

Formula (6) was compared with the maximum term in the one-sum lower limit. At $v = (1337)^2$, $s = 3$, $t = (1337)^3$, the ratio of the logarithm of Formula (6) to the logarithm of the maximum term was 1.0013, and converging.

Formula (7) was compared with the one-sum upper limit (Formula (3)). At $v = (34)^2$, $s = 3$, $t = (34)^3$ the ratio of the logarithms achieved a minimum of 0.988. For higher v it slowly increased.

Formula (8) was compared with the maximum term in the two-sum lower limit (Formula (3)). At $v=64$, $s=3$, $t=512$ the ratio of the logarithms was 1.5 and decreasing. Roundoff error prevented measurements at significantly larger v.

Formula (11) was compared with the two-sum upper limit (Formula (5)). At $v=(11)^2$, $s=3$, $t=(11)^3$ the ratio was 0.805 and erratically increasing.

Appendix 2. In this appendix we give more details of the derivation of the asymptotic results mentioned in § 5. The one-sum lower limit is

$$\sum_{0\leqq q_0<v} 2^{q_0} p_0^t = \sum_{0\leqq q_0<v} 2^{q_0}((2v)^s - q_0^s - 2(v-q_0)((q_0+1)^s - q_0^s))^t.$$

The logarithm of a term is

$$q_0 \ln 2 + t \ln [(2v)^s - q_0^s - 2(v-q_0)((q_0+1)^s - q_0^s)].$$

Set $t=v^\alpha$, take the derivative with respect to q_0, and set it equal to 0.

Assuming $q_0 \ll v$, the largest terms of the derivative give

$$2s(s-1)v^{\alpha+1}q_0^{s-2} = (2v)^s \ln 2,$$

or

$$q_0 = \left(\frac{2^{s-1}\ln 2}{s(s-1)}\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)}.$$

Substitute this value back into the original derivative; keeping the largest terms gives

$$q_0 = \left(\frac{2^{s-1}\ln 2}{s(s-1)}\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)}(1+\Theta(v^{-(\alpha-1)/(s-2)})+\Theta(v^{-(s-\alpha-1)/(s-2)})).$$

Substitute this value into the original formula to get the logarithm of the maximum term; the most significant terms give

$$v^\alpha \ln (2v)^s + \left(\frac{2\ln 2}{s(s-1)}\right)^{1/(s-2)} (2\ln 2)\left(\frac{s-1}{s-2}\right)^{(s-\alpha-1)/(s-2)} + \Theta v^{(s-2\alpha)/(s-2)} + \Theta(1).$$

Since all the terms are positive, the value of one term is a lower limit on the value of the sum.

Formula (4) gives the one-sum limit. Using Stirling's approximation for the binomial, the log of a term is

$$\ln v - \tfrac{1}{2}\ln(2\pi) + (v-\tfrac{1}{2})\ln(v-1) - (q_0+\tfrac{1}{2})\ln q_0 - (v-q_0-\tfrac{1}{2})\ln(v-q_0-1)$$

$$+ q_0 \ln 2 + v^\alpha \ln((2v)^s - q_0^s - 2(v-q_0)((q_0+1)^s - q_0^s)) + O\left(\frac{1}{q_0}\right) + O\left(\frac{1}{v-q_0}\right).$$

Take the derivative with respect to q_0 and set it equal to zero. Assuming $q_0 \ll v$, and keeping the asymptotically most significant terms, shows that q_0 near the peak satisfies the relation.

$$q_0^{s-2} \sim \frac{2^{s-1}}{s(s-1)} v^{s-\alpha-1} \ln\left(\frac{2v}{q_0}\right).$$

To obtain an approximation of q_0 at the peak, let

$$a(v) = \frac{2}{s(s-1)(s-2)} v^{1-\alpha}$$

and let $F(v)$ be the solution to the equation

$$\frac{1}{F(v)} = a(v) \ln F(v).$$

Then

$$q_0 \sim 2va(v)^{1/(s-2)} (\ln F(v))^{1/(s-2)};$$

attention to the O terms gives the value of q_0 given in the text preceding Formula (7). The local maximum at $q_0 = v-1$ is even less significant for this limit than it was for the one-sum lower limit. The value of the upper limit is no more than v times the value of the maximum term, and this factor is absorbed in the $\Theta(\ln v)$ term.

The two-sum lower limit is given in Formula (3). It can be rewritten in the following form by expanding the multinomial:

$$\sum_{0 \leqq q_0 \leqq q_1 < v} \sum_i 2^{q_1} \binom{v-q_0-1}{q_1-q_0} \binom{q_1-q_0}{i} (-1)^{q_1-q_0+i} (P_1 + iQ_0)^t.$$

Using the transformation

$$\sum_i \binom{a}{i} (-1)^i (1+ic)^t = \sum_j (-1)^a \binom{t}{j} \left\{ {j \atop a} \right\} a!\, c^j,$$

with $a = q_1 - q_0$, $c = Q_0/P_1$, gives, for the sum over i,

$$P_1^t (-1)^{q_1-q_0} \sum_i \binom{q_1-q_0}{i} (-1)^i \left(1 + i\frac{Q_0}{P_1}\right)^t = \sum_j \binom{t}{j} \left\{ {j \atop q_1-q_0} \right\} (q_1-q_0)!\, Q_0^j P_1^{t-j}.$$

A lower limit for this sum is $\binom{t}{q_1-q_0}(q_1-q_0)!\, Q_0^{q_1-q_0} P^{t-q_1+q_0}$. We thus obtain the lower limit

$$\sum_{0 \leqq q_0 \leqq q_1 < v} 2^{q_1} \binom{v-q_0-1}{q_1-q_0} \binom{t}{q_1-q_0} (q_1-q_0)!\, Q_0^{q_1-q_0} P_1^{t-q_1+q_0}.$$

Using Stirling's approximation, the log of a term is

$$\begin{aligned}
&q_1 \ln 2 (v-q_0-\tfrac{1}{2}) \ln (v-q_0-1) + (v^\alpha + \tfrac{1}{2}) \ln v^\alpha - \tfrac{1}{2} \ln 2\pi - (q_1-q_0+\tfrac{1}{2}) \ln (q_1-q_0) \\
&\quad - (v-q_1-\tfrac{1}{2}) \ln (v-q_1-1) - (v^\alpha - q_1 + q_0 + \tfrac{1}{2}) \ln (v^\alpha - q_1 + q_0) \\
&\quad + (q_1-q_0) \ln ((q_0+1)^s - q_0^s) \\
&\quad + (v^\alpha - q_1 + q_0) \ln [(2v)^s - q_1^s - 2(v-q_1)((q_1+1)^s - q_1^s) - (q_1-q_0)((q_0+1)^s - q_0^s)] \\
&\quad + \Theta\left(\frac{1}{v-q_0-1}\right) + \Theta\left(\frac{1}{v^\alpha}\right) + \Theta\left(\frac{1}{q_1-q_0}\right) + \Theta\left(\frac{1}{v-q_1-1}\right) + \Theta\left(\frac{1}{v^\alpha - q_1 + q_0}\right).
\end{aligned}$$

Take the derivative with respect to q_1, set it equal to 0, and solve, retaining asymptotically important terms. This gives

$$\ln \left(\frac{s q_0^{s-1}}{2^{s-1} v^{s-\alpha-1} (q_1-q_0)} \right) = \frac{s(s-1) q_1^{s-2}}{v^{s-\alpha-1} 2^{s-1}}.$$

The same procedure on the q_0 derivative gives

$$\begin{aligned}
0 = &\ln \left(\frac{2^{s-1} v^{s-\alpha-1} (q_1-q_0)}{s q_0^{s-1}} \right) + \ln 2 + (s-1)\left(\frac{q_1}{q_0} - 1\right) \\
&- \frac{v^{\alpha-s} s}{2^s} ((s-1) q_1 q_0^{s-2} - q_0^{s-1}).
\end{aligned}$$

Use the value of the ln term from the q_1 derivative to replace the term in the q_0 derivative. Identify the asymptotically important terms using

$$q_0 \sim v^{(s-\alpha-1)/(s-2)} \quad \text{and} \quad q_1 \sim v^{(s-\alpha-1)/(s-2)}.$$

This gives

$$q_0 \sim \frac{q_1}{sq_1^{s-2}/(v^{s-\alpha-1}2^{s-1}) - \ln 2/(s-1) + 1}.$$

Substitute this value of q_0 into the formula for the q_1 derivative. Changing the variable to

$$x = \frac{sq_1^{s-2}}{2^{s-1}v^{s-\alpha-1}}$$

gives the equation

$$(s-1)x = \ln \frac{x}{(x+1-\ln 2/(s-1))^{s-2}(x-\ln 2/(s-1))}.$$

This equation can be solved for x in terms of s. Setting $y = (s-1)x$ gives the results in the text. We have not calculated the errors in the values for q_0 and q_1 that maximize the value of a term; this is not necessary since the value of any term in a sum of positive terms is a lower limit on the value of the sum.

The two-sum upper limit is given in Formula (5). Because of limitations of our asymptotic methods, we derive an upper bound by using terms from Formula (5) over a part of the range and from Formula (4) on the remainder, as indicated in Formula (9). This is possible because each individual term in Formula (4) for a fixed q_0 bounds the partial sum in Formula (1) with q_m fixed to that same value. Likewise, a partial sum in Formula (5) with q_1 fixed to some value bounds the partial sum in Formula (1) with q_m fixed to that value.

To remove the sum over i_1 in Formula (9), use the transformation

$$(-1)^a \sum_i \binom{a}{i}(-1)^i(1+ic)^t = \binom{t}{a} a!\, c^a (1+\Theta(act)),$$

with $a = q_1 - q_0$ and $c = (q_1^s - (q_1-1)^s)/P_2$. This transformation is valid for $\alpha > s/2$ and $q_1 < v^{(s-\alpha)/s-\varepsilon}$, where $\varepsilon > 0$.

To find the values of q_1 and q_0 that maximize the first term in the resulting formula (Formula (10)), analyze it as follows. Using Stirling's approximation, the log of a term is

$$\begin{aligned}
&q_1 \ln 2 - \tfrac{1}{2}\ln 2\pi + (v - q_0 + \tfrac{1}{2})\ln(v - q_0 - 1) + (v^\alpha + \tfrac{1}{2})\ln v^\alpha - (v - q_1 - \tfrac{1}{2})\ln(v - q_1 - 1) \\
&\quad - (q_1 - q_0 - \tfrac{1}{2})\ln(q_1 - q_0) - (v^\alpha - q_1 + q_0 - \tfrac{1}{2})\ln(v^\alpha - q_1 + q_0) \\
&\quad + (q_1 - q_0)\ln(q_1^s - (q_1-1)^s) + (v^\alpha - q_1 + q_0)\ln[P_2 + (q_1 - q_0)(q_1^s - (q_1-1)^s)] \\
&\quad + \Theta\left(\frac{1}{v - q_0 - 1}\right) + \Theta\left(\frac{1}{v^\alpha}\right) + \Theta\left(\frac{1}{v - q_1 - 1}\right) + \Theta\left(\frac{1}{q_1 - q_0}\right) + \Theta\left(\frac{1}{v^\alpha - q_1 + q_0}\right) \\
&\quad + \Theta\left(\frac{(q_1 - q_0)v^\alpha(q_1^s - (q_1-1)^s)}{P_2 + q_1 - q_0(q_1^s - (q_1-1)^s)}\right).
\end{aligned}$$

Take the derivative with respect to q_1 and set it equal to zero; the important terms give

$$0 = \ln\left(\frac{sq_1^{s-1}}{(q_1-q_0)2^{s-1}v^{s-\alpha-1}}\right) + \frac{(q_1-q_0)(s-1)}{q_1} - \frac{s(s-1)q_1^{s-2}}{2^{s-1}v^{s-\alpha-1}} + O(v^{(1-\alpha)/(s-2)}) + O(v^{-(s-\alpha-1)/(s-2)}).$$

The important terms from the q_0 derivative give

$$0 = \ln\left(\frac{(q_1-q_0)2^s v^{s-\alpha-1}}{sq_1^{s-1}}\right) + O(v^{(1-\alpha)/(s-2)}) + O(v^{-(s-\alpha-1)/(s-2)}).$$

This implies the log term is of the same order as the O terms. Use this to rewrite the q_1 equation and solve it for q_0 in terms of q_1, giving

$$q_0 = \left(1+\frac{\ln 2}{s-1}\right)q_1 - \frac{sq_1^{s-1}}{2^{s-1}v^{s-\alpha-1}} + O(v^{(s-2\alpha)/(s-2)}) + O(1).$$

Since our approximation is only good for $s-2\alpha<0$, the first O term can be dropped.

Substitute this value into the equation from the q_0 derivative, exponentiate each side of the resulting equation, and expand the right side in a power series. This gives

$$\frac{(-(\ln 2/(s-1)) + sq_1^{s-2}/(2^{s-1}v^{s-\alpha-1}) + O(v^{-(s-\alpha-1)/(s-2)}))2^s v^{s-\alpha-1}}{sq_1^{s-2}} = 1 + O(v^{(1-\alpha)/(s-1)}) + O(v^{-(s-\alpha-1)/(s-2)}).$$

Letting

$$x = \frac{sq_1^{s-2}}{2^{s-1}v^{s-\alpha-1}},$$

we obtain

$$x = \frac{2\ln 2}{s-1} + O(v^{-(s-\alpha-1)/(s-2)}),$$

or

$$q_1 = \left(\frac{2^s \ln 2}{s(s-1)}\right)^{1/(s-2)} v^{(s-\alpha-1)/(s-2)} + O(1),$$

and

$$q_0 = \left(1-\frac{\ln 2}{s-1}\right)q_1 + O(1).$$

By substituting these values into Formula (10) and ignoring the second sum we obtain Formula (11). That sum is small relative to the terms we retain. The maximum term in the sum is obtained by setting $q_0 = q_*$. The sum is no bigger than v times its maximum term. The log of the sum is

$$2s^{-1/s}\left(\frac{\alpha \ln v}{s} + 1 + \ln 2\right)v^{(s-\alpha)/s} + \dot{O}(v^{(\alpha-s)/s}).$$

For $\alpha > s/2$, this can be neglected in relation to the terms in Formula (11) (since, for example, $\exp(v^2+O(1)) + \exp(v) = \exp(v^2+O(1))$ asymptotically).

Acknowledgment. We wish to thank our referee, Prof. James Fill, for his careful reading of this paper and his numerous helpful comments.

[1] J. R. BITNER AND E. M. REINGOLD, *Backtrack programming techniques*, Comm. ACM, 18 (1975), pp. 651–655.

[2] CYNTHIA A. BROWN AND PAUL W. PURDOM, JR., *An average time analysis of backtracking*, this Journal, 10 (1981), pp. 583–593.

[3] ———, *An empirical comparison of backtracking algorithms*, IEEE Trans. Patt. Match. and Machine Intell., 4 (1982), pp. 309–316.

[4] D. E. KNUTH, *Estimating the efficiency of backtracking programs*, Math. Comput., 29 (1975), pp. 121–136.

[5] ———, *The Art of Computer Programming, Vol.* 1, Addison-Wesley, Reading, MA, 1975, p. 65.

[6] PAUL W. PURDOM, JR., CYNTHIA A. BROWN AND EDWARD L. ROBERTSON, *Backtracking with multi-level serarch rearrangement*, Acta Informat., 15 (1981), pp. 99–113.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0010 $01.25/0

CONVEX CLUSTERS IN A DISCRETE m-DIMENSIONAL SPACE*

JOHN L. PFALTZ†

Abstract. A simple procedure is used to dynamically create convex clusters of items in a discrete m-dimensional space. Systems of difference equations are derived that describe the behavior of this cluster formation under the assumption that individual clusters are either of bounded or unbounded size. These equations are used to calculate the expected number and size of such clusters given the number n of items.

For an actual implemented database access and retrieval method, these results provide a way of determining both the expected storage overhead and the expected retrieval costs.

Key words. attribute space, cluster, database, dynamic item entry, retrieval cost, file organization, indexed-descriptor, partial-match retrieval, storage overhead

1. Introduction.

In this paper we will consider the creation of convex clusters formed by the random placement of items in a discrete m-dimensional space with finite bounds. That is, items will be associated with points of the space that may be denoted by m-tuples of integers $(i_1, i_2, \cdots, i_m)$, where $1 \leqq i_i \leqq w_i$. Here w_i denotes the upper bound on that coordinate of the space. Two questions are of particular interest. Given a clustering process and n items in the space:

a. What is the expected number of convex clusters?

b. What is the expected number of convex clusters containing precisely k items?

Although we have deliberately cast these problems in purely mathematical terms, for instance "convex cluster" and "m-dimensional space", they have been motivated by a very practical application, that of determining the expected retrieval cost and expected storage overhead of an efficient information retrieval organization for very large data files [8]. In this interpretation, cells of the m-dimensional space may be viewed as buckets in an m-dimensional attribute space; items may be regarded as data records; and clusters may be blocks in a data file. We will discuss these computer applications more fully in §§ 5 and 8.

2. Descriptors and convex clusters.

Since every cell[1] of the space can be identified by an m-tuple $(i_1, i_2, \cdots, i_m)$ of integers, we can equivalently identify the cell by a *descriptor* consisting of m distinct bit strings called *fields*. In the jth field (of width w_i bits) only the ith bit is set to 1. Thus

00100000 000001 0000100000 00000010

is a 4-field descriptor for the cell (3, 6, 5, 7) in a finite 4-dimensional space consisting of $8 \times 6 \times 10 \times 8 = 3840$ cells. The use of bit descriptors instead of integer m-tuples will at first seem awkward. But, in fact, this transformation simplifies the following analysis.

Every cell in the space is, by itself, a convex subset. Although other definitions of "convexity" in a discrete space are possible, we will adopt the convention that a subset is *convex* if and only if it is a $b_1 \times b_2 \times \cdots \times b_m$-cube. Thus in the 2-dimensional 15×15 space shown in Fig. 1(a) (which is deliberately "small" for illustrative purposes;

* Received by the editors March 6, 1981, and in final revised form August 9, 1982. This research was supported in part by the National Science Foundation under grant MCS80-17779.

† Department of Applied Math. and Computer Science, University of Virginia, Charlottesville, Virginia 22903.

[1] We could equally well speak of "points" in the space. However, the term "cell" seems preferable because we will be associating data items with such points in the space, and "cell" suggests that several such items can be associated with (or stored in) that "cell".

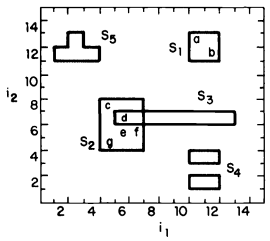


FIG. 1(a)

```
D1  0000000000011000 0000000000001100
D2  0000111000000000 0000111100000000
D3  0000011111111100 0000001000000000
D4  0000000000011000 0101000000000000
```

FIG. 1(b)

```
Da  0000000000010000 0000000000001100
Db  0000000000001000 0000000000001000
    0000000000011000 0000000000001100  = Da ∨ Db = D1

Dc  0000100000000000 0000000100000000
Dd  0000010000000000 0000000100000000
De  0000010000000000 0000000100000000
Df  0000001000000000 0000000100000000
Dg  0000100000000000 0000100000000000
    0000111000000000 0000111100000000  = Dc ∨ Dd ∨ De ∨ Df ∨ Dg
                                       = D2
```

FIG. 1(c)

normally $m > 2$) subsets S_1, S_2 and S_3 are convex subsets, while S_4 and S_5 are not. For convex subsets one can form a *descriptor* by simply OR-ing the descriptors of each of its cells. The descriptors of these convex $b_1 \times b_2 \times \cdots \times b_m$-cubes are characterized by the fact that each field has b_i *consecutive* 1-bits set. Such convex subsets are called *clusters*. Their associated descriptors are *convex cluster descriptors*, or just cluster descriptors. D_1, D_2 and D_3 in Fig. 1(b) are convex descriptors associated with the convex subsets S_1, S_2 and S_3 respectively. D_4 is the descriptor associated with S_4; but neither is convex. Note that there is no adequate descriptor for the set S_5. By the *extent* of a cluster, we mean the number of cells in the space comprising it; that is extent $= b_1 \times b_2 \times \cdots \times b_m$. (One can use "volume" as a synonym for extent.) In Fig. 1, extent $(S_2) = 4$, extent $(S_2) = 12$, and extent $(S_3) = 8$.

Let S_1 and S_2 denote any convex subsets of a space, so that their corresponding descriptors D_1 and D_2 are also convex. Clearly $S_1 \cap S_2$ is convex (or void) and $D_1 \wedge D_2$, if it exists, must be the convex descriptor of that intersection set. (To "exist", a descriptor must have at least one bit set in each field.) In general, $S_1 \cup S_2$ will not be convex, and similarly $D_1 \vee D_2$ need not be a convex descriptor. Moreover $D_1 \vee D_2$ will not, in general, denote $S_1 \cup S_2$. If $D_1 \vee D_2$ is convex, it denotes the *convex hull* of $S_1 \cup S_2$.

In Fig. 1, S_1 is the convex hull of the cells a:(11, 13) and b:(12, 12). S_2 is the convex hull of the points c:(5, 8), d:(6, 7), e:(6, 6), f:(7, 6) and g:(5, 5). See Fig. 1(c). These are convex clusters. By the *content* of a cluster, we will mean the number of items in it. For example, regarding S_2 as a cluster of the items c, d, e, f and g implies that content $(S_2) = 5$.

We will assume the following *clustering algorithm.* Let a new item I be entered into the space at a cell with descriptor D_I. Let D_C be the descriptor of a cluster C.

Condition 1. The item I will be added to C only if $D_I \vee D_C$ is convex. $D_I \vee D_C$ will be the new cluster descriptor.

Condition 2. If there are two, or more, clusters C_1 and C_2 such that $D_I \vee D_{C_1}$ and $D_I \vee D_{C_2}$ are both convex, then I will be added to a cluster of minimal content, i.e., containing the fewest items. This need not be a cluster of minimal extent, i.e., the smallest $b_1 \times b_2 \times \cdots \times b_m$-cube. Note that, whenever an item is added to an existing cluster C, its content is increased by one, its extent may, or may not, increase. Two items may be entered into the same cell of the space, but they need not belong to the same cluster. Condition 2 will not be used until we begin the analysis of § 6.

Finally note that with this algorithm, which dynamically creates clusters as items are entered and does not redefine cluster boundaries after the fact, the nature of the formed clusters is dependent on the order of entry. For example, the items (1, 1), (2, 2), (3, 3), (4, 4) entered in this order form a single cluster with descriptor (11110 $\cdots$ 11110 $\cdots$); but entered in the order (1, 1), (4, 4), (2, 2), (3, 3) form two distinct clusters with descriptors (11000 $\cdots$ 11000 $\cdots$) and (00110 $\cdots$ 00110 $\cdots$).

3. Halos and intersections. In this section, we calculate two values that will be used in the probabilistic arguments of succeeding sections. Let C be any $b_1 \times b_2 \times \cdots \times b_m$ cluster. If a new item is associated with a cell that is contained within, or is adjacent to, C then that item *may* be added to C. The set of cells adjacent to C we call its *halo.* Combined with its halo, the *effective extent* of C is $(b_1+\varepsilon_1) \times (b_2+\varepsilon_2) \times \cdots \times (b_m+\varepsilon_m)$, where $1 \leqq \varepsilon_j \leqq 2$, $1 \leqq j \leqq m$.

For field$_j$, ε_j counts the number of adjacent 0 bits in the descriptor, so for any particular cluster C, $\varepsilon_j = 1$ or 2, depending on whether C is centrally placed in the space with respect to attribute-j. (Note that we will always assume that $b_j \leqq w_j - 1$, since otherwise all bits in field$_j$ are set. Consequently we may assume $\varepsilon_j \neq 0$.) Now we need an expression for $\bar{\varepsilon}_j$, the *expected extent* of a halo, in terms of b_j, the extent of C.

Consider field$_j$ in D_C for any attribute j. If the string of b_j consecutive bits is in the "interior" of the field, there are adjacent unset bit positions at either end, and $\varepsilon_j = 2$. If the string is in either "extreme" position there is only one adjacent unset bit position and $\varepsilon_j = 1$. We may thus let

$$\begin{aligned} \bar{\varepsilon}_j &= 1 \cdot \text{prob (extreme config.)} + 2 \cdot \text{prob (interior config.)} \\ &= 1 \cdot (b_j+1)/w_j + 2 \cdot (w_j - b_j - 1)/w_j \\ &= (2w_j - b_j - 1)/w_j = 2 - (b_j+1)/w_j, \end{aligned} \tag{3.1}$$

where the probabilities of an "extreme" or "interior" configuration can be calculated by examining the generation sequences of the possible configurations. We will assume that (3.1) is valid even when using $\bar{b}_j$, the *expected number* of bits set in field j. Clearly $\bar{\varepsilon}_j$ is a function of $\bar{b}_j$, with $\bar{\varepsilon}_j \to 2$, when $\bar{b}_j \to 1$ and $\bar{\varepsilon}_j \to 1$ when $\bar{b}_j \to w_j - 1$.

Let C_1 and C_2, be two clusters, both of extent $b_1 \times b_2 \times \cdots \times b_m$. We will determine their expected *effective* intersection, that is, the expected number of cells in the intersection of C_1 and C_2 *together with their halos.* (Note that we have considerably simplified the more general problem of expected intersection of clusters by assuming that C_1 and C_2 are of identical extent. But it will suffice for our purposes.)

Again consider field$_j$ in D_{C_1} and D_{C_2}, for any attribute j, $1 \leqq j \leqq m$. Let s_1 and s_2 denote the strings of b_j consecutive 1-bits in each descriptor field respectively. Assume that s_1 is positioned anywhere in field$_j$ and that s_2 is displaced exactly d positions *to the left* of s_1, $0 \leqq d \leqq w_j - b_j$.

For various values of d one can show that the amount of effective overlap (of the descriptors and their halos) of s_1 and s_2 in field$_j$ is

$$\text{overlap } (d)\begin{cases} = b_j + \bar{\varepsilon}_j & \text{if } d = 0, \\ = b_j - d + 2 & \text{if } 1 \leqq d \leqq b_j + 1, \\ = 0 & \text{if } d > b_j + 1. \end{cases}$$

The probability of obtaining a displacement of $d \geqq 0$ bits is the probability of locating s_1 in any position with at least d positions to its left times the probability of locating s_2 in that position which is precisely offset by d bits. Thus

$$\text{prob (offset} = d) = \frac{(w_j - d - b_j + 1)}{(w_j - b_j + 1)^2}$$

hence the expected overlap in field$_j$ expressed in terms of the possible displacements d of s_2 relative to s_1 is

$$\begin{aligned} \text{exp (overlap)} &= \text{overlap } (0) \cdot \text{prob } (0) + \sum_{d=1}^{b_j+1} \text{overlap } (d) \cdot \text{prob } (d) \\ &= (b_j + \varepsilon_j) + \sum_{d=1}^{b_j+1} \left[\frac{(b_j - d + 2)(w_j - d - b_j + 1)}{(w_j - b_j + 1)^2} \right] \end{aligned} \tag{3.2}$$

(which replacing the last term with the sum of the finite series becomes)

$$= b_j + \frac{2w_j - b_j - 1}{w_j} + \frac{(b_j + 1)(b_j + 2)(3w_j - 4b_j)}{3 \cdot (w_j - b_j + 1)^2}.$$

Although we have derived (3.2) under the assumption that b_j is integral, we again assume that the expression holds for real values, $\bar{b}_j$. (One of the most convincing ways of verifying (3.1) and (3.2) is to work out and exhaustively count all possible configurations for small values of b_j and w_j, possibly for $b_j = 2$, $w_j = 7$.)

Let $\bar{\gamma}(n)$ denote the expected number of convex clusters given n items in the space. Assume that n items, chosen independently from a uniform distribution on the space, have been entered into the space, and that an $(n+1)$-st item I is entered. According to the clustering algorithm of the preceding section, item I will be added to an existing cluster if $D_I \vee D_C$ is convex for some cluster C. Thus item I will create a new singleton cluster if and only if $D_I \vee D_C$ is not convex for all C, and so one has the difference equation

$$\bar{\gamma}(n+1) = \bar{\gamma}(n) + \Delta\gamma(n), \tag{3.3}$$

where $\Delta\gamma(n) = \text{prob } (\forall C, D_I \vee D_C \text{ is not convex})$, and $\bar{\gamma}(1) = 1$.

Since $D_I \vee D_C$ is convex if and only if each field j is convex

$$\text{prob } (D_I \vee D_C \text{ is convex}) = \prod_{j=1}^{m} \frac{(\bar{b}_j + \bar{\varepsilon}_j)}{w_j} \tag{3.4}$$

so that

$$\begin{aligned} \rho(n) &= \text{prob } (D_I \vee D_C \text{ is not convex}) \\ &= 1.0 - \prod_{j=1}^{m} \frac{(\bar{b}_j + \bar{\varepsilon}_j)}{w_j} \end{aligned} \tag{3.5}$$

and thus

$$\begin{aligned}\Delta\gamma(n) &= \text{prob}\,(\forall C, D_I \vee D_C \text{ is not convex}) \\ &= (\text{prob}\,(D_I \vee D_C \text{ is not convex}))^{\bar{\gamma}(n)} \\ &= \rho(n)^{\bar{\gamma}(n)} = \left(1.0 - \prod_{j=1}^{m} \frac{(\bar{b}_j + \bar{\varepsilon}_j)}{w_j}\right)^{\bar{\gamma}(n)}.\end{aligned} \tag{3.6}$$

Note that expressions (3.4), (3.5) and (3.6) are valid only if the distributions of bits set in distinct fields are uniform and independent. Initially we can assume this uniformity. But in § 6 we will have to recalculate (3.5) under a condition in which the clusters are no longer uniformly distributed throughout the space. Independence will be assumed throughout this paper, even though it is well known that in any particular application the various attributes of the items represented in the space may exhibit considerable dependency. Note also, that $\bar{\gamma}(n)$ and $\rho(n)$ are functions of n, the number of items entered into the space as we have clearly indicated. So also are $\bar{b}_j$ and $\bar{\varepsilon}_j$; but to keep the expressions to a reasonable size we have not always used either $\bar{b}_j(n)$ or $\bar{\varepsilon}_j(n)$.

The difference equation (3.3) can now be simply expressed as

$$\bar{\gamma}(n+1) = \bar{\gamma}(n) + \rho(n)^{\bar{\gamma}(n)}. \tag{3.7}$$

4. Expected number of convex clusters. If we have an expression for $\bar{b}_j$ as a function of n, then we can calculate $\bar{\varepsilon}_j$ using (3.1), $\rho(n)$ using (3.5), and $\bar{\gamma}(n)$, the expected number of convex clusters, using (3.7). We will set up and solve a difference equation of the form

$$\bar{b}_j(n+1) = \bar{b}_j(n) + \Delta\bar{b}_j(n). \tag{4.1}$$

However, we will have to obtain $\Delta\bar{b}_j$ in a somewhat indirect manner.

Let $\bar{B}_j$ denote the expected *total* number of 1-bits set in field j of *all* cluster descriptors D_C. Then

$$\bar{b}_j(n) = \bar{B}_j(n)/\gamma(n) \quad \text{and} \quad \Delta\bar{b}_j(n) = (\gamma(n)\cdot\Delta\bar{B}_j(n) - \bar{B}_j(n)\cdot\Delta\gamma(n))/\gamma(n)^2. \tag{4.2}$$

We may approximate $\gamma(n)$ by $\bar{\gamma}(n)$ and using (3.6), $\Delta\gamma(n)$ by $\rho(n)^{\bar{\gamma}(n)}$. We thus need only an expression for $\Delta\bar{B}_j$.

Again we look at the incremental change in B_j resulting from the entry of a new item, I. If for all clusters C, D_I, D_C is not convex, then I will start a new cluster, and B_j will be increased by 1. If for some C, D_I, D_C is convex, then a new bit may, or may not, be set in the resulting descriptor field depending on the position of the bit in the field. The probability of adding an extra bit and incrementing B_j is

$$\begin{aligned}\text{prob}\,(\text{incrementing } B_j \,|\, D_I \vee D_C \text{ is convex}) &= \varepsilon_j/(b_j+\varepsilon_j) \\ &= (2w_j - b_j - 1)/[(b_j+2)w_j - b_j - 1].\end{aligned}$$

(Note that evaluating this expression for $b_j = w_j - 1$ and $b_j = 1$ provides the bounds $1/w_j \leqq \text{prob} < \frac{2}{3}$.) Thus the incremental change to $\bar{B}_j$ as the result of entering a new item is given by

$$\begin{aligned}\Delta\bar{B}_j(n) = 1 \cdot &\text{prob}\,(VC, D_I \vee D_C \text{ is not convex}) \\ &+ \text{prob}\,(\text{incrementing } B_j)\cdot\text{prob}\,(\exists C, D_I \vee D_C \text{ is convex})\end{aligned} \tag{4.3}$$

or

$$\Delta\bar{B}_j(n) = \rho^{\bar{\gamma}} + \left[\frac{2w_j - \bar{b}_j - 1}{(\bar{b}_j + 2)w_j - \bar{b}_j - 1}\right] \cdot (1 - \rho^{\bar{\gamma}}). \tag{4.4}$$

(Note that in practice, (4.4) may be an upper bound for $\Delta\bar{B}_j$, since for large n we expect there may be several clusters C to which the item I could be added, and a reasonable entry procedure might choose that "best" cluster C which minimizes the increase in cluster extent, and thus $\bar{B}_j$.)

Now, using (4.2)

$$\begin{aligned}\Delta\bar{b}_j(n) &= (\bar{\gamma}(n) \cdot \Delta\bar{B}_j(n) - \bar{B}_j(n) \cdot \Delta\bar{\gamma}(n))/\bar{\gamma}(n)^2 \\ &= \frac{(\bar{b}_j^2 + \bar{b}_j)\rho^{\bar{\gamma}} - ((\bar{b}_j^2 + \bar{b}_j)\rho^{\bar{\gamma}} - 2)w - \bar{b}_j - 1}{((\bar{b}_j + 2)w_j - \bar{b}_j - 1)\bar{\gamma}}.\end{aligned} \tag{4.5}$$

Expressions (3.7), (4.1) and (4.5) may now be used to set up a set of $m+1$ difference equations in $(\bar{\gamma}, \bar{b}_1, \cdots, \bar{b}_m)$ which, since the independent variable is an integer, may be just evaluated iteratively. These are:

$$\begin{aligned}&\bar{\gamma}(n+1) = \bar{\gamma}(n) + \Delta\gamma(n) = \bar{\gamma}(n) + \rho(n)^{\bar{\gamma}(n)}, \\ &\bar{b}_j(n+1) = \bar{b}_j(n) + \Delta\bar{b}_j(n) \\ &\quad = \bar{b}_j(n) + \frac{(\bar{b}_j(n)^2 + \bar{b}_j(n))\rho(n)^{\bar{\gamma}(n)} - [(\bar{b}_j(n)^2 + \bar{b}_j(n))\rho(n)^{\bar{\gamma}(n)} - 2]w - \bar{b}_j(n) - 1}{((\bar{b}_j^2 + 2)w - \bar{b}_j - 1)\bar{\gamma}(n)},\end{aligned} \tag{4.6}$$

where $\bar{\gamma}(1) = 1$, $\bar{b}_1(1) = \cdots = \bar{b}_m(1) = 1$, and using (3.5)

$$\rho(n) = 1.0 - \prod_{j=1}^{m}\left[\frac{(\bar{b}_j(n) + 2)w_j - \bar{b}_j(n) - 1}{w_j^2}\right].$$

Tables 2 (a, b, c) illustrate numerical solutions for three sets of such difference equations. Initially nearly every new item begins a new cluster. The probability that no $D_I \vee D_C$ is convex, that is $\rho^{\bar{\gamma}}$, is close to 1.0. But as $\bar{\gamma}$ and $\Pi_j\bar{b}_j$ (the expected cluster extent) increase, the likelihood of an item I being added to an existing cluster increases.

TABLE 2(a)

Numbers and extents of unbounded clusters in the space (8, 6, 10, 8).

	w_j	8	6	10	8
n	$\bar{\gamma}(n)$	$\bar{b}_1(n)$	$\bar{b}_2(n)$	$\bar{b}_3(n)$	$\bar{b}_4(n)$
10	9.3	1.047	1.046	1.048	1.047
20	17.4	1.095	1.094	1.096	1.095
40	30.5	1.193	1.190	1.195	1.193
60	40.4	1.292	1.286	1.295	1.292
100	53.5	1.489	1.488	1.505	1.489
150	61.8	1.772	1.753	1.783	1.772
200	65.4	2.056	2.027	2.073	2.056
300	67.3	2.617	2.564	2.647	2.617
400	67.5	3.315	3.045	3.171	3.125
500	67.5	3.573	3.465	3.636	3.573

$m = 4$, $\Pi_j w_j = 3.840$

TABLE 2(b)
Numbers and extents of unbounded clusters in the space (4, 7, 10, 15, 20).

	w_j	4	7	10	15	20
n	$\bar{\gamma}(n)$	$\bar{b}_1(n)$	$\bar{b}_2(n)$	$\bar{b}_3(n)$	$\bar{b}_4(n)$	$\bar{b}_5(n)$
10	9.9	1.006	1.006	1.006	1.006	1.006
20	19.6	1.011	1.012	1.012	1.012	1.012
40	38.6	1.023	1.024	1.024	1.025	1.025
60	56.8	1.034	1.036	1.037	1.037	1.037
100	91.2	1.057	1.061	1.062	1.062	1.063
150	130.7	1.086	1.092	1.093	1.095	1.096
200	166.6	1.117	1.123	1.126	1.127	1.128
300	228.7	1.178	1.188	1.192	1.195	1.196
400	279.6	1.241	1.255	1.261	1.265	1.267
500	320.9	1.307	1.326	1.333	1.338	1.340
600	354.1	1.375	1.399	1.408	1.415	1.418
700	380.4	1.445	1.475	1.486	1.494	1.498
800	401.0	1.518	1.554	1.567	1.576	1.581
900	416.8	1.593	1.635	1.650	1.662	1.667
1,000	428.7	1.669	1.719	1.736	1.750	1.756
1,500	453.0	2.062	2.152	2.185	2.209	2.222
2,000	456.1	2.431	2.571	2.623	2.661	2.679

$m = 5$, $\Pi_j w_j = 84{,}000$

TABLE 2(c)
Numbers and extents of unbounded clusters in the space (5, 10, 15, 20, 25, 30).

	w_j	5	10	15	20	25	30
n	$\bar{\gamma}(n)$	$\bar{b}_1(n)$	$\bar{b}_2(n)$	$\bar{b}_3(n)$	$\bar{b}_4(n)$	$\bar{b}_5(n)$	$\bar{b}_6(n)$
500	494.3	1.007	1.007	1.007	1.008	1.008	1.008
1,000	977.2	1.014	1.015	1.015	1.015	1.015	1.015
1,500	1,449.1	1.021	1.022	1.023	1.023	1.023	1.023
2,000	1,909.9	1.029	1.030	1.031	1.031	1.031	1.031
3,000	2,799.6	1.044	1.046	1.046	1.046	1.047	1.047
4,000	3,647.5	1.059	1.061	1.062	1.062	1.063	1.063
5,000	4,455.2	1.074	1.077	1.078	1.079	1.079	1.079
10,000	7,934.0	1.154	1.161	1.163	1.164	1.165	1.166
15,000	10,585.1	1.241	1.253	1.257	1.258	1.259	1.260
20,000	12,538.7	1.335	1.353	1.359	1.361	1.363	1.364
30,000	14,827.8	1.550	1.582	1.592	1.596	1.599	1.601
40,000	15,705.5	1.791	1.841	1.857	1.864	1.869	1.872
50,000	15,936.6	2.040	2.119	2.135	2.146	2.152	2.157
60,000	15,975.1	2.279	2.377	2.407	2.421	2.430	2.435
70,000	15,978.9	2.502	2.625	2.663	2.681	2.692	2.700
80,000	15,979.1	2.707	2.858	2.904	2.927	2.940	2.949
90,000	15,979.2	2.900	3.077	3.132	3.159	3.175	3.186
100,000	15,979.2	3.079	3.285	3.349	3.380	3.400	3.411

$m = 6$, $\Pi_j w_j = 11{,}250{,}000$

Indeed, with the configuration of Table 2(a), which represents a fairly small 4-dimensional space of $\Pi_j w_j = 8 \cdot 6 \cdot 10 \cdot 8 = 3{,}840$ points, by the time $n = 300$ items have been entered the entire space has been effectively covered by 67 existing clusters. $\rho^{\bar{\gamma}} \to 0.0$ and $\bar{\gamma}(n) \to$ constant, since all newly entered items are included in an existing cluster. But $\bar{b}_j(n)$ keeps increasing as the extent of individual clusters keeps growing.

Table 2(b) illustrates the behavior of clusters in a slightly larger 5-dimensional space of $\Pi_j w_j = 4 \cdot 7 \cdot 10 \cdot 15 \cdot 20 = 84{,}000$ cells. It is much the same, except that when $n = 2{,}000$ an expected $\bar{\gamma} = 456$ clusters have been formed with an expected content of $n/\bar{\gamma} = 4.38$ items and expected extent of $\Pi_j \bar{b}_j = 116{,}207$ cells per cluster. At $n = 1{,}500$ there were virtually the same number of clusters, with expected content $= 3.31$ but with an expected extent of only 47.15.

In a still larger 6-dimensional space of $\Pi_j w_j = 5 \cdot 10 \cdot 15 \cdot 20 \cdot 25 \cdot 30 = 11{,}250{,}000$ cells, after $n = 50{,}000$ items have been entered there will be approximately $\bar{\gamma} = 15{,}936$ distinct clusters with an expected content of only 3.14 items per cluster. After 100,000 items have been entered there will be nearly the same number of clusters, but now each cluster will contain an expected 6.26 items.

5. Practical implications. Bit descriptors, as developed in this paper, can be the key to very efficient multi-attribute file retrieval methods. We are primarily concerned with a retrieval method called "indexed descriptor access" described in detail in [8]. References [1, 7, 10, 11] represent a sample of other multi-attribute retrieval methods which are similar, or related in some way. Virtually all retrieval methods work most effectively when "similar" items are stored together; that is are clustered in some fashion. The item (or record) entry algorithm of § 2 is one way of dynamically organizing any data file.

The solution of the system of $m+1$ difference equations in the preceding § 4, that calculate $\bar{\gamma}$ and $\bar{b}_j$, for $1 \leqq j \leqq m$, provide precisely the information needed to estimate expected retrieval costs, assuming that we will let clusters contain an arbitrary number of items. In [8] it is shown that

$$\bar{\gamma}(n) \cdot \prod_{j \in Q} \frac{\bar{b}_j}{w_j} \tag{5.1}$$

denotes the expected number of accesses in the data file to respond to a query Q which conjunctively specifies some, or all, of the m item attributes. (Here, $\Pi_{j \in Q}$ denotes a product involving those fields j specified in the query.)

If, for example, all m attributes were specified in a retrieval query (normally denoting at most a single record of the data file), then using the final values in Tables 2(a, b, c) and (5.1), the expected storage accesses to the data file are:

file 1	(500 items)	2.831 accesses,
file 2	(2,000 items)	0.631 accesses,
file 3	(100,000 items)	1.886 accesses.

Note that these values do not denote the total cost of retrieval using indexed descriptor access. A few additional accesses must be made in one, or more, index files. But it is not the purpose of this paper to discuss any retrieval method in detail. Still we do want to indicate (1) that this dynamic clustering procedure can be used in conjunction with an effective retrieval scheme; (2) that the resulting analyses can be used to estimate expected retrieval costs; and (3) that retrieval in very large files need

be no more expensive than much smaller files—provided they are represented over a sufficiently large attribute space.

The expressions of the preceding section provide the means for numerically calculating $\bar{\gamma}(n)$ and $\bar{b}_j(n)$ as in Tables 2(a, b, c) and thereby calculating the average number of items per cluster, $n/\bar{\gamma}(n)$. But the tables do not indicate the extreme variability in either the extent or the content of these clusters. A few clusters have large content, while most contain only one or two items. Intuitively, of the many initially scattered singleton clusters, chance dictates that a few will begin to grow. But then the probability that new items will be contained in, or adjacent to, these larger clusters (that is, $D_I \vee D_C$ will be convex) is greater, and so they tend to keep growing; at least until the space is well covered by clusters.

If we are modeling an actual computer file organization in which items (or records) with similar attributes are to be clustered (blocked) together in a data file, then the presence of large clusters is unrealistic. There will be some finite upper limit on the number of records that can form a physical block of the data file. Thus we really want to derive the results of the preceding section, subject to the constraint that "no cluster can contain more than kmax items".

In the next section we add this bounding constraint, and in so doing must use a different analytic approach. The expected number of clusters with precisely k items, $1 \leqq k \leqq \text{kmax}$, must be determined. The results are somewhat surprising, and in many respects more revealing of the behavior of this clustering process in actual practice.

6. Number of clusters containing precisely k items. A *k-cluster*, C_k, is one containing precisely k items, $1 \leqq k \leqq \text{kmax}$. kmax denotes the maximum possible content of (number of items in) any cluster. To express the expected number of these k-clusters and their extents we could use a notation such as $\bar{\gamma}^{(k)}(n)$, $\bar{b}_j^{(k)}(n)$ and $\bar{\varepsilon}_j^{(k)}(n)$ to indicate a functional dependence on both k and n. But rigorous adherence to this notation would lead to unwieldy expressions below. Moreover, we will show that $\bar{b}_j^{(k)}$ and $\bar{\varepsilon}_j^{(k)}$ are not, in fact, dependent on n. Consequently, we will use an abbreviated notation $\bar{\gamma}(k)$, $\bar{b}_j(k)$ and $\bar{\varepsilon}_j(k)$ and let the dependence on n in the case of $\bar{\gamma}(k)$ be tacitly understood. Now

$$\bar{\gamma} = \sum_{k=1}^{\text{kmax}} \bar{\gamma}(k), \qquad \bar{B}_j = \sum_{k=1}^{\text{kmax}} \bar{b}_j(k) \cdot \bar{\gamma}(k), \qquad \bar{b}_j = \frac{\bar{B}_j}{\bar{\gamma}}, \tag{6.1}$$

where the terms $\bar{\gamma}$, $\bar{B}_j$ and $\bar{b}_j$ on the left are identical to $\bar{\gamma}(n)$, $\bar{B}_j(n)$ and $\bar{b}_j(n)$ of preceding sections.

We now concentrate on the behavior of just k-clusters. By definition, for all 1-clusters, $\bar{b}_j(1) = 1.0$, for $1 \leqq j \leqq m$. A new 2-cluster is formed by adding an item to an existing 1-cluster. For each field j there is a distinct probability that the entered item will add another bit to that field of the cluster descriptor. And since all 2-clusters are formed in the same way, the expected number of bits per field in any single 2-cluster descriptor is that of all 2-clusters as a whole. Thus

$$\bar{b}_j(2) = \bar{b}_j(1) + \left[1 - \frac{\bar{b}_j(1)}{\bar{b}_j(1) + \bar{\varepsilon}_j(1)}\right],$$

where the expression in brackets denotes the probability of adding a new bit to field$_j$ of the cluster descriptor, assuming $D_I \vee D_C$ is convex; and where $\bar{\varepsilon}_j(1)$ is calculated from $\bar{b}_j(1)$ using (3.1). In general, for $2 \leqq k \leqq \text{kmax}$

$$\bar{b}_j(k) = \bar{b}_j(k-1) + [1.0 - \bar{b}_j(k-1)/(\bar{b}_j(k-1) + \bar{\varepsilon}_j(k-1))]. \tag{6.2}$$

Note that $\bar{b}_j(k)$, the expected number of bits set in field$_j$ of a k-cluster, and $\bar{\varepsilon}_j(k)$ are both independent of n or $\gamma(k)$, the number of such clusters.

Let I be a new item entered into the space. Following the clustering algorithm of § 2, by Condition (1) I will be added to some existing C_k only if $D_I \vee D_{C_k}$ is convex, and by Condition (2) if $D_I \vee D_{C_j}$ is *not* convex for any $1 \leqq j < k$. The item I will start a new singleton cluster only if $D_I \vee D_{C_k}$ is not convex for any C_k, $1 \leqq k \leqq \text{kmax}-1$. (Note that I cannot be added to any kmax-cluster.) Consequently,

$$\begin{aligned}&\text{prob (adding } I \text{ to a } k\text{-cluster, } 1 \leqq k < \text{kmax})\\ &\quad = \text{prob } (\forall C_j, I < k, D_I \vee D_{C_I} \text{ is not convex}) \cdot \text{prob } (\exists C_k, D_I \vee D_{C_k} \text{ is convex}).\end{aligned} \tag{6.3}$$

And, in general, the incremental change in $\bar{\gamma}(k)$ as a result of entering a new item I into the space can be expressed by

$$\Delta\bar{\gamma}(k) = \text{prob (adding } I \text{ to a } (k-1)\text{-cluster} - \text{prob (adding } I \text{ to a } k\text{-cluster}). \tag{6.4}$$

The second term accounts for the fact that every time a $(k+1)$-cluster is formed from a k-cluster, $\gamma(k)$ is decreased by one. This expression must be modified in the case of $k=1$ and $k=\text{kmax}$. When $k=1$, the first term of (6.4) is the probability that $D_I \vee D_{C_i}$ is not convex *for any cluster*, $i < \text{kmax}$. When $k = \text{kmax}$, the second term is simply omitted.

The probabilities of (6.3) may be calculated in two different ways. First we assume, as in preceding sections, that k-clusters are uniformly distributed through the space in an independent manner, so that using (3.4) we have

$$\text{prob } (D_I \vee D_{C_k} \text{ is convex}) = \prod_{j=1}^{m} (\bar{b}_j(k) + \bar{\varepsilon}_j(k))/w_j,$$

where again this probability is independent of n. As in §§ 3 and 4 we let $\rho(k)$ denote the probability that $D_I \vee D_{C_k}$ is not convex for any particular k-cluster. Thus (6.3) becomes

$$\text{prob (adding } I \text{ to a } k\text{-cluster, } 1 \leqq k < \text{kmax}) = \prod_{i<k} \rho(j)^{\bar{\gamma}(j)} \cdot [1.0 - \rho(k)^{\bar{\gamma}(k)}], \tag{6.5}$$

which when substituted into (6.4) yields the following difference expressions

$$\Delta\bar{\gamma}(1) = \prod_{k<\text{kmax}} \rho(k)^{\bar{\gamma}(k)} - (1.0 - \rho(1)^{\bar{\gamma}(1)})$$

$$\begin{aligned}\Delta\bar{\gamma}(k) &= \prod_{i<k-1} \rho(j)^{\bar{\gamma}(j)} \cdot (1.0 - \rho(k-1)^{\bar{\gamma}(k-1)}) - \prod_{i<k} \rho(j)^{\bar{\gamma}(j)} \cdot (1.0 - \rho(k)^{\bar{\gamma}(k)})\\ &= \prod_{i<k-1} \rho(j)^{\bar{\gamma}(j)} \cdot [(1.0 - \rho(k-1)^{\bar{\gamma}(k-1)}) - \rho(k-1)^{\bar{\gamma}(k-1)} \cdot (1.0 - \rho(k)^{\bar{\gamma}(k)})]\end{aligned}$$

for $2 \leqq k \leqq \text{kmax}-1$, and

$$\Delta\bar{\gamma}(\text{kmax}) = \prod_{i<\text{kmax}-1} \rho(j)^{\bar{\gamma}(j)} \cdot [1.0 - \rho(\text{kmax}-1)^{\bar{\gamma}(\text{kmax}-1)}]. \tag{6.6}$$

With these one can set up kmax difference equations of the form

$$\bar{\gamma}^{(k)}(n+1) = \bar{\gamma}^{(k)}(n) + \Delta\bar{\gamma}(k) \tag{6.7}$$

and evaluate them iteratively using $\bar{\gamma}^{(1)}(1) = 1.0$, $\bar{\gamma}^{(k)}(1) = 0.0$, $2 \leqq k \leqq \text{kmax}$, as initial conditions.

The iterative evaluation of this set of equations yields values which are reasonably close to those values obtained in practice. For example, the maximum relative error

between the expected $\bar{\gamma}(n)$ calculated in this fashion, and the actual $\gamma(n)$ observed in more than 50 generated clustered files over different attribute spaces is less than 0.19.

However, it is intuitively evident that k-clusters may not be independently distributed throughout the space. This leads to a second way of determining at least the first two differences of (6.6).

Consider just the case of 1-clusters. A new 1-cluster will be created by a newly entered item only if it is not within or adjacent to any existing k-cluster, $k < \text{kmax}$. 1-clusters are *formed* only in that portion of the space not covered by k-clusters. (Note that in this dynamic process, 1) 1-clusters which were formed at any earlier time may *exist* in other portions of the space if they have not been subsequently enlarged; and 2) once a cluster has its maximal possible content, kmax, it no longer affects the clustering process—it is effectively no longer there.)

Now for 1-clusters we can re-evaluate (6.3) as

$$\begin{aligned}\text{prob (adding } I \text{ to a 1-cluster)} &= \text{prob } (\exists \text{ a 1-cluster } C, D_I \vee D_C \text{ is convex}) \\ &= \sum_j \text{prob } (D_I \vee D_{C_j} \text{ is convex}) \\ &\quad - \sum_{i,j} \text{prob } (D_I \vee D_{C_i} \text{ and } D_I \vee D_{C_j} \text{ are convex}) \\ &\quad + \sum_{i,j,k} \cdots,\end{aligned}$$

where the sums are run over all 1-clusters C_i, C_j, C_k. The first term of this expansion is simply

$$\bar{\gamma}(1) \cdot \prod_j (\bar{b}_j(1) + \bar{\varepsilon}_j(1)) \Big/ \prod_j w_j.$$

If the 1-clusters were uniformly distributed then the second term would be

$$\binom{\bar{\gamma}(1)}{2} \cdot \prod_j \exp \text{ (overlap in field}_j) \Big/ \prod_j w_j$$

into which we could substitute (3.2).

But (3.2) was derived assuming that the clusters were distributed over the entire space. They are not. They are confined to a much smaller portion of the space, and the expected intersection is higher than that predicted by (3.2). We can approximately account for this by replacing the actual field widths, w_j, in (3.2) with apparent field widths, call them w'_j, where $\prod_j w'_j =$ total volume of space *not* covered by k-clusters, $2 \leqq k \leqq \text{kmax}$. We approximate this reduction factor by

$$F_1 = \prod_{k \geqq 2} \rho(k)^{\bar{\gamma}(k)}.$$

Then we set $w'_j = F_1^{1/m} \cdot w_j$. Since $\bar{b}_j(1) = 1.0$ for all j, (3.2) becomes, in the case of 1-clusters

$$\exp \text{ (overlap)} = \frac{1}{w_j'^2} [3w_j'^2 + 4w'_j - 8]$$

and (6.3) becomes

$$\begin{aligned}&\text{prob (adding } I \text{ to a 1-cluster)} \\ &= \frac{1}{\prod_j w_j} \cdot \left[\bar{\gamma}(1) \cdot \prod_j (\bar{b}_j(1) + \bar{\varepsilon}_j(1)) - \binom{\bar{\gamma}(1)}{2} \cdot \prod_j (3w_j'^2 + 4w'_j - 8/w_j'^2 + \cdots)\right],\end{aligned} \tag{6.8}$$

where the third and succeeding terms of the expansion are ignored.

There are enough approximations in this refined derivation to make it unreliable for larger k-clusters. But observation of the clustering process indicates that the distribution of k-clusters becomes more nearly uniform as $k \to \text{kmax}$ anyway. We have found that use of the alternate computation of prob (adding I to a 1-cluster) in the first two differences of (6.6) leads to slightly more accurate expected values when compared to actual behavior. This refinement was used to generate Figs. 3(a, b, c) and Tables 5 and 6 of the following section.

However calculated, the behavior of the solutions of the set of difference equations (6.7) is quite interesting. As one would intuitively expect, only 1-clusters are formed initially. Later, as these begin forming 2-clusters with successively entered items (which in turn serve as the nuclei of 3-clusters), the number of 1-clusters, $\bar{\gamma}(1)$, decreases. As shown in Fig. 3(a) (which denotes the same space as Table 2(a), but with kmax = 5), $\bar{\gamma}(1), \bar{\gamma}(2), \cdots, \bar{\gamma}(\text{kmax}-1)$ all oscillate periodically. But this periodic behavior dies out and an equilibrium is achieved in which $\bar{\gamma}(1), \bar{\gamma}(2), \cdots, \bar{\gamma}(\text{kmax}-1)$ become effectively constant. Only $\bar{\gamma}(\text{kmax})$ will be monotone increasing.

Once equilibrium is attained, the expected number of all clusters, $\bar{\gamma}(n)$, can be expressed by the simple linear equation

$$\bar{\gamma}(n) = c_1 + c_2(n - c_3), \tag{6.9}$$

where $c_1 = \sum_{k<\text{kmax}} \bar{\gamma}(k)$, $c_2 = 1/\text{kmax}$ and $c_3 = \sum_{k<\text{kmax}} k \cdot \bar{\gamma}(k)$.

In Figs. 3(b, c), which duplicate the spaces of Tables 2(b, c) subject to the constraints of kmax = 4 and 3 respectively, the oscillation of $\bar{\gamma}(1), \cdots, \bar{\gamma}(\text{kmax}-1)$ is less apparent, since the amplitude is small and with much longer period; but the steady state behavior is evident.

7. How accurate are these solutions? Given the dimensions $(i_1, i_2, \cdots, i_m)$ of any particular attribute space, one may use (4.6), or (6.7) to derive an expected $\bar{\gamma}(n)$. To test their accuracy a number of files were actually created using the insertion algorithm of § 2 and randomly generated data.

First, we verify the expected number of clusters with unbounded content predicted by (4.6). The expected $\bar{\gamma}(n)$ for spaces of dimensions (8, 6, 10, 8) given in Table 2(a). Table 4 compares those expected values with those of the test files. Five test files of 500 records each were created with no constraint on maximum cluster content, $\bar{\gamma}(n)$ is seen to be consistently greater than $\gamma(n)$. The derivation of (4.6) takes no account of the variability of either cluster extent or content. All probabilities are based on an average cluster content $\prod_j \bar{b}_j$. However, the creation of unbounded clusters by the entry algorithm of § 2 tends to be unstable with respect to perturbations from the mean. A cluster which is already large is more likely to "capture" a newly entered item than a smaller cluster. Thus large clusters tend to become very large, eventually dominating the space, while the remaining clusters remain small. Condition (2) of the entry procedure minimizes this instability somewhat, but not completely.

If it were practical to create unbounded clusters and retrieve items from them, then either a revised algorithm or a refined analysis would be essential. Since this is not so, Table 4 (and similar unprinted comparisons) should be sufficient to convince us of at least the basic correctness of (4.6). In contrast, with bounded cluster creation as considered in the preceding section, the behavior of the entry algorithm is "self-correcting" with respect to cluster extent. Large clusters still grow more rapidly. But once their content reaches kmax, they no longer affect the entry procedure; they cannot dominate the space.

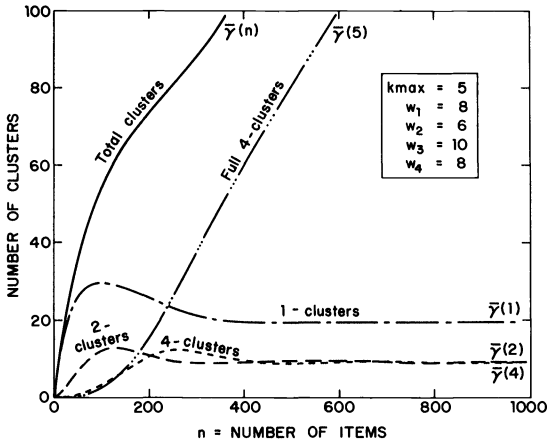


FIG. 3(a)

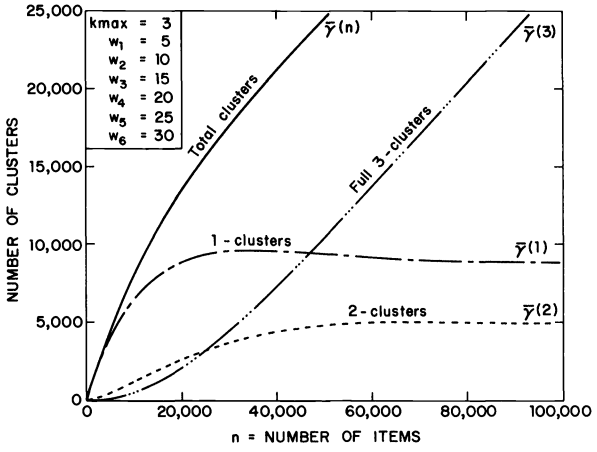


FIG. 3(b)

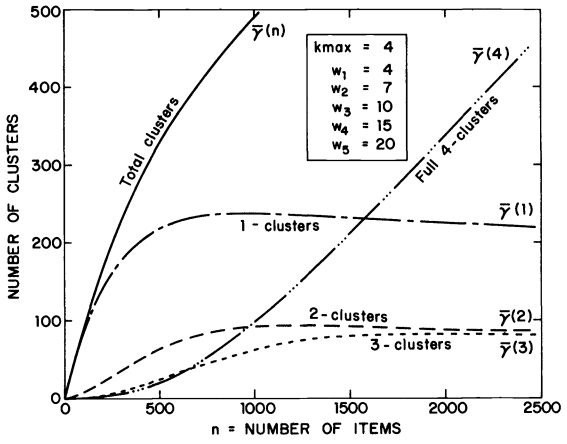


FIG. 3(c)

TABLE 4

Comparison of expected vs. observed numbers of clusters in the space (8, 6, 10, 8), *with unbounded cluster content.*

n	expected $\bar{\gamma}(n)$	observed $\gamma(n)$ file 1	file 2	file 3	file 4	file 5
10	9.4	10	10	9	9	10
20	17.4	17	20	18	18	19
40	24.5	28	31	32	31	31
60	40.4	34	39	39	43	42
80	47.9	37	45	44	46	50
100	53.5	39	51	47	52	53
150	61.8	44	60	54	57	54
200	65.5	46	60	55	60	55
300	67.3	48	63	55	61	59
400	67.5	48	63	55	61	59
500	67.5	48	63	55	61	59

To verify the more important derivation presented in § 6, we have calculated the expected $\bar{\gamma}(n)$ and $\bar{\gamma}(1), \cdots, \bar{\gamma}(\text{kmax})$ using (6.7) for each of the same three spaces given in the preceding examples; and then executed a series of five actual clustering runs for each. Maximum cluster bounds of kmax = 5, 4 and 3 respectively were used. The choice of these bounds is derived from tables 2(a, b, c). These represent, in each case, the average content of a cluster when the space was first "effectively covered". Larger, or smaller, cluster bounds may be dictated by other constraints in individual applications; but in the absence of such constraints these bounds will yield optimal retrieval performance. Indeed, the major value in practice of the derivation of § 4 is to be able to establish approximately optimal cluster bounds.

Table 5 compares the expected numbers of clusters with the observed means of the five test files generated over the space with dimensions (8, 6, 10, 8). Only the expected total clusters, $\bar{\gamma}(n)$, and the expected 1-clusters and 5-clusters, $\bar{\gamma}(1)$ and $\bar{\gamma}(\text{kmax})$ are displayed. The value of $\bar{\gamma}(n)$ is quite accurate. The expected value deviates

TABLE 5

Comparison of expected vs. observed numbers of bounded clusters in the space (8, 6, 10, 8); *max. cluster content* = kmax = 5.

n	expected $\bar{\gamma}(n)$	observed mean	expected $\bar{\gamma}(1)$	observed mean	expected $\bar{\gamma}(5)$	observed mean
20	17.4	17.4	15.2	16.2	0.0	0.0
60	40.3	38.0	27.4	25.4	0.6	0.6
100	54.1	50.2	29.5	28.0	2.8	3.4
200	73.5	69.6	25.7	21.8	15.6	16.6
300	88.1	90.4	21.2	20.0	37.6	37.8
400	106.0	113.2	19.5	16.8	60.2	55.2
500	126.1	127.6	19.6	17.8	80.6	76.2
600	146.2	148.4	19.5	20.0	100.4	98.2
700	166.1	168.4	19.4	21.9	120.4	119.8
800	186.1	188.4	19.5	20.8	140.5	138.8
900	206.1	208.6	19.5	21.6	160.5	156.4
1,000	226.1	229.4	19.5	18.0	180.5	177.6

from the observed mean by at most −7.2 when $n = 400$. This corresponds to a maximum relative error of 0.068.

The estimation of expected k-clusters is somewhat less accurate. An initial periodic behavior as shown in Fig. 3(a) can be seen in all individual runs; but the phase and amplitude of these periods, being largely determined by random events, can be very variable. For example, when $n = 800$, the observed values of $\gamma(1)$ were 21, 17, 17, 17 and 32 with a mean of 20.8, but standard deviation of 5.81. However, by $n = 500$ the predicted equilibrium and consequent general stability are apparent in most individual cluster systems.

Table 6 displays a similar comparison of expected versus observed numbers of clusters in files clustered over the slightly larger space with dimensions (4, 7, 10, 15, 20). The prediction of expected total clusters is somewhat better. A maximum relative deviation from the observed mean of 0.016 occurs when $n = 800$. Again the estimations of $\bar{\gamma}(1)$ and $\bar{\gamma}(4)$ are less accurate.

TABLE 6

Comparison of expected vs. observed numbers of bounded clusters in the space (4, 7, 10, 15, 20); *max. cluster content* = kmax = 4.

n	expected $\bar{\gamma}(n)$	observed mean	expected $\bar{\gamma}(1)$	observed mean	expected $\bar{\gamma}(4)$	observed mean
100	91.0	91.6	83.3	83.6	0.2	0.0
200	165.9	166.4	139.4	139.8	1.7	1.6
300	227.9	229.0	176.6	177.4	5.5	6.0
400	280.2	277.5	200.9	198.8	11.9	12.2
500	325.2	322.2	216.6	211.5	20.9	20.4
600	364.7	365.0	226.5	229.0	32.2	33.4
700	400.0	397.3	232.3	233.0	45.8	45.4
800	432.1	425.2	235.6	226.0	61.4	68.2
900	461.8	455.0	237.1	227.6	78.9	79.0
1,000	489.7	485.2	237.4	229.8	98.0	95.5
1,100	516.2	510.8	236.9	229.3	118.6	119.5
1,200	541.6	537.7	236.0	232.0	140.4	139.8
1,300	566.6	560.6	234.7	235.5	163.3	161.0
1,400	590.6	588.8	233.3	229.0	187.0	179.5
1,500	614.5	614.0	231.8	232.6	211.4	203.0
1,600	638.2	638.9	230.2	234.2	236.3	228.7
1,700	661.8	664.2	228.7	235.4	261.5	249.7
1,800	685.5	687.6	227.3	231.3	287.0	269.8
1,900	709.1	712.0	225.8	234.3	312.6	303.7
2,000	732.9	738.6	224.6	229.7	338.2	326.8

Table 7 was compiled to test the validity of the difference equations (6.7) in large files, where the payoff of any clustering procedure is most pronounced. To generate Table 7, five files of 40,000 items each were clustered over the still larger space of dimensions (5, 10, 15, 20, 25, 30). This represents a space of more than 11 million virtual cells. The maximum relative deviation of the predicted mean $\bar{\gamma}(n)$ from the observed mean is 0.01. That any system of 1-point boundary difference equations should attain this measure of accuracy after 40,000 steps attests to the inherent stability of the difference expressions of (6.6).

TABLE 7

Comparison of expected vs. observed number of bounded clusters in the space (5, 10, 15, 20, 25, 30); *max. cluster content* = kmax = 3.

n	expected $\bar{\gamma}(n)$	observed mean	expected $\bar{\gamma}(1)$	observed mean	expected $\bar{\gamma}(3)$	observed mean
1,000	977.0	987.3	955.0	975.0	0.9	0.8
2,000	1,908.5	1,907.5	1,824.3	1,819.2	7.2	4.7
3,000	2,796.2	2,811.6	2,614.5	2,646.0	22.9	24.1
4,000	3,642.1	3,658.4	3,332.2	3,376.3	48.0	56.8
5,000	4,448.9	4,472.8	3,983.7	4,032.7	86.0	97.2
10,000	7,984.7	8,006.3	6,436.4	6,452.0	467.0	452.5
15,000	10,892.1	10,886.3	7,925.5	7,911.1	1,141.3	1,130.4
20,000	13,374.3	13,387.0	8,802.5	8,810.6	2,053.9	2,043.7
25,000	15,565.1	15,663.2	9,285.4	9,436.4	3,155.2	3,122.3
30,000	17,554.1	17,748.0	9,514.3	9,827.8	4,406.0	4,341.9
35,000	19,403.1	19,672.9	9,582.0	9,983.1	5,775.7	5,684.0
40,000	21,156.1	21,542.4	9,551.5	10,133.7	7,239.4	7,121.8

8. Summary and conclusions. The asymptotic behavior of $\bar{\gamma}(n)$, the expected number of clusters in a finite m-dimensional space, depends on assumptions about the clusters themselves. If the clusters are unbounded, as in § 4, then $\bar{b}_j$, and hence cluster extent, are monotone increasing as a function of n until the space is effectively covered by a few clusters so that $\bar{\gamma}(n) \to c$. And in § 6, where a constraint of a maximum cluster content is imposed, $\bar{\gamma}(n) \to c_1 + c_2(n - c_3)$.

In actual computer applications the latter model is more realistic, since practical considerations invariably impose an upper limit on the number of items that may be clustered in a data file. Its linear behavior has profound implications. In very large data bases, a dynamic clustering algorithm which left many blocks of the data file only partially filled with items would impose intolerable storage overhead. The fact that convex clustering procedures attain an equilibrium, with a fixed upper bound on the number of partially filled physical blocks is crucial. Indeed, the overhead of unused storage, as a percent of the total data file, actually decreases as n becomes large.

Expected storage accesses per query can also be derived from this model by first using (6.2) to calculate $\bar{b}_j(k)$ for $1 \leqq k \leqq \text{kmax}$, and then using (6.1) to calculate $\bar{b}_j(n)$ for $1 \leqq j \leqq m$. With the large file of Fig. 3(c), when $n = 100{,}000$ one obtains $\bar{\gamma}(n) =$ 40,793 and $\bar{b}_j(n) = 1.801$, 1.845, 1.860, 1.867, 1.871 and 1.874 respectively for $j = 1, 2, \cdots, 6$. Comparing these values with those of Table 2(c), one finds more clusters (40.793 vs. 15,979) of smaller extent (40.46 vs. 1327.81 cells per cluster). By using these values, one can now show that to retrieve a single item from the file, an expected $\bar{\gamma}(n) \cdot \prod_{j=1}^{6} \bar{b}_j(n)/w_j = 0.1467$ accesses to the data file will be required. Retrieval performance of this order, which is considerably better than that described in § 5 and which has been verified in actual data files, is impressive.

Readily, the analyses of this paper were undertaken to determine the expected behavior of a particular access and retrieval method (described in [8] and [9]), that represents the attributes of data items by bit descriptors to form a descriptor for a block of several items. This is not an original access method. The first known reference to this technique seems to be [7]. Edgar Cagley independently discovered this approach and, moreover, was the first to implement it in a practical system [5]. Our system closely follows Cagley's design; indeed he provided it, although we use a different clustering mechanism.

Many retrieval systems have employed binary descriptors to represent items; but most form the descriptor by superimposed coding which does not separate distinct attributes; [3] and [10] are representative examples (the latter provides an extensive list of further references). Superimposed coding is superior for a different set of retrieval problems, those for which a single attribute of an item may have a set of values; and leads to a different kind of analysis.

Retrieval of items in an m-dimensional cellular bucket space has been analyzed in [1], [4]. Since the access method is different, neither item descriptors nor clustering is employed, these results are not directly transferable, but there are some definite similarities. In particular they show that as the size of the conceptual attribute space increases, the cost of retrieval decreases.

There is an abundance of "clustering algorithms" in the literature; [2] and [6] provide fine surveys. Most operate over continuous m-dimensional spaces assigning centroids to created clusters and computing metric distances between them. Moreover most require that all the items be known at the time of clustering; newly entered items may cause a restructuring of the clusters. A dynamic data base requires a one-pass procedure that can organize its items "on the fly". Ours is unusual, but hardly unique, cf. [11]. Much of its appeal lies in the fact that it is conceptually simple and easy to implement, that its expected behavior can be analyzed and its performance predicted, and that it works well in practice.

REFERENCES

[1] A. V. AHO AND J. D. ULLMAN, *Optimal partial match retrieval when fields are independently specified*, ACM Trans. Database Sys., 4 (1979), pp. 168–179.

[2] M. R. ANDERBERG, *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[3] W. J. BERMAN, ISIS *users manual*, NASA Tech. Memo. 80144, Hampton, VA, March 1980.

[4] W. J. BERMAN AND J. L. PFALTZ, *Multi-dimensional bucket arrays*, DAMACS Tech. Report TR-16-78, Univ. of Virginia, Charlottesville, 1978.

[5] E. M. CAGLEY, et al., *Information management system reference manual*, GSA/FPA/MCL TM-208, October 1976.

[6] B. EVERITT, *Cluster Analysis*, John Wiley, New York, 1974.

[7] W. D. FRAZER, *A proposed system for multiple descriptor data retrieval*, Some Problems in Information Science, M. Kochen, ed., The Scarecrow Press, Grolier, Danbury, CN, 1965.

[8] J. L. PFALTZ, W. J. BERMAN AND E. M. CAGLEY, *Partial-match retrieval using indexed descriptor files*, Comm. ACM, 23 (1980), pp. 522–528.

[9] J. L. PFALTZ, *Efficient multi-attribute retrieval over very large geographical data files*, Proc. AUTO-CARTO IV, Washington, DC, 1979, pp. 54–62.

[10] C. S. ROBERTS, *Partial-match retrieval via the method of superimposed codes*, Proc. IEEE, 67 (1979), pp. 1624–1642.

[11] G. SALTON AND A. WONG, *Generation and search of clustered files*, ACM Trans. Database Sys., 3 (1978), pp. 321–346.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0011 $01.25/0

NEW RESULTS ON THE COMPLEXITY OF p-CENTER PROBLEMS*

NIMROD MEGIDDO†‡ AND ARIE TAMIR†

Abstract. An $O(n \log^3 n)$ algorithm for the continuous p-center problem on a tree is presented. Following a sequence of previous algorithms, ours is the first one whose time bound in uniform in p and less than quadratic in n. We also present an $O(n \log^2 n \log\log n)$ algorithm for a weighted discrete p-center problem.

Key words. location, p-centers, parallel computation, tree partitioning, parametric combinatorial optimization

1. Introduction. The p-center problems are defined on a weighted undirected graph $G=(V, E)$. Each edge (i, j) has a positive length d_{ij}. An edge is identified with a line segment of length d_{ij} so that we can refer to any "point" on (i, j) at a distance of t from i and $d_{ij}-t$ from j $(0 \leqq t \leqq d_{ij})$. The set of all such points of the graph is denoted by A. If $x, y \in A$ then by $d(x, y)$ we mean the length of the shortest path from x to y.

The *continuous* p-center problem is to find p points $y_1, \cdots, y_p \in A$ so as to minimize

$$\max_{x \in A} \min_{1 \leqq j \leqq p} d(x, y_j).$$

Intuitively, we wish to locate p "centers" anywhere on the graph so as to minimize the maximum distance between any point and its respective nearest center. An optimal solution $y_1, \cdots, y_p$ is called a p-center and the corresponding largest distance is denoted r_p and is called the p-radius. Results on other versions of the p-center problem in which the supply points must be located on vertices can be found in [6], [7], [8], [9], [10], [11], [12], [13], [16]. Such problems have been shown to be NP-hard on general graphs [13] and it has been conjectured that the continuous p-center problem should be "difficult" on general graphs and, indeed, we include in the last section a proof of its NP-hardness. Our discussion here is hence limited to tree graphs. Our main result in this paper is a substantial improvement of the upper bound on the complexity of the continuous p-center problem on a tree. We devise in this paper an algorithm which runs in $O(n \log^3 n)$ time, $n=|V|$. As is apparent from Table 1, the

TABLE 1
Algorithms for the continuous p-center problem on a tree.

	Reference		Bound
(1)	Handler	[11], [12]	$O(n)$ for $p \leqq 2$
(2)	Chandrasekaran & Daughety	[2]	polynomial, not specified
(3)	Chandrasekaran & Tamir	[4]	$O(\min(n^2 \log^2 p, n^2 \log n + p \log^2 n))$
(4)	Megiddo, Tamir, Zemel & Chandrasekaran	[16]	$O(\min(n^2 \log p, pn \log^2 n))$
(5)	Chandrasekaran & Daughety	[3]	$O(n^2 \log p)$
(6)	Frederickson & Johnson	[6], [7], [8]	$O(n \min(p, n) \log(\max(\binom{p}{n} \binom{n}{p})))$
(7)	Megiddo & Tamir		$O(n \log^3 n)$

* Received by the editors November 13, 1981, and in final form November 5, 1982.

† Department of Statistics, Tel Aviv University, Tel Aviv, Israel.

‡ The work of this author was partially supported by the National Science Foundation under grant ECS-8121741. Currently visiting Department of Computer Science, Stanford University, Stanford, California 94305.

improvement is in two respects. First, our bound is the only one which is uniform with respect to p, while all previous algorithms run in time which tends to infinity with $\log p$, even when the tree is fixed. Secondly, all previous bounds are at least quadratic with n (for general p) while ours is $o(n^{1+\varepsilon})$ for any $\varepsilon > 0$; however, for values of p that are $O((\log n)^2)$, Frederickson and Johnson's algorithm is better.

In order to explain the contribution of the present paper, we start with an overview of the previous results. The first polynomial algorithm was given by Chandrasekaran and Daughety [2]. This algorithm is in fact an application of a general method of solving parametric combinatorial problems presented in Megiddo [14]. The parametrization enters here in the following way. Consider the problem $P(r)$ of locating a minimum number of "centers", $M(r)$, so that every point is within a distance r from at least one center. The p-center problem is to minimize the value of the parameter r subject to $M(r) \leqq p$. The function $M(r)$ is a step function and the problem $P(r)$ is solvable in $O(n)$ time for a fixed r [2]. The method presented in [14] simulates the computation of $M(r)$ where r belongs to a certain interval and is not just fixed at a unique value. The interval is repeatedly narrowed, always containing the minimal r such that $M(r) \leqq p$. It then finds that minimal r exactly. The details are further developed throughout the present paper.

Chandrasekaran and Tamir [4] proved that the jump points of $M(r)$ are of the form $d(x, y)/2l$ where l is integral and x, y are vertices of degree 1. In particular, the p-radius r_p belongs to the set

$$R = \left\{ \frac{d(x, y)}{2l} : x, y \in V, 1 \leqq l \leqq p \right\}.$$

Chandrasekaran and Tamir's algorithm is based on that fundamental result. Note that the cardinality of R is $O(n^2 p)$. However, R has a special structure which enables searching in R without generating the entire set in advance. Indeed, Megiddo, Tamir, Zemel and Chandrasekaran [16] found a succinct representation of all the distances $d(x, y)$, which allows for finding the kth longest path in the tree as well as solving discrete p-center problems in $O(n \log^2 n)$ time despite the cardinality of $\{d(x, y)\}$ being $O(n^2)$. This representation also yielded the $O(\min(n^2 \log p, pn \log^2 n))$ bound for the continuous p-center problem. Later, Frederickson and Johnson [5], [6], [7], [8] found an even better representation which yielded an $O(n \log n)$ algorithm for the unweighted discrete problems as well as the $O(n \min(n, p) \cdot \log(\max(p/n, n/p)))$ algorithm for the continuous p-center problem. Our algorithm in this paper is based on a more efficient search in the set R which exploits both the special structure of the set $\{d(x, y)\}$ and the monotonicity with respect to l (see the definition of R). The search employs parallel computation algorithms along the lines suggested in Megiddo [15].

Another related problem which we attack in the present paper is the following *weighted* discrete p-center problem on a tree. Assuming that every vertex i has a positive weight w_i associated with it, find p points $y_1, \cdots, y_p \in A$ so as to minimize

$$\max_{i \in V} \min_{1 \leqq j \leqq p} w_i d(i, y_j).$$

This problem is solved in [13] in $O(n^2 \log n)$ time. A better implementation yielding an $O(n^2)$ bound appears in [4]. The optimal value of the objective function (i.e. the analogue or r_p) is known [13] to belong to the set

$$\left\{ \frac{d(x, y)}{w_x^{-1} + w_y^{-1}} : x, y \in V \right\}$$

which is of cardinality $O(n^2)$. Applying methods similar to those used in the continuous case, we find the solution in this case in $O(n \log^2 n \log\log n)$ time. We remark that the case where the p-centers may be established only at the vertices is solved in [16] in $O(n \log^2 n)$ time.

2. The continuous problem.

2.1. An overview. It has been pointed out earlier that the p-radius is of the form $d(x, y)/2l$ where $x, y \in V$ and $1 \leqq l \leqq p$. It will be convenient for us to deal with the set R of all these candidates for r_p by looking at the functions $k_{xy}(r)$, defined for all $x, y \in V$:

$$k_{xy}(r) = \left\lfloor \frac{d(x, y)}{2r} \right\rfloor \qquad (r > 0).$$

Obviously, k_{xy} is a step function with jumps at $\lfloor d(x, y)/2l \rfloor$, $l = 1, 2, \cdots$. We will determine an interval $[a, b]$ such that $M(a) > p \geqq M(b)$ and such that all the k_{xy}'s are constant over $(a, b]$. These characteristics imply that $a < r_p \leqq b$ (since $r_p = \text{Min}\{r: M(r) \leqq p\}$) and hence $r_p = b$, since at least one of the functions k_{xy} must jump at r_p.

The determination of the final interval is carried out in two phases. During the first phase an interval $[a_0, b_0]$ is found such that $a_0 < r_p \leqq b_0$ and at least the $k_{ij}(r)$'s corresponding to *edges* (i, j) are constant over $(a_0, b_0]$. During the second phase the set of pairs (x, y) for which $k_{xy}(r)$ is constant is gradually increased (while the interval is gradually narrowed down) until we reach the final interval. The second phase is organized in the form of $O(\log n)$ stages determined by a centroid decomposition [16], [6] of the tree. During each stage we consider $O(n^2)$ pairs of vertices, however, only $O(\log^2 n)$ jump points are tested, i.e., $M(r)$ is evaluated at no more than $O(\log^2 n)$ jump points.

2.2. Phase 1. Let the tree be rooted at an arbitrary vertex u. We will use the convention that if (i, j) is an edge such that j is closer to u than i, then i belongs to the set of points of (i, j) while j does not. Thus every vertex except for u belongs precisely to one edge.

Consider the related problem $P(r)$ (see § 1). It is easy to verify the truth of the following:

ASSERTION 1. *In order for every point to be within a distance of r from at least one center, at least $k_{ij}(r)$ centers must be located on the edge (i, j) (including exactly one endpoint).*

ASSERTION 2. *In order to satisfy the requirement mentioned in Assertion* 1 *with respect to points of the edge (i, j) it is sufficient to allocate $k_{ij}(r) + 1$ centers to that edge.*

COROLLARY. *If $f(r) = \sum_{(i,j) \in E} k_{ij}(r)$ then*

$$f(r) \leqq M(r) \leqq f(r) + (n - 1).$$

Consider the case $r = r_p$. First, by definition $M(r_p) \leqq p$. We also claim that $p - (n-1) < M(r_p)$. For if $M(r_p) + (n-1) \leqq p$ then $r_p \leqq r_{M(r_p)+(n-1)}$ and $r_{M(r_p)+(n-1)} < r_{M(r_p)}$ since by adding one more center per edge the radius certainly decreases. On the other hand, obviously $r_{M(r_p)} = r_p$ and a contradiction has been reached. It now follows that

$$p - 2(n-1) < M(r_p) - (n-1) \leqq f(r_p) \leqq M(r_p) \leqq p.$$

For every r the function f jumps at r if and only if one of the k_{ij}'s jumps at r. Consider the number $r' = \frac{1}{2p} \sum_{(i,j) \in E} d(i, j)$. By the definition of f, $f(r') \leqq \sum_{(i,j) \in E} d(i, j)/2r' = p$. Also $f(r') > \sum_{(i,j) \in E} ((d(i,j)/2r') - 1) = p - (n-1)$. We have, thus,

obtained the following bounds,

$$p-2(n-1)<f(r_p)\leqq p,$$
$$p-(n-1)<f(r')\leqq p.$$

Compute $M(r')$ to determine whether $r'\geqq r_p$ or $r'<r_p$. Suppose first that $r'\geqq r_p$ and imagine that we continuously decrease r, starting from $r=r'$ until we reach $f(r)=p+1$. It follows from the above discussion that we will approach at most $p+1-f(r')<n$ jump points of f. All these jumps are at points of the form $\frac{1}{2}d_{ij}/(k_{ij}(r')+l)$ where $1\leqq l\leqq p+1-f(r')$. As a matter of fact, these jumps occur at the $p+1-f(r')$ largest elements of the set

$$R_0=\left\{\frac{1}{2}\frac{d_{ij}}{k_{ij}(r')+l}: (i,j)\in E,\ l=1,\cdots,p+1-f(r')\right\}.$$

Since R_0 is naturally partitioned into $n-1$ sorted subsets, corresponding to the $n-1$ edges, we can find and sort all these jumps in $O(n+(p+1-f(r'))\log n)=O(n+\min(n,p)\log n)$ time using a standard priority queue [1]. Similarly, if $r'<r_p$ imagine that we continuously increase r, starting with $r=r'$ until we reach $f(r)=\max(0,p-2(n-1))$. It follows that we will approach at most $f(r')-\max(0,p-2(n-1))\leqq\min(2n,p)$ jump points of f. Using the scheme of the previous case, all these jumps are found in $O(n+(f(r')-\max(0,p-2(n-1))\log n))$ time. In any case, the effort so far is $O(n+\min(n,p)\log n)$.

Once we have all the jumps in the relevant domain (depending on whether $r'<r_p$ or $r'\geqq r_p$), we can search for two consecutive jump points a_0, b_0 such that $a_0<r_p\leqq b_0$. The search amounts to $O(\log(\min(n,p)))$ evaluations of the function $M(r)$. This completes Phase 1, at the end of which we have the interval $(a_0,b_0]$ such that $k_{ij}(r)$, $(i,j)\in E$, is constant for all $r\in(a_0,b_0]$. The total effort during this phase uses $O(q\log n+n\log q)$ time where $q=\min(n,p)$. We note in passing that at this point, when we have $k_{ij}(r_p)$, $(i,j)\in E$, r_p can be found in $O(n^2)$ time using the method of [14] as described in § 1; this is because the evaluation of $M(r)$ takes $O(n)$ time and the method of [14] always leads to no more than the squared amount of time of the master algorithm used. We note that the number of centers on (i,j) is either $k_{ij}(r_p)$ or $k_{ij}(r_p)+1$ so that the value of $M(r)$ is computed $O(n)$ times.

2.3. Phase 2. When the second phase starts, we have an interval $(a_0,b_0]$ such that $a_0<r_p\leqq b_0$, over which the functions $k_{ij}(r)$ (for $(i,j)\in E$) are constant; we have to narrow this interval down until all the $k_{xy}(r)$'s become constant. We will describe the algorithm here in a recursive way.

Given is a tree T together with an interval $(\alpha,\beta]$ such that $\alpha<r_p\leqq\beta$ and the functions $k_{ij}(r)$ ($(i,j)\in E$) are known to be constant over $(\alpha,\beta]$. Our recursive routine in the present subsection will produce the following: A subinterval $(\alpha',\beta']$ ($\alpha\leqq\alpha'<r_p\leqq\beta'\leqq\beta$) will be found such that all the functions $k_{xy}(r)$ (for any vertices x, y of T) will be constant on $(\alpha',\beta']$.

Let c be a centroid of T, i.e., T can be represented as a union of two subtrees T_1 and T_2 whose only common vertex is c, such that each has at most $\frac{2}{3}$ of the vertices of T. Such a vertex can be found in linear time [10], [13], [16]. We call this partition a centroid decomposition [7], [16]. The decomposition may proceed into the subtrees and their components and so on, and that whole hierarchy (done in $O(\log n)$ phases) will be called a total centroid decomposition.

We first apply the routine recursively to the trees T_1 and T_2. We thus obtain an interval $(\alpha_0,\beta_0]$ ($\alpha_0<r_p\leqq\beta_0$) such that $k_{xy}(r)$ is constant on $(\alpha_0,\beta_0]$ whenever x and

y are in the same subtree (either T_1 or T_2). It takes linear time to find all the distances $d(x, c)$ and $d(c, y)$ $(x \in T_1, y \in T_2)$, and hence the constant value over $(\alpha_0, \beta_0]$ of $k_{xc}(r)$ for $x \in T_1$ and of $k_{cy}(r)$ for $y \in T_2$ is also assumed to be known. Moreover, $k_{xc}(r) + k_{cy}(r) \leqq k_{xy}(r) \leqq k_{xc}(r) + k_{cy}(r) + 1$. Thus, the function $k_{xy}(r)$ $(x \in T_1, y \in T_2)$ may have at most one jump in the interval $(\alpha_0, \beta_0]$, namely, when it jumps from $k_{xc}(r) + k_{cy}(r) + 1$ to $k_{xc}(r) + k_{cy}(r)$. This occurs at the value

$$\frac{1}{2} \cdot \frac{d(x, y)}{k_{xc}(r) + k_{cy}(r) + 1}.$$

Denote $a_x = \frac{1}{2}d(x, c)$, $b_y = \frac{1}{2}d(c, y)$, $c_x = k_{xc}(r)$ and $d_y = k_{cy}(r) + 1$. As a matter of fact, we now have to search for r_p within the set

$$R' = \left\{ \frac{a_x + b_y}{c_x + d_y} : x \in T_1, y \in T_2 \right\}.$$

In other words, we look for α', $\beta' \in R' \cup \{\alpha_0, \beta_0\}$ such that $\alpha_0 \leqq \alpha' < r_p \leqq \beta' \leqq \beta_0$ and $(\alpha', \beta') \cap R' = \varnothing$. It is explained in the Appendix how to perform this search in $O(n \log^2 n)$ time. It follows from the definition of the centroid that the total centroid decomposition consists of $O(\log n)$ phases. Therefore our routine runs in $O(n \log^3 n)$ time.

This implies that the continuous p-center problem is solvable by our algorithm in $O(n \log^3 n)$ time.

3. The weighted discrete case. As pointed out in the Introduction the weighted discrete case is equivalent to searching for the optimal value, denoted here by r^*, in the set $R^* = \{d(x, y)/(w_x^{-1} + w_y^{-1}) : x, y \in V\}$. The solution process here is quite similar to that of the continuous case. It is easy to see that an adaptation of Phase 2 to the present problem solves the problem in $O(n \log^3 n)$. However, a further improvement is yet possible.

All the intervertex distances $d(x, y)$ can be organized in the form $d(x, y) = d(x, c_i) + d(c_i, y)$ where c_i is a centroid in the total decomposition of our tree (see § 2.3 and [7], [16]). Let I denote the index set of these centroids c_i and let T_i^1 and T_i^2 denote the two subtrees given by the decomposition of a previous subtree as induced by c_i (see § 2.3). We then have to search for r^* in a collection of sets of the form

$$R_i = \left\{ \frac{d(x, c_i) + d(c_i, y)}{w_x^{-1} + w_y^{-1}} : x \in T_i^1, y \in T_i^2 \right\},$$

$i \in I$, where $\sum_{i \in I} |T_i^1| = O(n \log n)$ and $\sum_{i \in I} |T_i^2| = O(n \log n)$.

The search procedure employed here is similar to the two-stage scheme described in the Appendix. In the first stage we identify an interval $[s_1, t_1]$ such that $s_1 < r^* \leqq t_1$ and such that for each i, the linear order induced on the vertices of T_i^1 by the numbers $w_x^{-1} r - d(x, c_i)$ is independent of r provided $r \in [s_1, t_1]$. For each $i \in I$ we employ $|T_i^1|$ "processors" for sorting $\{w_x^{-1} r - d(x, c_i) : x \in T_i^1\}$. We use Valiant's [18] parallel sorting algorithm which takes $O(\log |T_i^1| \log \log |T_i^1|)$ time. Thus, $\sum |T_i^1| = O(n \log n)$ processors suffice for parallel sorting of each of the $|I|$ sequences $\{w_x^{-1} r - d(x, c_i) : x \in T_i^1\}$ in $O(\log n \log \log n)$ time.

As in the Appendix, a single step of the parallel sorting scheme gives $n \log n$ critical values for the parameter r. Given these $n \log n$ values, and an interval $(s_0, t_0]$ which contains r^*, we can in $O(n \log n)$ time (as in the Appendix) narrow down the interval so that it will still contain r^* but none of the above $n \log n$ critical values in

its interior. Hence, the total time for the first stage is $O((\sum_{i\in I}|T_i^1|+n\log n)\log n\log\log n)=O(n\log^2 n\log\log n)$.

The second stage is the same as Stage 2 of the Appendix, with the exception that here we use $\sum_{i\in I}|T_i^2|=O(n\log n)$ processors. However, the total effort for this stage still remains $O(n\log^2 n)$. Thus, the total effort involved in finding r^* is $O(n\log^2 n\log\log n)$.

We note that if the tree is a path connecting two vertices v_1 and v_n then r^* is an element of the set

$$R=\left\{\frac{d(x,v_1)-d(y,v_1)}{w_x^{-1}+w_y^{-1}}: x,y\in V\right\}.$$

Therefore, using the scheme described in the Appendix, r^* is found in $O(n\log^2 n)$ time.

4. NP-hardness of the continuous p-center problem. The result of the present section was not particularly hard to achieve even though it has not been previously proved to our knowledge. The NP-completeness of other versions of p-center problems was proved in [13].

In order to establish that the p-center problem is NP-hard on general graphs, we will reduce the minimum dominating set problem (see [9]) to the following problem: Given a graph $G=(V,E)$, all of whose edges are of unit length, find out whether there exists a set of p "centers" (anywhere on the edges of G) such that every point of G is within a distance of 2 from some center. We should remark that it may be necessary to locate centers in the interior of an edge in order to solve the latter problem (see Fig. 1).

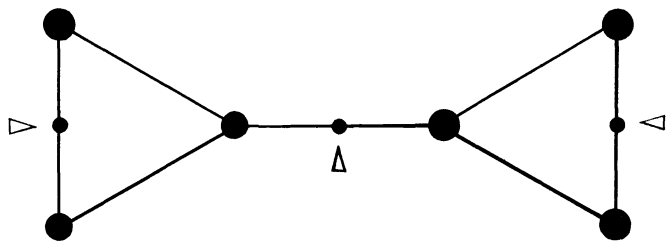


FIG. 1

The reduction is as follows. Suppose that $G=(V,E)$ is a graph for which we have to recognize whether there exists a dominating set of cardinality p. Let G' denote a graph obtained from G by adjoining one vertex v' per each vertex v of G, such that v' is adjacent only to v. All the edges of G' will be of unit length. We consider the problem of p centers on G'. Obviously, a dominating set for G will solve the center problem. Conversely, suppose that we have p centers on G' such that every vertex is at distance not greater than 2 from some center. Clearly, every vertex of the original graph G is within unit distance from some center. If we translate every center to one of its respective nearest vertices, then every vertex of G still has a center within unit distance from it. We thus have a dominating set of the appropriate cardinality.

Finally, we remark that since the dominating set problem is NP-complete even on bipartite planar graphs of maximum degree 3 (see [13, proof of Lemma 3.1]), then the above reduction implies NP-hardness of the continuous p-center problem even on bipartite planar graphs of maximum degree 4.

Appendix. Searching in $\{(a_i+b_j)/(c_i+d_j)\}$. In this appendix we describe how to search for the number r_p within a set of the form $S=\{(a_i+b_j)/(c_i+d_j): 1\leqq i,j\leqq n\}$. Thus there will be given $4n$ numbers, a_i, b_j, c_i, d_j $(1\leqq i,j\leqq n)$, and we will have to find two elements s, $t\in S$ such that $s<r_p\leqq t$ and no element of S is strictly between s and t.

We first note that the set S consists of the points of intersection of straight lines $y=(c_ix-a_i)+(d_jx-b_j)$ with the x-axis. The search will be conducted in two stages. During the first stage we will identify an interval $[s_1, t_1]$ such that $s_1<r_p\leqq t_1$ and such that the linear order induced on $\{1,\cdots,n\}$ by the numbers c_ix-a_i is independent of x provided $x\in[s_1, t_1]$. The rest of the work is done in Stage 2.

Stage 1. An equivalent description of Stage 1 is that we search for r_p among the points of intersection of lines $y=c_ix-a_i$ with each other. The method was introduced in [15]. It is based on Preparata's [17] parallel sorting scheme. This scheme employs $n\log n$ "processors" during $O(\log n)$ steps. Imagine that we sort the set $\{1,\cdots,n\}$ by the (c_ix-a_i)'s, where x is not known yet. Whenever a processor in Preparata's scheme has to compare some c_ix-a_i with c_jx-a_j, we will in our algorithm compute the critical value $x_{ij}=(a_i-a_j)/(c_i-c_j)$. Thus, a single step in Preparata's scheme gives rise to the production of $n\log n$ points of intersection of lines $y=c_ix-a_i$ with each other. Given these $n\log n$ points and an interval $(s_0, t_0]$ which contains r_p, we can in $O(n\log n)$ time narrow down the interval so that it will still contain r_p but no intersection point in its interior. This requires the finding of medians in sets of cardinalities $n\log n$, $\frac{1}{2}n\log n$, $\frac{1}{4}n\log n$, $\cdots$ plus $O(\log n)$ evaluations of $M(r)$. We then proceed to the next step in Preparata's scheme. We note that since the outcomes of the comparisons so far are independent of x in the updated interval, we can proceed with the sorting even though x is not specified. The effort per step is hence $O(n\log n)$ and the entire Stage 1 takes $O(n\log^2 n)$ time.

Stage 2. When the second stage starts we can assume without loss of generality that for $x\in[s_1, t_1]$ $c_ix-a_i\leqq c_{i+1}x-a_{i+1}$, $i=1,\cdots,n-1$. Let j $(1\leqq j\leqq n)$ be fixed and consider the set S_j of n lines $S_j=\{y=c_ix-a_i+d_jx-b_j, i=1,\cdots,n\}$. Since S_j is "sorted" over $[s_1, t_1]$, we can obviously find in $O(\log n)$ evaluations of $M(r)$ a subinterval $[s_1^j, t_1^j]$ such that $s_1^j<r_p\leqq t_1^j$, and such that no member of S_j intersects the x-axis in the interior of this interval. We will, however, work on the S_j's in parallel. Specifically, there will be $O(\log n)$ steps. During a typical step, the median of the remainder of every S_j is selected (in constant time) and its intersection point with the x-axis is computed. The set of these n points is then searched for r_p and the interval is updated accordingly. This enables us to discard a half from each S_j. Clearly a single step lasts $O(n\log n)$ time and the entire stage is carried out in $O(n\log^2 n)$ time.

At the end of the second stage we have the values $\{s_1^j\}$ and $\{t_1^j\}$, $j=1,\cdots,n$. Defining $s=\max_{1\leqq j\leqq n}\{s_1^j\}$ and $t=\min_{1\leqq j\leqq n}\{t_1^j\}$ we obtain $s<r_p\leqq t$, and no element of S is strictly between s and t.

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] R. CHANDRASEKARAN AND A. DAUGHETY, *Problems of location on trees*, Discussion Paper, 357, Center for Mathematical Studies in Economics and Management, Northwestern Univ., Evanston, IL, 1978.

[3] ———, *Location on tree networks: p-center and n-dispersion problems*, Math. Oper. Res., 6(1981), pp. 50–57.

[4] R. CHANDRASEKARAN AND A. TAMIR, *An $O((n\log p)^2)$ algorithm for the continuous p-center on a tree*, SIAM J. Alg. Disc. Meth., 1(1980), pp. 370–375.

[5] G. N. FREDERICKSON AND D. B. JOHNSON, *Generalized selection and ranking*, in Proc. 12th Annual ACM Symposium on Theory of Computing, Los Angeles, April 1980, Assoc. Comput. Mach., New York, 1980, pp. 420–428.

[6] ———, *Generating and searching sets induced by networks*, in Proc. 7th EATCS Colloquium on Automatic Languages and Programming, Noordwijkerhout, the Netherlands, July 1980, Lecture Notes in Computer Science, 85, Springer-Verlag, Berlin, 1980, pp. 221–233.

[7] ———, *Generating and searching sets for path selection and p-center location*, Computer Science Department, Pennsylvania State Univ., University Park, PA, August 1981.

[8] ———, *Finding k-th paths and p-centers by generating and searching good data structures*, J. Algorithms, to appear.

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

[10] A. J. GOLDMAN, *Optimal center location in simple networks*, Transportation Sci., 5 (1971), pp. 212–221.

[11] G. Y. HANDLER, *Minimax location of a facility in an undirected tree graph*, Transportation Sci., 7 (1973), pp. 287–293.

[12] ———, *Finding two-centers of a tree: The continuous case*, Transportation Sci., 12(1978), pp. 1–15.

[13] O. KARIV AND S. L. HAKIMI, *An algorithmic approach to network location problems, Part I. The p-centers*, SIAM J. Appl. Math, 37(1979), pp. 513–538.

[14] N. MEGIDDO, *Combinatorial optimization with rational objective function*, Math. Oper. Res., 4(1979), pp. 414–424.

[15] ———, *Applying parallel computation algorithms in the design of serial algorithms*, in Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science, 1981, IEEE Computer Society Press, Los Angeles, 1981, pp. 399–408; J. Assoc. Comput. Math., to appear.

[16] N. MEGIDDO, A. TAMIR, E. ZEMEL AND R. CHANDRASEKARAN, *An $O(n \log^2 n)$ algorithm for the k-th longest path in a tree with applications to location problems*, this Journal, 10(1981), pp. 328–337.

[17] F. P. PREPARATA, *New parallel-sorting schemes*, IEEE Trans. Comput., C-27(1978), pp. 669–673.

[18] L. G. VALIANT, *Parallelism in comparison problems*, this Journal, 4(1975), pp. 348–355.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0012 $01.25/0

LINEAR-TIME ALGORITHMS FOR LINEAR PROGRAMMING IN R^3 AND RELATED PROBLEMS*

NIMROD MEGIDDO†

Abstract. Linear-time algorithms for linear programming in R^2 and R^3 are presented. The methods used are applicable for other graphic and geometric problems as well as quadratic programming. For example, a linear-time algorithm is given for the classical problem of finding the smallest circle enclosing n given points in the plane; this disproves a conjecture by Shamos and Hoey [Proc. 16th IEEE Symposium on Foundations of Computer Science, 1975] that this problem requires $\Omega(n \log n)$ time. An immediate consequence of the main result is that the problem of linear separability is solvable in linear time. This corrects an error in Shamos and Hoey's paper, namely, that their $O(n \log n)$ algorithm for this problem in the plane was optimal. Also, a linear-time algorithm is given for the problem of finding the weighted center of a tree, and algorithms for other common location-theoretic problems are indicated. The results apply also to the problem of convex quadratic programming in three dimensions.

The results have already been extended to higher dimensions, and we know that *linear programming can be solved in linear time when the dimension is fixed.* This will be reported elsewhere; a preliminary version is available from the author.

Key words. linear programming, 1-center, weighted center, smallest circle, linear time, median, separability, quadratic programming

1. Introduction. The problem of finding the convex hull of n points in the plane has been studied by many authors, and its complexity is known to be $O(n \log n)$ not only in the plane but also in R^3 (Graham [G], Preparata and Hong [PH] and Yao [Y]). Several known problems in computational geometry, such as farthest points, smallest circle, extreme point, etc., are closely related to the problem of finding the convex hull of n points in the plane (Shamos [Sh], Shamos and Hoey [ShH] and Dobkin and Reiss [DR]). We have not found in these references an explicit statement about the complexity of linear programming in two and three dimensions. A closely related problem is the "separability" problem for which a statement of complexity was made. The separability problem is to separate two sets of n points in R^d by means of a hyperplane. Dobkin and Reiss [DR] report that this problem is solvable in $O(n \log n)$ time when $d \leqq 3$, referring to Preparata and Hong's work [PH]. Moreover, Shamos and Hoey solve the separability problem in R^2 in $O(n \log n)$ time and claim (erroneously) [ShH, p. 224] their algorithm to be optimal. The truth is that the separability problem in R^d is obviously solvable by linear programming in d variables. In particular, it follows from the results of the present paper that it can be solved in $O(n)$ time when $d \leqq 3$.

We may learn about the state-of-art of the complexity of linear programming in R^2 by considering the "extreme point" problem, i.e., the problem of determining whether a given point P_0 in R^2 is a convex combination of n given points $P_1, \cdots, P_n$ in R^2. Dobkin and Reiss [DR, p. 17] state without proof or reference that this problem (in R^2) is solvable in linear time. This statement is rather obvious since the extreme point problem in the plane can be modeled as a problem of finding a straight line which crosses through P_0 and has all the points $P_1, \cdots, P_n$ lying on one side of it. The latter, however, amounts to linear programming in R^1 which is trivial. The same

* Received by the editors February 9, 1982, and in revised form November 15, 1982. This research was partially supported by the National Science Foundation under grants ECS-8121741 and ECS-8218181, at Northwestern University.

† Department of Statistics, Tel Aviv University, Tel Aviv, Israel. Currently visiting Department of Computer Science, Stanford University, Stanford, California 94305.

observation implies that the separability problem in R^d $(d \geqq 2)$ can be solved by linear programming in $d-1$ variables so that, in view of the present paper, it is solvable in linear time in R^4.

Another problem, related to linear programming in three variables, which we solve in $O(n)$ time, is that of finding the smallest circle enclosing n given points in the plane. Shamos and Hoey [ShH] solve this problem in $O(n \log n)$ time, improving the previously known bound of $O(n^3)$ very significantly. A seemingly related problem, namely, that of finding the largest empty circle, was shown to require $\Omega(n \log n)$ time, and that led Shamos and Hoey to the (wrong) conjecture that $\Omega(n \log n)$ was also a lower bound for the smallest enclosing circle problem. They were convinced that the so-called Voronoi diagram would always provide optimal algorithms, so they stated [ShH, p. 231]: "... the proper attack on a geometry problem is to construct those geometric entities that delineate the problem ...". Our results prove that this is not always the case, since the construction of the Voronoi diagram does require $\Omega(n \log n)$ time, while the smallest enclosing circle can be found in $O(n)$ time.

The problems discussed in this paper are presented in order of increasing difficulty. We start with linear programming in R^2 which is a subroutine for the three-dimensional problem. The two-dimensional case is discussed in § 2. In § 3 we consider the problem of the weighted center of a tree. The latter is more complicated than linear programming in two variables but yet does not involve the difficulties which arise in the three-dimensional case. The best known bound for it was $O(n \log n)$ [KH]. The problem of the smallest circle enclosing n points in the plane, which is discussed in § 4, is more complicated than linear programming in the plane. It is in fact a three-dimensional problem in a certain sense, and the algorithm which we present for it leads to the design of a linear-time algorithm for linear programming in R^3. In § 4 we also point out how our results apply to other location-theoretic problems in the plane. The problem of linear programming in three variables is discussed in § 5. Our linear programming algorithm for R^3 can easily be extended to solve convex quadratic programming problems in R^3 in $O(n)$ time. The latter is also discussed in § 5. In the Appendix we include an efficient algorithm for the extreme-point problem in the plane (discussed earlier in this Introduction) which is a routine for solving the smallest circle problem.

2. Linear programming in the plane.

2.1. Preliminaries. The linear programming problem in the plane can be stated as follows:

$$\begin{aligned} &\underset{x_1,x_2}{\text{minimize}} && c_1x_1+c_2x_2 \\ &\text{s.t.} && a_{i1}x_1+a_{i2}x_2 \geqq \beta_i \qquad (i=1,\cdots,n). \end{aligned}$$

It will be convenient for us to deal with the problem in an equivalent form, which can be obtained from the original one in $O(n)$ time:

$$\begin{aligned} &\underset{x,y}{\text{minimize}} && y \\ &\text{s.t.} && y \geqq a_ix+b_i \qquad (i \in I_1), \\ &&& y \leqq a_ix+b_i \qquad (i \in I_2), \\ &&& a \leqq x \leqq b \end{aligned}$$

where $|I_1|+|I_2| \leqq n$ and $-\infty \leqq a, b \leqq \infty$. We also define the following functions:

$$g(x) = \max\{a_i x + b_i : i \in I_1\},$$

$$h(x) = \min\{a_i x + b_i : i \in I_2\}.$$

Obviously, both of these functions are piecewise linear, and g is convex while h is concave. A number x, $a \leqq x \leqq b$, is said to be feasible if $g(x) \leqq h(x)$. We can pose our problem also in a one-dimensional form:

$$\begin{aligned} &\text{minimize} && g(x) \\ &\text{s.t.} && g(x) \leqq h(x), \\ & && a \leqq x \leqq b. \end{aligned}$$

Our algorithm works as follows. We test values of x in a fashion resembling binary search. Each test runs in linear time and enables us to drop at least a quarter of the constraints of the problem. We first have to describe our test in detail.

2.2. Testing a value of x. Given any value x' of x $(a \leqq x' \leqq b)$, we test the following: (i) Is x' feasible? (ii) If x' is not feasible then if there are any feasible values of x then they must lie on one side of x'; our test either determines that side or concludes that no feasible values exist; (iii) If x' is feasible then our test will recognize whether x' is also optimal and if not then it will tell us on what side of x' the minimum lies.

We start with the case of infeasible x'. In other words, $g(x') > h(x')$. Consider the function $f(x) = g(x) - h(x)$. This function is convex so the values of x such that $f(x) \leqq 0$ (if there are any) all lie on one side of x'. In order to tell the correct side we look at the one-sided derivatives of f at x'. This is done as follows (see Fig. 1). Define

$$s_g = \min\{a_i : i \in I_1,\ a_i x' + b_i = g(x')\},$$

$$S_g = \max\{a_i : i \in I_1,\ a_i x' + b_i = g(x')\},$$

$$s_h = \min\{a_i : i \in I_2,\ a_i x' + b_i = h(x')\},$$

$$S_h = \max\{a_i : i \in I_2,\ a_i x' + b_i = h(x')\}.$$

If $s_g > S_h$ then $f(x)$ is ascending at x' so that a feasible x can only be smaller than x'. Analogously, if $S_g < s_h$, then $f(x)$ is descending at x' so that a feasible x can only be larger than x'. The remaining case is when $s_g - S_h \leqq 0 \leqq S_g - s_h$. In this case f attains its minimum at x', i.e., there are no feasible values of x.

Consider now the case when x' is found to be feasible. We are interested in finding out on what side of x' the optimal solution lines. Assume, first, that $g(x') < h(x')$. Here the analysis is quite simple. We need to look only at the numbers s_g and S_g. If $s_g > 0$ then an optimal solution (denote it by x^*) must satisfy $x^* < x'$. Analogously, if $S_g < 0$ then $x^* < x'$. Otherwise, $s_g \leqq 0 \leqq S_g$ and x' itself is a minimum of g. If $g(x') = h(x')$ then the situation could be one of the following: (i) If $s_g > 0$ and $s_g \geqq S_h$ then $x^* < x'$. (ii) If $S_g < 0$ and $S_g \leqq s_h$ then $x^* > x'$. Otherwise x' itself is a minimum of $g(x)$ under the constraint $g(x) \leqq h(x)$.

In summary, if x' is any value in $[a, b]$ then in linear time we can either find that the problem is infeasible, recognize that x' itself is an optimal solution, or decide that the rest of the computation may be confined to one of the subintervals $[a, x']$, $[x', b]$.

2.3. The algorithm. We start the procedure by arranging the elements of I_1 in disjoint pairs and, similarly, those of I_2 in disjoint pairs (a single element from either

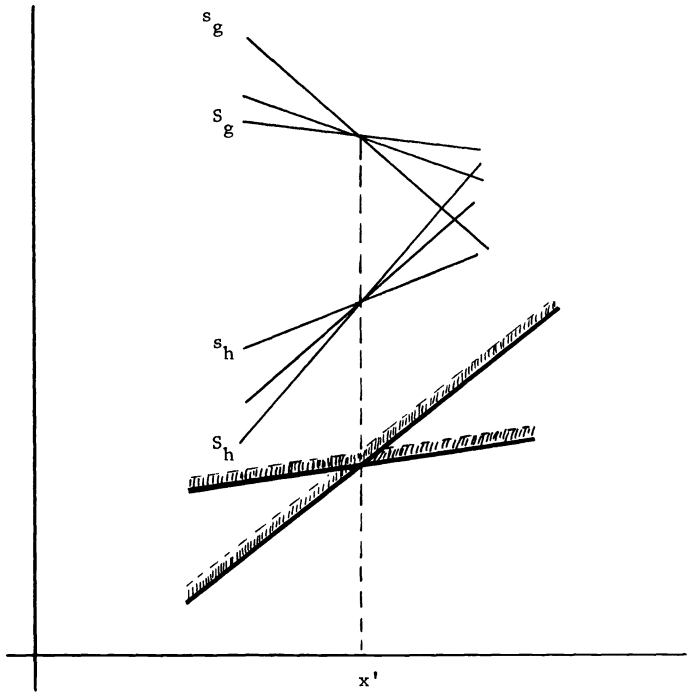


FIG. 1

set may be left unmatched). Consider, for example, a pair $i, j \in I_1$. If $a_i = a_j$ then, obviously, one of the constraints $y \geqq a_i x + b_i$, $y \geqq a_j x + b_j$ may be dropped without affecting the optimal solution of the problem. Otherwise, $a_i \neq a_j$ and the number $x_{ij} = (b_i - b_j)/(a_j - a_i)$ (i.e., the solution of the equation $a_i x + b_i = a_j x + b_j$) has the property that if x is confined to an interval which does not contain x_{ij} in its interior then, again, one of the two constraints is redundant and may be dropped. A similar observation is of course valid for a pair of constraints of the form $y \leqq a_i x + b_i$.

Consider all the pairs that have been formed. First, drop every constraint which is redundant according to the previous observation, i.e., either because of an inequality $a_i = a_j$ or because of a relation $x_{ij} \notin (a, b)$. We now consider only those pairs i, j (that have been formed) for which $a_i \neq a_j$ and $a < x_{ij} < b$. The next step is to find the median x_m of the set of x_{ij}'s. This can be done in linear time (see [AHU]). We then test the value x_m along the lines described in § 2.2, i.e., we either recognize that our problem is infeasible, find out that x_m is an optimal solution for our problem, or deduce that the interval $[a, b]$ may be redefined (the new interval being either $[a, x_m]$ or $[x_m, b]$). In the first two cases our task is finished. In the latter case we do the following. At least half of the critical values x_{ij} (as defined by the original pairs) will not be in the interior of the new interval. We will thus be able to drop one constraint per each such pair which is at least a quarter of the set of constraints (including those that have been dropped prior to the evaluation of x_m). We are thus left with a linear programming problem in the plane with at most $[3n/4]$ constraints. This implies that the runtime time (n) of our algorithm on an n-constraint problem satisfies time $(n) \leqq C \cdot n +$ time $(3n/4)$ and hence time $(n) = O(n)$. Of course, when n is small (e.g., $n \leqq 4$) the problem will be solved directly.

3. The weighted center of a tree.

3.1. Introduction. The weighted center of a tree is defined as follows. Given is a tree $T=(V, E)$ with n vertices. A nonnegative length d_{ij} is associated with every edge (i, j) and a nonnegative weight w_i is associated with every vertex i. An edge (i, j) is identified with a line segment of length d_{ij} so that we can talk about any "point" on the edge (i, j); formally, a point $x=(i, j; t)$ is characterized by being located at a distance of t from i and $d_{ij}-t$ from j. Thus the distance $d(x, y)$ between any two points x, y on the tree is well defined, namely, it is the length of the unique path from x to y. The weighted center of T is a point x which minimizes the function $r(x)=\max\{w_i d(x, i): i \in V\}$. The center is unique unless all the weights equal zero. A related problem is to find a vertex j which minimizes the function $r(x)$, i.e., x is restricted to be a vertex of the tree.

The best known algorithm for the weighted center problem is an $O(n \log n)$ procedure by Kariv and Hakimi [KH]. Other algorithms which run in $O(n^2)$ time have been given in Dearing and Francis [DF], Levin [L] and Hakimi, Schmeichel and Pierce [HSP].

The unweighted case, namely, when all the weights w_i are equal, is much easier and is solvable in $O(n)$ time (see Handler and Mirchandani [HM]). We will present here a linear-time algorithm for the general weighted case.

3.2. Preliminaries. The function $r(x)$ is convex on every simple path of the tree. Specifically, if P is a simple path and i is any vertex, then consider the vertex j which is on the path P and is nearest to i. The vertex j partitions the path into two pieces over each of which the function $g_i(x)=d(x, i)$ is linear and increasing as we move away from j. Thus, $g_i(x)$ is piecewise linear on P (with at most two pieces) and convex. The function $r(x)$ is hence piecewise linear and convex, being the maximum of convex functions.

Let x be any point on the tree. A vertex j $(j \neq x)$ is said to be adjacent to x if x lies on an edge which is incident upon j (x may itself be a vertex but then it is not considered adjacent to itself). Let $V_j(x)$ denote the set of vertices i such that j lies on the simple path from x to i (see Fig. 2). Let $T_j(x)$ denote the subtree which is spanned by the set $V_j(x) \cup \{x\}$; in particular, this subtree contains an edge (j, x) which is just a subsegment of (j, k) for some $k \in V$. Consider the function $r_j(x)=\max\{w_i d(x, i): i \in V_j(x)\}$. Clearly, $r_j(x)$ decreases as we move from x in the direction of j. Let $j_1, \cdots, j_l$ be all the vertices adjacent to x ($l=2$ if x is interior to some edge). Obviously, $r(x)=\max\{r_{j_1}(x), \cdots, r_{j_l}(x)\}$. Moreover, if the maximum is attained at more than one index then x is a local minimum of the function r, and hence it must be the center since r is convex. On the other hand, if the maximum is attained at a unique index, then the center must lie in the corresponding subtree. Formally, if

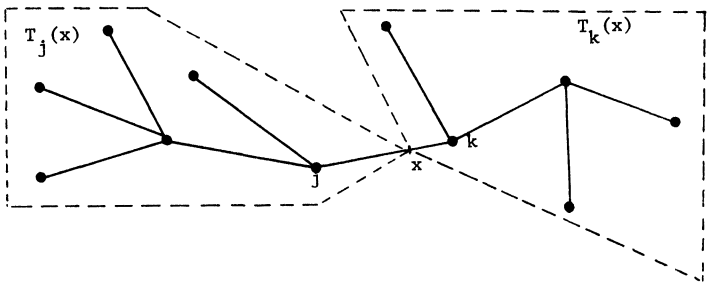


FIG. 2

$r_j(x) > r_k(x)$ for every vertex k $(k \neq j)$ adjacent to x then the center lies in $T_j(x)$. In summary, given any point x, we can easily tell (in $O(n)$ time) in which of the subtrees $T_j(x)$ the center lies.

Kariv and Hakimi [KH] based their procedure on the above observations. Given that the center lies in a subtree T' they "test" the centroid x of T' to find out which of the subtrees of T', rooted at x, contains the center. Since the centroid defines subtrees whose sizes are at most half the size of T', the process terminates within $O(\log n)$ tests and hence runs in $O(n \log n)$ time. The improvement we suggest here is in reducing the cost of a test. We will also perform $O(\log n)$ tests; however, the cost of each test will be no more than three quarters the cost of the preceding one.

3.3. The linear-time algorithm. Let c denote the centroid of T, i.e., c is a vertex such that for every adjacent vertex j, $|V_j(c)| \leqq n/2$ (where n is the number of vertices of T). We note that c can be found in $O(n)$ time [HM]. This is accomplished by a walk over the vertices, always moving in the direction in which the number of vertices, in the subtree entered into, is being maximized. Now assume that c is known.

First, evaluate $r_j(c)$ for each adjacent vertex j. This amounts to finding all the distances $d(c, i)$ and hence can also be carried out in $O(n)$ time. If there are two vertices j_1, j_2 $(j_1 \neq j_2)$ adjacent to c, such that $r_{j_1}(c) = r_{j_2}(c) = r(c)$, then c itself is the center and we terminate. Thus, let us now assume that j is adjacent to c and $r_j(c) > r_k(c)$ for every other adjacent vertex k. We now know that the center lies in $T_j(c)$.

If u is a vertex not in $V_j(c)$ and if x is in $T_j(c)$ at a distance of t from c, then $d(u, x) = d(u, c) + t$. If u and v are vertices not in $V_j(c)$ then by solving (for t) the equation $w_u(d(u, c) + t) = w_v(d(v, c) + t)$ we can tell the following: Assume, without loss of generality, that $w_u d(u, c) \geqq w_v d(v, c)$. There is a value t_{uv} $(0 \leqq t_{uv} \leqq \infty)$ such that, for every x in $T_j(c)$ at a distance of t from c, $w_u d(u, x) \geqq w_v d(v, x)$ if and only if $0 \leqq t \leqq t_{uv}$. Thus, if we knew that the center lay at a distance smaller than t_{uv} from c then we could disregard the vertex v from that point and on in the process of finding the center. Similarly, the vertex u could be eliminated if we knew that the center lay at a distance greater than t_{uv} from c. We will show below how to efficiently exploit this observation. However, we first need to show how to recognize whether or not the center lies within a distance of t from x, where x is any leaf vertex and t is any positive real number; our discussion applies to the tree $T_j(c)$ where $x = c$ is a leaf and t is some value of the type t_{uv}, derived from data which are external relative to the tree $T_j(c)$.

Given a leaf x and a positive real number t, we can (in linear time) find all the points $y_1, \cdots, y_l$ such that $d(x, y_\nu) = t$, $\nu = 1, \cdots, l$. This is done as follows. First, evaluate all the distances $d(x, i)$. Now note that every edge (i, j), such that $d(x, i) \leqq t \leqq d(x, j)$, contains a unique point y_ν at a distance of $t - d(x, i)$ from i, and hence $d(x, y_\nu) = t$. The set of all points z such that $d(x, z) \geqq t$ can be represented as a union of subtrees rooted at the points $y_1, \cdots, y_l$ (each y_ν may contribute several such subtrees). Let these subtrees be simply denoted by $T_1, \cdots, T_m$ and let their roots be denoted $u_1, \cdots, u_m$ $(\{u_1, \cdots, u_m\} \subset \{y_1, \cdots, y_l\})$. Let V_i denote the set of vertices of T_i except for u_i (see Fig. 3). Define $R_i(x) = \max\{w_k d(x, k): k \in V_i\}$. Since the sets V_i are pairwise disjoint, it follows that we can evaluate all the quantities $R_i(u_i)$, $i = 1, \cdots, m$, in $O(n)$ time. Let $R = \max\{R_i(u_i): i = 1, \cdots, m\}$. First, if $R_i(u_i) < R$ then the center is certainly not in T_i. Similarly, if $R_{i_1}(u_{i_1}) = R_{i_2}(u_{i_2}) = R$ for some $i_1 \neq i_2$, then the center cannot lie inside any T_i, $i = 1, \cdots, m$. The remaining case is when there is a unique i $(1 \leqq i \leqq m)$ such that $R_i(u_i) = R$. In this case the center may lie in T_i. However, this can be recognized by evaluating the functions $r_j(y)$, where j is any

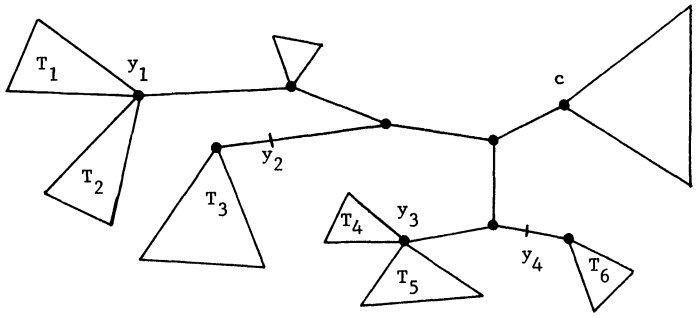


FIG. 3

vertex adjacent to u_i and $y = u_i$. All this again requires only $O(n)$ time. In summary, we can decide whether or not the center is within a distance of t from x (equivalently, whether or not it lies in one of the T_i's) in $O(n)$ time.

Returning to the original tree T, its centroid c and the subtree $T_j(c)$ which is known to contain the center, we now do the following. Arrange the vertices outside $T_j(c)$ in disjoint pairs $(u_1, v_1), (u_2, v_2), \cdots, (u_s, v_s)$ (leaving one out if there is an odd number of them). Note that there will be at least $[n/4]-1$ such pairs since at most $n/2$ vertices are in $T_j(c)$. For every such pair (u, v) consider the equation $w_u(d(u, c)+t) = w_v(d(v, c)+t)$, assuming, without loss of generality, that $w_u d(u, c) \geqq w_v d(v, c)$. If $w_u \geqq w_v$ then the vertex v is "discarded"; otherwise, let $t_{uv} = (w_u d(u, c) - w_v d(v, c))/(w_v - w_u)$.

Having calculated the values $t_{u_i v_i}$ (for all pairs from which no vertex was discarded), we now find the median of these values. This is done in $O(n)$ time (see [AHU]). Let the median be denoted by t_m. We now check whether the center lies (in $T_j(c)$) within a distance of t_m from c. This can be carried out in $O(n)$ time as we have already seen. Suppose, for example, we find that the center indeed lies within a distance of t_m from c. Consider a pair (u, v) such that $t_{uv} \geqq t_m$. It follows that wherever the center x^* lies (provided it is in $T_j(c)$ at a distance of no more than t_m from c) it must be true that $w_u d(u, x^*) \geqq w_v d(v, x^*)$. Thus, the vertex v is "dominated" by u in the sense that if the maximum weighted distance from the center is determined by v then it is also determined by u. Hence, we can safely discard the vertex v in this case. Similarly, if x^* is known to be at a distance greater than t_m from c, then from pairs (u, v), such that $t_{uv} \leqq t_m$, we can discard the vertex u.

It follows that one vertex is discarded from approximately half the pairs. In other words, we will discard approximately $\frac{1}{8}$ of the vertices of the tree. At this point we have reduced our problem to the weighted center problem on a tree T' which is defined as follows. For each vertex u not in $T_j(c)$, which has not been discarded, form an edge (c, u) and let its length be precisely $d(c, u)$. Also, let w_u be the same as in T. Adjoin all these edges to the tree $t_j(c)$ and call the new tree T'. Since $n/8$ vertices have been discarded, it follows that the run time, time(n), for a tree of n vertices, satisfies time $(n) \leqq$ time $(7n/8) + Cn$, which implies time $(n) = O(n)$.

The discrete problem of finding a vertex which minimizes the function $r(x)$ can now be solved. It follows from the convexity of the function $r(x)$ that the vertex minimizing $r(x)$ is either identical with or adjacent to the point at which $r(x)$ has its global minimum. Thus, by finding this global minimum we obtain at most two vertices (endpoints of an edge), one of which is minimizing $r(x)$ relative to the set of vertices.

4. Smallest circle enclosing n points.

4.1. Introduction. In the present section we shall deal with the classic problem of finding the smallest circle enclosing n given points (a_i, b_i), $i = 1, \cdots, n$, in the Euclidean plane. In the language of location theory this is the (unweighted) Euclidean 1-center problem in the plane. Formally, we are looking for a point (x, y) so as to minimize $\text{Max}\{((x-a_i)^2+(y-b_i)^2)^{1/2}: 1 \leqq i \leqq n\}$. Thus, the point (x, y) is an optimal location for a facility if we wish to minimize the largest distance that a customer would have to travel from his residence (in one of the given points (a_i, b_i)) to the facility.

The smallest enclosing circle problem has a long history. It was posed by Sylvester [Sy1] in 1857 and different solutions have been suggested in Sylvester [Sy2], Rademacher and Toeplitz [RT], Courant and Robbins [CR], Francis [F], Smallwood [Sm], Francis and White [FW], Nair and Chandrasekaran [NC], Elzinga and Hearn [EH], and finally, Shamos and Hoey [ShH]. Shamos and Hoey's algorithm runs in $O(n \log n)$ time and is the only one which has been proved to run in $o(n^3)$ time. It is based on constructing the so-called "farthest point Voronoi diagram" which we review below. This powerful structure is very useful for solving a number of computational geometric problems and its construction requires $\Omega(n \log n)$ time. This led Shamos and Hoey to the (wrong) conjecture that the smallest enclosing circle problem also had a lower bound of $\Omega(n \log n)$ [ShH, p. 154]. We shall present here a linear-time algorithm for this problem.

The diagram is a partition of the plane into regions V_j where a point (x, y) is in V_j if and only if the point (a_j, b_j) is farthest from (x, y) among the points (a_i, b_i), $i = 1, \cdots, n$. These regions are either empty or unbounded polyhedral sets. The construction of the diagram also yields the vertices of the polytope $\pi = \text{convex hull}\{(a_1, b_1), \cdots, (a_n, b_n)\}$ in their cyclic ordering on the boundary of π. Once the boundary is known, it takes $O(n)$ time to find the two farthest points. These two points define a circle whose diameter equals the distance between them. If the entire π is contained in this circle then this is the smallest possible circle. Otherwise, the smallest enclosing circle is centered at a point where some three regions V_i, V_j, V_k meet, i.e., the circle is defined by the points (a_i, b_i), (a_j, b_j), (a_k, b_k). It can be shown that there are at most $n-2$ such points in the diagram (relying on the fact that, as a graph, the farthest-point Voronoi diagram has no circuits) and the distances from such points to their respective defining points (a_i, b_i) can be produced during the construction of the diagram. It thus takes $O(n)$ time to find the center of the smallest enclosing circle once the diagram has been constructed.

4.2. A constrained version of the smallest circle problem. We will first develop an algorithm for a constrained problem, namely, where the center of the enclosing circle is forced to lie on a given straight line. For simplicity of presentation assume this line is the x-axis. Furthermore, at the end of the computation in this constrained problem we will be able to tell on which side of the straight line the unconstrained center lies. This will play an important role in the solution of the unconstrained problem.

Consider the problem of minimizing $g(x) = \max\{(x-a_i)^2+b_i^2: 1 \leqq i \leqq n\}$. Let i, j be any two distinct indices and consider the equation $(x-a_i)^2+b_i^2 = (x-a_j)^2+b_j^2$. This is in fact a linear equation: $-2a_ix+a_i^2+b_i^2 = -2a_jx+a_j^2+b_j^2$. If $a_i = a_j$ then (assuming, for example, $b_i^2 \leqq b_j^2$) we may drop the function $(x-a_i)^2+b_i^2$ from the definition of g. If $a_i \neq a_j$ then there is a critical value $x_{ij} = (a_j^2-a_i^2+b_j^2-b_i^2)/2(a_j-a_i)$ such that (assuming $a_j > a_i$) $(x-a_i)^2+b_i^2 \geqq (x-a_j)^2+b_j^2$ if and only if $x \geqq x_{ij}$.

The algorithm for the constrained center problem works as follows: Consider the pairs $(1, 2)$, $(3, 4)$, $\cdots$. For each pair $(i, i+1)$ (i odd), such that $a_i = a_{i+1}$, drop one of

the functions as explained above. Compute the critical values $x_{i,i+1}$. Find the median x_m of the values $x_{i,i+1}$ by any linear-time median-finding algorithm (see [AHU]). Let x^* denote the minimizer of g. Compute $g(x_m)$. We shall now discuss the question of recognizing whether $x_m < x^*$, $x_m = x^*$ or $x_m > x^*$. Let $I = \{i: (x_m - a_i)^2 + b_i^2 = g(x_m)\}$. If $x_m < a_i$ for every $i \in I$ then $x_m < x^*$. If $x_m > a_i$ for every $i \in I$ then $x_m > x^*$; otherwise, $x_m = x^*$. Knowing either that $x^* < x_m$ or that $x^* > x_m$, we can discard a quarter of our functions as follows. Assume for example $x^* < x_m$. We have at least half of our critical values $x_{i,i+1}$, greater than x^*. If $x_{i,i+1} > x_m$ then, since $(x - a_i)^2 + b_i^2 \geqq (x - a_{i+1})^2 + b_{i+1}^2$ if and only if $x \geqq x_{i,i+1}$, we may discard the function $(x - a_i)^2 + b_i^2$. This is because $(x^* - a_i)^2 + b_i^2 \leqq (x^* - a_{i+1})^2 + b_{i+1}^2$. Thus for at least half the pairs we can discard one function per pair. This implies that at least one quarter of the functions are dropped at the end of this stage. The linearity of the run time follows.

We now address the question of recognizing on which side of the straight line the unconstrained center lies. First, note that the function $f(x, y) = \max\{(x - a_i)^2 + (y - b_i)^2: i \leqq i \leqq n\}$ is convex. It is essential to note that f is convex, not only in each variable, but also as a function of two variables. This also implies that the function $h(y) = \min_x f(x, y)$ is convex. By minimizing $g(x)$ we in fact evaluate $h(0)$. The y-coordinate of the unconstrained center is precisely where $h(y)$ attains its minimum. Denote this value by y^c. Since $h(y)$ is convex we can tell the sign of y^c simply by looking in the neighborhood of $y = 0$. Thus, let $(x^*, 0)$ be the constrained center we have found. Let $I = \{i: (x^* - a_i)^2 + b_i^2 = g(x^*)\}$. Obviously, if $I = \{i\}$ then $x^* = a_i$ and y^c has the sign of b_i. If $I = \{i, j\}$ then $(x^*, 0)$ lies on the perpendicular bisector of the line segment $[(a_i, b_i), (a_j, b_j)]$. Obviously, y^c has the sign of the y-coordinate of the midpoint of this segment, i.e., $\frac{1}{2}(b_i + b_j)$. In general, all the points (a_i, b_i), $i \in I$ lie on a circle centered at $(x^*, 0)$. If $(x^*, 0)$ is in the convex hull of these points then $y^c = 0$. Otherwise, there exist two points (a_i, b_i), (a_j, b_j), $(i, j \in I)$ such that $f(x, y)$ decreases as we move from $(x^*, 0)$ in the direction of the midpoint of the line segment $[(a_i, b_i), (a_j, b_j)]$ (i.e., along the perpendicular bisector of that segment; see Fig. 4). In this case y^c has the sign of $\frac{1}{2}(b_i + b_j)$. It should be noticed that the determination of these two points or the recognition that $(x^*, 0)$ is in the convex hull can be carried out in linear time with the aid of linear programming in the plane (see § 2); a more straightforward method is given in the Appendix.

In summary, given any straight line in the plane, we can in $O(n)$ time determine on which side of the line the center of the smallest enclosing circle lies. Moreover, if

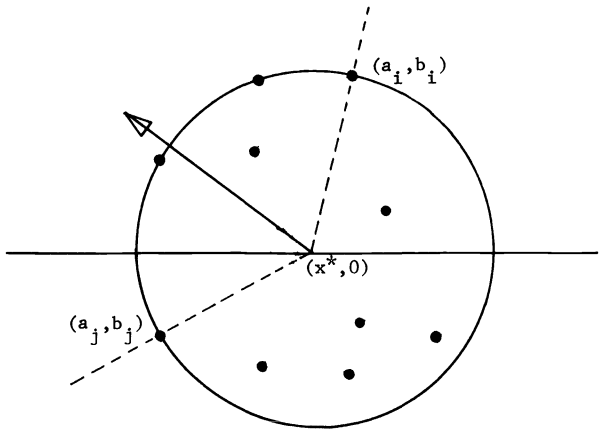


FIG. 4

this center happens to lie on the line then we discover its exact location during the procedure.

4.3. The $O(n)$ algorithm for the smallest circle. We shall now utilize the result of the preceding section for finding the center of the unconstrained problem. We start by producing the perpendicular bisectors of the line segments $[(a_{2i-1}, b_{2i-1}), (a_{2i}, b_{2i})]$, $i = 1, \cdots, [n/2]$. Denote them by L_i. Consider the angles α $(-\pi/2 \leqq \alpha < \pi/2)$ which these lines form with the positive direction of the x-axis. Let α_m denote the median of these angles. Consider the linear transformation that takes the x-axis to the line $y = \alpha_m x$ and leaves the y-axis fixed. By applying this transformation we can have at least half of our lines with nonnegative angles and at least half with nonpositive angles. It is obvious that this can be achieved in linear time.

The next step is to form pairs of lines (L_i, L_j) so that each pair has one line with nonnegative angle and one line with nonpositive angle; the pairs are disjoint. Thus, there will be $[n/4]$ such pairs. For each pair (L_i, L_j) define a value y_{ij} as follows. If L_i and L_j are parallel to the x-axis then let y_{ij} be the mean of their constant y-coordinates. Otherwise, they must intersect; let (x_{ij}, y_{ij}) denote their point of intersection. Now, let y_m denote the median of the $[n/4]$ values y_{ij}. The value y_m can be found in linear time. At this point we test on what side of the straight line $y = y_m$ the center must lie. This test runs in $O(n)$ time as we have shown in § 4.2. If the center lies on the line $y = y_m$ then we are done. Thus, suppose it does not lie on the line and assume, without loss of generality, that it lies underneath this line. Consider a pair L_i, L_j of parallel lines such that $y_{ij} \geqq y_m$. At least one of these lines lies above the line $y = y_m$. Suppose it is the line L_i (which is the perpendicular bisector of the line segment $[(a_{2i-1}, b_{2i-1}), (a_{2i}, b_{2i})]$). We can now drop one of the two defining points, namely the one which lies underneath the line L_i, since the other point is farther from the center. However, in general the lines L_i, L_j are not expected to be parallel.

Consider now the set of all pairs (L_i, L_j) of nonparallel lines for which $y_{ij} \geqq y_m$. We find the median x_m of the x_{ij}'s corresponding to such pairs. Like in the case of the y-coordinate, we now test on what side of the line $x = x_m$ the center of the smallest enclosing circle must lie. Suppose, for example, it lies to the left of this line. Consider pairs (i, j) such that $x_{ij} \geqq x_m$ and $y_{ij} \geqq y_m$. One of the lines, say L_i, forms a nonpositive angle with the positive direction of the x-axis. It follows (see Fig. 5) that one of the

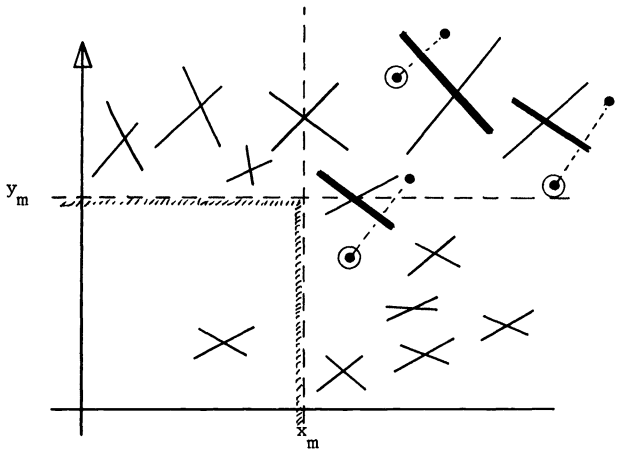


FIG. 5

points defining L_i, namely, the one which lies "southwest" of it, can be dropped since the other point will be at least as far from the center.

It follows that during this process we drop one point per pair for at least a quarter of our pairs of lines. In other words, at least $[n/16]$ points will be dropped with an $O(n)$ effort. It thus follows that the entire process runs in linear time.

4.4. A remark on other planar location problems. An analogous problem is the rectilinear 1-center problem in the plane for which linear-time algorithms are known [FW]. However, the weighted rectilinear problem, i.e.,

$$\underset{x,y}{\text{minimize}} \max \{w_i(|x-a_i|+|y-b_i|): i=1,\cdots,n\}$$

(where (a_i, b_i) are given points and w_i are given positive weights) can now be solved in $O(n)$ time by our methods in the present paper. The previously known bound was $O(n \log n)$ and followed from separating the planar problem into two one-dimensional problems. The one-dimensional problem is a special case of the weighted center problem of a tree, provided the numbers are sorted.

A much more complicated problem is the weighted Euclidean 1-center problem. The best known bound for this problem used to be $O(n^3)$ [EH], [DW], [CP]. In a recent paper the author [M2] presented an algorithm which required $O(n(\log n)^3(\log \log n)^2)$ time. The methods presented in the present paper combined with those of [M1], [M2] can yield an $O(n(\log n)^2)$ algorithm for the weighted Euclidean 1-center problem. The details will be given elsewhere.

5. Linear programming in R^3.

5.1. Preliminaries. In this section we will be dealing with the following problem:

$$\begin{aligned} &\underset{x_1,x_2,x_3}{\text{minimize}} && \gamma_1 x_1+\gamma_2 x_2+\gamma_3 x_3 \\ &\text{s.t.} && a_{i1}x_1+a_{i2}x_2+a_{i3}x_3 \geqq \beta_i \qquad (i=1,\cdots,n). \end{aligned}$$

We first transform the problem into the following form (in $O(n)$ time):

$$\begin{aligned} &\underset{x,y,z}{\text{minimize}} && z \\ &\text{s.t.} && z \geqq a_i x+b_i y+c_i \quad (i\in I_1), \\ &&& z \leqq a_i x+b_i y+c_i \quad (i\in I_2), \\ &&& a_i x+b_i y+c_i \leqq 0 \quad (i\in I_3) \end{aligned}$$

where $|I_1|+|I_2|+|I_3|=n$. We define the following functions:

$$\begin{aligned} g(x,y) &= \max\{a_i x+b_i y+c_i: i\in I_1\}, \\ h(x,y) &= \min\{a_i x+b_i y+c_i: i\in I_2\}, \\ e(x,y) &= \max\{a_i x+b_i y+c_i: i\in I_3\} \end{aligned}$$

which are all piecewise linear, g and e being convex while h is concave. Let also $f(x,y)=\max\{g(x,y)-h(x,y), e(x,y)\}$. Note that our problem is equivalent to the following:

$$\begin{aligned} &\underset{x,y}{\text{minimize}} && g(x,y) \\ &\text{s.t.} && f(x,y)\leqq 0, \end{aligned}$$

where f is convex. We call a point (x, y) feasible if $f(x,y)\leqq 0$.

The algorithm in R^3 generalizes that of the plane along the following lines. We will first develop a routine which tests straight lines in the plane. The goal of the test is to select one of the half planes determined by the line, such that the rest of the computation may be confined to that half plane. Then a procedure is developed during which applications of the test enable us to drop relatively large families of inequalities, so that the cost of the tests which follow becomes smaller and smaller.

5.2. Testing a line. Given any straight line in the x, y-plane, we wish to recognize which of the two half planes, determined by the line, is relevant for our problem. For simplicity, assume that the line coincides with the x-axis (otherwise transform the coordinate system accordingly). So, the goal of the test is to find whether we should look for a point (x, y) with positive y or negative y. The conclusion of the test may be one of the following: (i) The problem is infeasible. (ii) There is a global minimum with $y = 0$. (iii) The problem is unbounded. (iv) The rest of the computation should be in the half plane $\{(x, y): y > 0\}$. (v) The rest of the computation should be in the half plane $\{(x, y): y < 0\}$.

We may reach conclusions (iv) or (v) in one of the following circumstances. We may find that there are no feasible solutions on the x-axis and realize that if there are any feasible points then they all must lie in the half plane we have found. On the other hand, we may find a feasible point on the x-axis but realize that, in order to decrease the value of the function g, we must proceed into the half plane we have identified.

The test amounts to finding the minimum of $g(x, 0)$ subject to $f(x, 0) \leqq 0$ and then analyzing the picture in the neighborhood of the solution of this optimization problem. Our planar linear programming algorithm in § 2 is capable of reaching one of the following results: (i) It may find out that the problem is unbounded (even when restricted to the x-axis). (ii) It may find out that the problem (on the x-axis) is infeasible, in which case it will produce a point $(x^*, 0)$ which minimizes $f(x, 0)$ (in this case $f(x, 0) > 0$ for every x). (iii) It may find a point $(x^*, 0)$ which minimizes $g(x, 0)$ subject to the constraint $f(x, 0) \leqq 0$. If the problem is unbounded on the x-axis then, of course, we are done. So, assume a point $(x^*, 0)$ has been produced. Without loss of generality assume $x^* = 0$ (otherwise, translate the x-coordinate accordingly). The test relies only on the inequalities which are tight at $(0, 0)$. Formally, we define subsets $I_j^* \subset I_j$ $(j = 1, 2, 3)$ as follows. An index $i \in I_1$ belongs to I_1^* if $c_i = g(0, 0)$ (i.e., $c_i = \max\{c_j: j \in I_1\}$). An index $i \in I_2$ belongs to I_2^* if $c_i = h(0, 0)$ provided $f(0, 0) = g(0, 0) - h(0, 0) \geqq 0$. Note that $I_2^* = \varnothing$ if either $f(0, 0) < 0$ or $f(0, 0) > g(0, 0) - h(0, 0)$. Finally, an index $i \in I_3$ belongs to I_3^* if $c_i = e(0, 0)$ provided $f(0, 0) = e(0, 0) \geqq 0$.

Distinguish two cases according to the circumstances of producing the point $(x^*, 0)$ (which is now assumed to be equal to $(0, 0)$):

Case 1. $f(0, 0) \leqq 0$. In this case we have found a feasible point and are interested in decreasing the value of the function g (subject to $f(x, y) \leqq 0$). The test is based on the following:

PROPOSITION 1. *The existence of a point* (x, y) *such that* $y > 0$, $g(x, y) < g(0, 0)$ *and* $f(x, y) \leqq 0$ *is equivalent to the existence of a* λ *such that*

$$\text{(i)} \qquad \max\{a_i\lambda + b_i: i \in I_1^*\} < 0,$$

$$\text{(ii)} \qquad \max\{a_i\lambda + b_i: i \in I_1^*\} \leqq \min\{a_i\lambda + b_i: i \in I_2^*\}, \quad \textit{and}$$

$$\text{(iii)} \qquad \max\{a_i\lambda + b_i: i \in I_3^*\} \leqq 0.$$

Proof. Suppose there is such a point (x, y). Let $\lambda = x/y$. It follows that

$$\max\{a_i\lambda y + b_i y + c_i : i \in I_1^*\} \leqq g(\lambda y, y) < g(0, 0) = \max\{c_i : i \in I_1^*\}$$

so that (i) holds. If $I_2^* = \varnothing$ then (ii) is trivial. Thus, assume $I_2^* \neq 0$. This implies $f(0, 0) = g(0, 0) - h(0, 0) = 0$. Here for every $i \in I_1^*$ and $j \in I_2^*$, $c_i = c_j$. Since

$$\begin{aligned}\max\{a_i\lambda y + b_i y + c_i : i \in I_1^*\} &\leqq g(\lambda y, y) \leqq h(\lambda y, y) \\ &\leqq \min\{a_i\lambda y + b_i y + c_i : i \in I_2^*\},\end{aligned}$$

it follows that (ii) holds. The validity of (iii) is proved analogously. Conversely, suppose that there is λ which satisfies (i), (ii), and (iii). If $y > 0$ is sufficiently small, then obviously $g(\lambda y, y) < g(0, 0)$ and $f(\lambda y, y) \leqq 0$. This completes the proof.

PROPOSITION 2. *The existence of a point (x, y) such that $y < 0$, $g(x, y) < g(0, 0)$ and $f(x, y) \leqq 0$ is equivalent to the existence of a number λ such that*

(i) $$\min\{a_i\lambda + b_i : i \in I_1^*\} > 0,$$

(ii) $$\min\{a_i\lambda + b_i : i \in I_1^*\} \geqq \max\{a_i\lambda + b_i : i \in I_2^*\}, \quad \textit{and}$$

(iii) $$\min\{a_i\lambda + b_i : i \in I_3^*\} \geqq 0.$$

The proof is analogous to that of Proposition 1.

We can now describe the rest of the test in the case where $f(0, 0) \leqq 0$. Given the sets I_j^* $(j = 1, 2, 3)$, we solve the following problem:

$$\begin{aligned}\underset{\lambda,\eta}{\text{minimize}} \quad & \eta \\ \text{s.t.} \quad & \eta \geqq a_i\lambda + b_i \qquad (i \in I_1^*), \\ & \eta \leqq a_i\lambda + b_i \qquad (i \in I_2^*), \\ & a_i\lambda + b_i \leqq 0 \qquad (i \in I_3^*).\end{aligned}$$

If a negative η is obtained then the half plane $\{(x, y) : y > 0\}$ is the proper one. Otherwise, we need to solve the following problem:

$$\begin{aligned}\underset{\lambda,\eta}{\text{minimize}} \quad & \eta \\ \text{s.t.} \quad & \eta \leqq a_i\lambda + b_i \qquad (i \in I_1^*), \\ & \eta \geqq a_i\lambda + b_i \qquad (i \in I_2^*), \\ & a_i\lambda + b_i \geqq 0 \qquad (i \in I_3^*).\end{aligned}$$

If a positive η is obtained then the half plane $\{(x, y) : y < 0\}$ is the proper one. In the remaining case the constraint $y = 0$ does not affect the global minimum and hence the point $(0, 0)$ (i.e., $(x^*, 0)$) is an optimal solution.

Case 2. $f(0, 0) > 0$. We are then interested in decreasing the value of f by entering one of the half planes. The conclusions in this case are based on the following propositions whose proofs are similar to that of Proposition 1.

PROPOSITION 3. *The existence of a point (x, y) such that $y > 0$ and $f(x, y) < f(0, 0)$ is equivalent to the existence of a number λ such that*

(i) $$\max\{a_i\lambda + b_i : i \in I_1^*\} < \min\{a_i\lambda + b_i : i \in I_2^*\} \quad \textit{and}$$

(ii) $$\max\{a_i\lambda + b_i : i \in I_3^*\} < 0.$$

PROPOSITION 4. *The existence of a point* (x, y) *such that* $y<0$ *and* $f(x, y)<f(0, 0)$ *is equivalent to the existence of a number* λ *such that*

(i) $$\min\{a_i\lambda+b_i: i\in I_1^*\}>\max\{a_i\lambda+b_i: i\in I_2^*\} \quad \textit{and}$$

(ii) $$\min\{a_i\lambda+b_i: i\in I_3^*\}>0.$$

Thus, in the case where $f(0, 0)>0$ the test proceeds as follows. Consider the function

$$\varphi(\lambda)=\max(\max\{a_i\lambda+b_i: i\in I_1^*\}-\min\{a_i\lambda+b_i: i\in I_2^*\}, \max\{a_i\lambda+b_i: i\in I_3^*\}).$$

This is a convex piecewise linear function, and our methods in § 2 are applicable for finding its minimum in $O(n)$ time. In minimizing $\varphi(\lambda)$ we form pairs (i, j) of indices only when i and j belong to the same set I_k^* $(1\leqq k\leqq 3)$. If φ attains a negative value then the half plane $\{(x, y): y>0\}$ is the correct domain wherein to look for feasible points (see Proposition 3). Otherwise, we need to consider the function

$$\psi(\lambda)=\min(\min\{a_i\lambda+b_i: i\in I_1^*\}-\max\{a_i\lambda+b_i: i\in I_2^*\}, \min\{a_i\lambda+b_i: i\in I_3^*\}).$$

Analogously, if ψ attains a positive value then the half plane $\{(x, y): y<0\}$ is the correct one. In the remaining case f attains its global minimum on the x-axis, and that implies that our original problem is infeasible.

This completes the statement of our test of a given straight line.

5.3. The algorithm. The algorithm is based on the following principle. Consider two inequalities of the form $z\geqq a_ix+b_iy+c_i$, $z\geqq a_jx+b_jy+c_j$, i.e., $i, j\in I_1$. If $(a_i, b_i)=(a_j, b_j)$ then one of these constraints is redundant. Otherwise, let $L_{ij}=\{(x, y): a_ix+b_iy+c_i=a_jx+b_jy+c_j\}$. If $(a_i, b_i)\neq(a_j, b_j)$ then L_{ij} is a straight line which divides the plane into two halves. If we know that an optimal solution to our problem (if there is any) must lie in a certain half plane determined by L_{ij} then we may discard one of the two inequalities. A similar observation is true for pairs of inequalities $z\leqq a_ix+b_iy+c_i$, $z\leqq a_jx+b_jy+c_j$, i.e., when $i, j\in I_2$ as well as for pairs i, j such that $i, j\in I_3$.

We start the procedure by arranging the inequalities (except for at most three of them) in disjoint pairs so that the two members of each pair belong to the same set I_k $(1\leqq k\leqq 3)$. For each pair, either we can drop one of the participating inequalities right away, or we have a dividing line L_{ij}. Consider the set of lines that are generated in this way. At this point our procedure is essentially the same as in the problem of the smallest circle enclosing n points (see § 4.3). We review the basic ideas here in short. Given a set of straight lines, we will in $O(n)$ time find a subset of at least a quarter of the lines, such that for each line in that subset, it is known which of the two corresponding half planes may contain the solution. This is done as follows: First, we transform the coordinate system so that half the lines will have nonnegative slope and half the lines will have nonpositive slope. We then form pairs of lines where in every pair we will have one line with nonnegative slope and one line with nonpositive slope. Let the lines be redenoted $L_1, \cdots, L_k$. If L_i and L_j are members of one of our pairs then let (x_{ij}, y_{ij}) denote their point of intersection if there is a unique point. Otherwise, the two lines must be parallel to the x-axis and we define y_{ij} to be the mean of their y-coordinates. By testing the line $y=y_m$ (where y_m is the median of the y_{ij}'s) we identify a set of at least half the pairs whose y_{ij} values are either all greater than the y-coordinate of an optimal solution or all smaller than that. We then test the line $x=x_m$ (where x_m is the median of the x_{ij}'s of those pairs of nonparallel lines, that have been identified after the test at $y=y_m$). For at least a quarter of the pairs

we will be able to find a line and a corresponding half plane in which the optimal solution cannot lie. This enables us to drop one inequality per pair for at least a quarter of the pairs, i.e., at least $\frac{1}{16}$ of the inequalities are dropped. This establishes the linearity of the run time of our algorithm. Again, the details are given in § 4.3.

5.4. Quadratic programming in R^3. Our results can be extended to solve the problem of minimizing a convex quadratic function, subject to linear constraints, in R^3 in linear time. Formally, the problem is

$$\begin{aligned} &\underset{v\in R^3}{\text{minimize}} && v^T\cdot A\cdot v+b^T\cdot v \\ &\text{s.t.} && a_i^T\cdot v\leqq\beta_i \qquad (i=1,\cdots,n) \end{aligned}$$

where A is a 3×3 positive semidefinite matrix and $b, a_1,\cdots,a_n\in R^3$. If A is not identically zero then by an appropriate affine transformation of R^3 (which can be found in constant time) we can transform our problem (in linear time) to the following:

$$\begin{aligned} &\underset{x,y,z}{\text{minimize}} && \alpha x^2+\beta y^2+\gamma xy+z^2 \\ &\text{s.t.} && z\geqq a_ix+b_iy+c_i \qquad (i\in I_1), \\ &&& z\leqq a_ix+b_iy+c_i \qquad (i\in I_2), \\ &&& a_ix+b_iy+c_i\leqq 0 \qquad (i\in I_3) \end{aligned}$$

where $\gamma^2\leqq 4\alpha\beta$, $\alpha,\beta>0$. The algorithm for linear programming can be extended to solve a problem of this kind along the same lines. That includes a routine for solving quadratic programming problems in the plane. The latter can be modeled as

$$\begin{aligned} &\underset{x,y}{\text{minimize}} && \alpha x^2+\beta x+y^2 \\ &\text{s.t.} && y\geqq a_ix+b_i \qquad (i\in I_1), \\ &&& y\leqq a_ix+b_i \qquad (i\in I_2), \\ &&& a\leqq x\leqq b \end{aligned}$$

where $\alpha,\beta\geqq 0$. Let $g(x)$ and $h(x)$ be defined as in § 2.1. Also, let $g^+(x)=\text{Max}\,(0,g(x))$ and $h^-(x)=\text{Min}\,(0,h(x))$. Consider the following problems:

(1)
$$\begin{aligned} &\text{minimize} && \alpha x^2+\beta x+(g^+(x))^2 \\ &\text{s.t.} && g^+(x)\leqq h(x), \\ &&& a\leqq x\leqq b; \end{aligned}$$

(2)
$$\begin{aligned} &\text{minimize} && \alpha x^2+\beta x+(h^-(x))^2 \\ &\text{s.t.} && g(x)\leqq h^-(x), \\ &&& a\leqq x\leqq b. \end{aligned}$$

It can be verified that our problem reduces to solving both these problems. A solution for our problem is then produced as follows: Select the problem whose minimum is smaller and let x be its minimizer. We then let y be equal to either $g^+(x)$ or $h^-(x)$, according to the case. The solution of either problem is analogous to linear programming in R^2. A similar observation holds for the variable z in the three-dimensional case.

6. Conclusion. We have demonstrated a powerful computational method for solving different problems which is based on successive reductions of the input of the given problem. The method yields linear-time algorithms. The natural question now is whether the general linear programming problem is solvable in linear time in any fixed number of variables. We already know that the answer is in the affirmative [M3]; however, this requires a nontrivial extension of the present paper, as we argue below.

Consider any pair of constraints

$$y \geqq \sum_{j=1}^{d-1} a_{1j}x_j + b_1, \qquad y \geqq \sum_{j=1}^{d-1} a_{2j}x_j + b_2$$

in a linear programming problem where we seek to minimize y. In the space of the x_j's we have the hyperplane $H_{12} \equiv \sum a_{1j}x_j + b_1 = \sum a_{2j}x_j + b_2$ which defines two half spaces; in each of these half spaces we have one of the two constraints dominated by the other one. Thus, it may be useful to know on which side of H_{12} the solution lies. Attempting to generalize what we know from the case where $d = 3$, we do the following: Let H_{12} be represented by an equation of the form $\sum a_j x_j = b$, and for any $S \subset \{1, \cdots, d-1\}$, denote

$$R^S = \{x \in R^{d-1}: x_i \geqq 0 \text{ if } j \in S \text{ and } x_i \leqq 0 \text{ if } j \notin S\}.$$

Also let $T = \{j: a_j \geqq 0\}$ and $\bar{T} = \{1, \cdots, d-1\} \backslash T$. It can be easily verified that at least one of the two orthants, R^T and $R^{\bar{T}}$, lies entirely on one side of H_{12} (depending on the sign of b). For example, if $b \leqq 0$ then R^T lies entirely on one side of H_{12}. In this case, if we knew that our solution lay in R^T then we could drop one of our two constraints mentioned above.

For the case where $d = 3$ we found a way to exploit this useful observation. We will now review that method from a somewhat different point of view.

First, the lines are grouped in disjoint pairs in a manner which takes their slopes into account. Then a single point (x_m, y_m) is found with the following property: If the origin is translated into (x_m, y_m), and if a certain linear transformation is applied, then, relative to the new coordinate system, the solution lies in an orthant which lies entirely on one side of each line, for $\frac{1}{8}$ of the set of all lines. This is based on the property that if the solution lies in the orthant R^T, then at least a quarter of the pairs of lines intersect in $R^{\bar{T}}$, and each pair is guaranteed to contain a line whose coefficients suit the sign type of R^T. This idea works in the case where $d = 3$, since then there are only two pairs of "opposing" orthants (recall that the space of x_j's is of dimension $d-1$): $(R^{\{1,2\}}, R^{\varnothing})$ and $(R^{\{1\}}, R^{\{2\}})$. Thus if, for example, the solution is known to belong to $R^{\{1\}}$, then we also have a quarter of our pairs intersecting in $R^{\{2\}}$, and at least one line per such pair has the orthant $R^{\{1\}}$ lying on one side of it.

In higher dimensions we face the following difficulty. Suppose that in the d-dimensional case (the dimension of the x-space is $d-1$) we group the hyperplanes in disjoint sets of cardinality $d-1$. Suppose that each such set is linearly independent so that it determines a single point (the other case is even simpler). Now, it is obvious that we can find a point $(x_1, \cdots, x_{d-1})$ and an orthant R^T (defined relative to this point), such that $R^{\bar{T}}$ contains the solution, and at least $1/2^{d-1}$ of the sets intersect in $R^{\bar{T}}$. However, here is the critical point. The number of pairs of opposing orthants is 2^{d-2}. We need each set of $d-1$ hyperplanes to contain (for each pair of orthants $(R^T, R^{\bar{T}})$) at least one member whose coefficients match the signs pattern of $(R^T, R^{\bar{T}})$. It is thus required that $d-1 \geqq 2^{d-2}$, which holds only if $d = 3$. Thus, a different approach is needed here. It is also interesting to mention here that previous work on

the convex hull [PH] applied only for $d \leqq 3$. However, we already know that the methods of the present paper do extend to higher dimensions [M3].

With regard to practical implications of the algorithms in the present paper we can say the following: First, we do not expect the linear-time median-finding algorithm to be useful. Thus, it is preferable to use a good practical algorithm like Floyd and Rivest's [FR]. Moreover, we do not have to find the exact median. Another practical consideration is that information may be saved when starting a new iteration of the process. A practical version of our algorithm would be very efficient for solving problems arising in computer graphics such as hidden-line elimination by means of linear programming (see [BS]).

Appendix. The extreme-point problem in the plane. In this Appendix we solve the following problem: Given n points (a_i, b_i), $i = 1, \cdots, n$, in the plane and another point (a, b), find out whether or not (a, b) is a convex combination of the points $(a_1, b_1), \cdots, (a_n, b_n)$. If so, then represent (a, b) as a convex combination of three points; otherwise, find two points (a_i, b_i), (a_j, b_j) such that all the points belong to the cone whose vertex is at (a, b) and whose extreme rays are determined by (a_i, b_i) and (a_j, b_j). This problem can be solved by our linear programming algorithm in the plane. However, a more straightforward method can be developed as follows:

Without loss of generality assume $(a, b) = (0, 0)$ and $(a_1, b_1) = (0, 1)$ (otherwise apply an affine transformation accordingly). If $(0, 0)$ is not in the convex hull of the n given points, then there exists an α such that $b_i > \alpha a_i$ for every i. That is, there is a separating line (as mentioned in the introduction). Thus, α must satisfy

$$\max \{b_i/a_i : a_i < 0\} < \alpha < \min \{b_i/a_i : a_i > 0\},$$

and also $b_i > 0$, for every i such that $a_i = 0$. If this is indeed the case, then the extreme rays are determined by a point at which the maximum on the left-hand side is attained (or the point $(0, 1)$ if $a_i \geqq 0$ for every i), and by a point at which the minimum on the right-hand side is attained (or, again, the point $(0, 1)$). Otherwise, the point $(0, 0)$ is either a convex combination of two such points together with the point $(a_1, b_1) = (0, 1)$, or a convex combination of two points on the y-axis.

REFERENCES

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

[BS] R. P. BURTON AND D. R. SMITH, *A hidden-line algorithm for hyperspace*, this Journal, 11 (1982), pp. 71–80.

[CP] R. CHANDRASEKARAN AND M. J. A. P. PACCA, *Weighted min-max and max-min location problems: Finite and polynomially bounded algorithms*, Oper. Res., 17 (1980), pp. 172–180.

[CR] R. COURANT AND H. ROBBINS, *What is Mathematics?*, Oxford Univ. Press, New York, 1941.

[DF] P. M. DEARING AND R. L. FRANCIS, *A minimax location problem on a network*, Transportation Sci., 8 (1974), pp. 333–343.

[DR] D. P. DOBKIN AND S. P. REISS, *The complexity of linear programming*, Theoret. Comput. Sci., 11 (1980), pp. 1–18.

[DW] Z. DREZNER AND G. O. WESOLOWSKY, *Single Facility l_p-distance minimax location*, SIAM J. Alg. Disc. Meth., 1 (1980), pp. 315–321.

[EH] J. ELZINGA AND D. E. HEARN, *Geometrical solutions for some minimax location problems*, Transportation Sci., 6 (1972), pp. 379–394.

[FR] R. W. FLOYD AND R. L. RIVEST, *Expected time bounds for selection*, Comm. ACM, 8 (1975), pp. 165–172.

[F] R. R. L. FRANCIS, *Some aspects of a minimax location problem*, Operat. Res., 15 (1967), pp. 1163–1168.

[FW] R. L. FRANCIS AND J. A. WHITE, *Facility Layout and Location*, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[G] R. L. GRAHAM, *An efficient algorithm for determining the convex hull of a finite planar set*, Inform. Process. Lett., 1 (1972), pp. 132–133.

[HSP] S. L. HAKIMI, E. F. SCHMEICHEL AND J. G. PIERCE, *On p-centers in networks*, Transportation Sci., 12 (1978), pp. 1–15.

[HM] G. Y. HANDLER AND P. B. MIRCHANDANI, *Location on Networks Theory and Algorithms*, MIT Press, Cambridge, MA, 1979.

[KH] O. KARIV AND S. L. HAKIMI, *An algorithmic approach to network location problems, Part* I. *The* 1*-centers*, SIAM J. Appl. Math., 37 (1979), pp. 513–538.

[L] N. LEVIN, *Location problems on weighted graphs*, unpublished thesis, Tel Aviv Univ., 1977.

[M1] N. MEGIDDO, *Applying parallel computation algorithms in the design of serial algorithms*, in Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Angeles, 1981, pp. 399–408; J. Assoc. Comput. Math., to appear.

[M2] ———, *The weighted Euclidean* 1*-center problem*, Math. Oper. Res., to appear.

[M3] ———, *Linear programming in linear time when the dimension is fixed*, J. Assoc. Comput. Mach., to appear.

[NC] K. P. K. NAIR AND R. CHANDRASEKARAN, *Optimal location of a single service center of certain types*, Naval Res. Logist. Quart., 18 (1971), pp. 503–510.

[PH] 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.

[RT] H. RADEMACHER AND O. TOEPLITZ, *The Enjoyment of Mathematics*, Princeton Univ. Press, Princeton, NJ, 1957.

[Sh] M. I. SHAMOS, *Geometric complexity*, in Proc. 7th Annual ACM Symposium on Theory of Computing, 1975, ACM, New York, 1975, pp. 224–233.

[ShH] M. I. SHAMOS AND D. HOEY, *Closest-point problems*, in Proc. 16th Annual IEEE Symposium on Foundations of Computer Science, 1975, IEEE Computer Society Press, Los Angeles, 1975, pp. 151–162.

[Sm] R. D. SMALLWOOD, *Minimax detection station placement*, Oper. Res., 13 (1965), pp. 636–646.

[Sy1] J. J. SYLVESTER, *A question in the geometry of situation*, Quart. J. Math., 1 (1857), p. 79.

[Sy2] ———, *On Poncelet's approximate valuation of Surd forms*, Philosophical Mag., XX, 4th Series (1860), pp. 203–222.

[Y] A. C. YAO, *A lower bound for finding convex hulls*, J. Assoc. Comput. Mach., 28 (1981), pp. 780–787.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0013 $01.25/0

THE COMPLEXITY OF COUNTING CUTS AND OF COMPUTING THE PROBABILITY THAT A GRAPH IS CONNECTED*

J. SCOTT PROVAN† AND MICHAEL O. BALL‡

Abstract. Several enumeration and reliability problems are shown to be # P-complete, and hence, at least as hard as NP-complete problems. Included are important problems in network reliability analysis, namely, computing the probability that a graph is connected and counting the number of minimum cardinality (s, t)-cuts or directed network cuts. Also shown to be # P-complete are counting vertex covers in a bipartite graph, counting antichains in a partial order, and approximating the probability that a graph is connected and the probability that a pair of vertices is connected.

Key words. complexity, # P-complete, graphs, reliability, network reliability

1. Introduction. The inherent intractability of certain counting and reliability problems has been studied by Ball [1], Rosenthal [11], and Valiant [12]. Valiant defines the notion of the # P-complete class of counting problems, and shows that problems in this class are at least as hard as NP-complete problems. He then goes on to show that several important counting and reliability problems are # P-complete, among them, counting perfect matchings in bipartite graphs and evaluating the probability that two given nodes in a probabilistic graph are connected. Three important problems are mentioned by Ball and Valiant, for which the complexity is not known, namely:

(1) evaluating the probability that a probabilistic graph is connected,

(2) approximating the probability that a probabilistic graph is connected,

(3) approximating the probability that two vertices of a probabilistic graph are connected.

In view of results by the authors in [2], the probability measure associated with problems (1) and (2) seems to have considerably more structure than that associated with (3). In [3] they also show the power of the structure in providing good upper and lower bounds for this measure. We show in this paper, however, that all three of these problems are NP-hard, in particular, # P-complete. In the process, we show that several counting problems are also # P-complete, among them: counting the number of node covers in a bipartite graph, counting antichains in a partial order, and counting minimum cardinality directed network cuts.

We now fix some terminology. Let $G = (V, E)$ be a graph with vertex set V and edge set E and let $m = |V|$ and $n = |E|$. When specified, G directed implies that the edges are taken to be ordered pairs, and G undirected implies the pairs are unordered. When not specified, G is allowed to be either. We allow loops (edges whose two end points are the same) and multiple edges (edges with the same pair of end points), although these are not strictly required for the results of this paper. Let s and t be two vertices in the graph G (directed or undirected). An (s, t)-*path* in G is any sequence $s = v_0, e_1, v_1, \cdots, v_{k-1}, e_k, v_k = t$ of vertices $v_0, v_1, \cdots$ and edges $e_1, e_2, \cdots$ with $e_j = (v_{j-1}, v_j)$ for $j = 1, \cdots, k$. An (s, t)-*cut* in G is any minimal set of edges that intersects every (s, t)-path. A *network cut (with respect to s)* is any minimal set of edges that is an (s, t)-cut for some vertex t in G. A *spanning tree (rooted at s)* is a

* Received by the editors January 23, 1981, and in final revised form October 26, 1982.

† Curriculum in Operations Research and Systems Analysis, University of North Carolina, Chapel Hill, North Carolina, 27514. This work was performed while this author was an NRC/NAS postdoctoral associate at the National Bureau of Standards.

‡ College of Business and Management, University of Maryland, College Park, Maryland 20742.

minimal set of edges that contains paths from s to all other vertices in G. Note that if G is undirected, a network cut comprises any minimal set of edges whose removal disconnects G and a spanning tree is any minimal set of edges that connects all vertices; in both cases the definition is independent of the choice of s.

We now define our reliability measures. Given any real p, $0 \leqq p \leqq 1$, we impose the stochastic structure on G in which the edges of G are subject to random failure, independently and each with equal probability p. Edges that have not failed are said to be operative. We are concerned with two composite reliability measures on this stochastic model, which we will denote as functions of p. The first is the (s, t) *connectedness measure*: given vertices s and t in G,

$$f(G, s, t; p) = \Pr\{\text{there is a path of operative edges from } s \text{ to } t\}$$
$$= \Pr\{\text{the failed edges of } G \text{ do not contain an } (s, t)\text{-cut}\}.$$

The second is the *connectedness measure*: given vertex s in G,

$$g(G, s; p) = \Pr\{\text{there is a path of operative edges from } s \text{ to every other vertex in } G\}$$
$$= \Pr\{\text{the failed edges of } G \text{ do not contain a network cut}\}.$$

These measures are defined for both directed and undirected graphs. If G is undirected, then $f(G, s; p)$ is the probability that the operative edges in G form a connected graph on V, and is independent of the vertex s. The combinatorial significance of these reliability measures can be seen by expanding f and g:

$$f(G, s, t; p) = \sum_{j=0}^{n} f_j p^j (1-p)^{n-j},$$

$$g(G, s; p) = \sum_{j=0}^{n} g_j p^j (1-p)^{n-j},$$

where

f_j = number of sets of edges of cardinality j whose complement admits a path from s to t,
= number of sets of edges of cardinality j that do not contain an (s, t)-cut;
g_j = number of sets of edges of cardinality j whose complement admits a path from s to every vertex in G,
= number of sets of edges of cardinality j that do not contain a network cut with respect to s.

The use of this form of the polynomial might seem slightly unnatural since coefficients are defined in terms of complements. However, it is consistent with the independence system interpretation of the reliability analysis problem used in other papers [2], [3]. Thus, the evaluation of f and g depend on the counting problems associated with (s, t)-cuts and network cuts in a way that will be shown precisely below.

We explore the computational complexity of counting and reliability problems in the manner proposed by Valiant [12]. The study of the complexity of feasibility and optimization problems has been pursued in the setting of recognition problems [5]. An important class is NP, which consists of those recognition problems accepted by a nondeterministic Turing machine of polynomial time complexity. The "hardest" problems in NP are called NP-complete; it is generally considered unlikely that polynomial algorithms exist for solving problems in this class. Valiant defines # P to be the set of integer-valued functions that can be computed by counting the number

of accepting computations of some nondeterministic Turing machine of polynomial time complexity. We extend Valiant's definition slightly to include rational and multiple valued functions that can be evaluated using functions of the above type. We say that a function f is *polynomially reducible* to a function g ($f \propto g$) is there exists an algorithm which, for any input z, evaluates $f(z)$ with a number of elementary operations and evaluations of g that is polynomial in the length of z. A function f is called # P-*complete* if (a) f is in # P and (b) every function g in # P can be reduced to f by a polynomial time reduction. The classes # P and # P-complete provide a natural setting for studying the complexity of counting and reliability problems. We remark that the counting problem associated with a given recognition problem is at least as hard as the recognition problem. In particular, the counting versions of NP-complete problems are NP-hard, i.e. at least as hard as NP-complete problems. To illustrate this point, note that a polynomial algorithm to determine the *number* of Hamiltonian circuits in a graph would immediately give a polynomial algorithm to determine if a graph contained at least one Hamiltonian circuit. In fact, the counting versions of most NP-complete problems can be easily shown to be # P-complete. See [5] for a detailed treatment of NP-completeness and its relationship to # P-completeness.

With these definitions in mind we state our main result:

THEOREM. *The following functions are* # P-complete:

1. BIPARTITE VERTEX COVER
   Input: *bipartite graph* $G = (V, E)$
   Output: $|\{S \subseteq V$: *for each* $e = (u, w) \in E$, $u \in S$ *or* $w \in S\}|$;
2. BIPARTITE INDEPENDENT SET
   Input: *bipartite graph* $G = (V, E)$
   Output: $|\{S \subseteq V$: *for all* $u, w \in S$, $e = (u, w) \notin E\}|$;
3. ANTICHAIN
   Input: *partial order* $(X, \leqq)$
   Output: $|\{S \subseteq X$: *there are no* $x, y \in S$ *with* $x \leqq y\}|$;
4. MINIMUM CARDINALITY BIPARTITE VERTEX COVER (MAXIMUM CARDINALITY BIPARTITE INDEPENDENT SET, MAXIMUM CARDINALITY ANTICHAIN, RESPECTIVELY)
   Input: *same as* 1 (2, 3, *resp.*)
   Output: *the number of minimum cardinality* (*maximum cardinality, resp.*) *elements of the output set*;
5. BIPARTITE 2-SAT WITH NO NEGATIONS
   Input: *Boolean expression* B *in the variables* $x_1, \cdots, x_k, y_1, \cdots, y_l$ *of the form* $B = (x_{i_1} \vee y_{j_1}) \wedge \cdots \wedge (x_{i_n} \vee y_{j_n})$
   Output: $|\{x_1, \cdots, x_k, y_1, \cdots, y_l)$ *that satisfy* $B\}|$;
6. MINIMUM CARDINALITY (s, t)-CUT
   Input: *graph* $G = (V, E)$, $s, t \in V$
   Output: $|\{C \subseteq E$: C *is a minimum cardinality* (s, t)*-cut in* $G\}|$;
7. MINIMUM CARDINALITY DIRECTED NETWORK CUT
   Input: *directed graph* $G = (V, E)$, $s \in V$
   Output: $|\{C \subseteq E$: C *is a minimum cardinality network cut with respect to* $s\}|$;
8. CONNECTEDNESS RELIABILITY
   Input: *graph* $G = (V, E)$, $s \in V$, *rational* p, $0 \leqq p \leqq 1$
   Output: $g(G, s; p)$;
9. CONNECTEDNESS RELIABILITY ε-APPROXIMATION
   Input: *graph* $G = (V, E)$, $s \in V$, $\varepsilon \leqq 0$, *rational* p, $0 \leqq p \leqq 1$
   Output: *rational* r *with* $r - \varepsilon < g(G, s; p) < r + \varepsilon$;

10. (s, t) CONNECTEDNESS RELIABILITY ε-APPROXIMATION
Input: *graph* $G=(V, E)$, $s, t \in V$, $\varepsilon > 0$, *rational* p, $0 \leqq p \leqq 1$
Output: *rational* r *with* $r-\varepsilon < f(G, s, t; p) < r+\varepsilon$;

Before going on to the proof of the theorem, we illustrate how our results fit in with previous results concerning reliability and important related counting problems. Computation of the functions f and g are considered the two most important and well-studied network reliability problems. The theorem settles the complexity of computing g exactly and the ε-approximation problem for f and g. In terms of computing or approximating f, two important quantities are the number of minimum cardinality (s, t)-cuts and the number of minimum cardinality (s, t)-paths. These correspond, respectively, to the first $f_i < \binom{n}{i}$ and the last $f_i > 0$. The two corresponding quantities for g are the number of minimum cardinality network cuts and the number of minimum cardinality connected sets, i.e. spanning trees, and these correspond, respectively, to the first $g_i < \binom{n}{i}$ and the last $g_i > 0$. Table 1 describes the known

TABLE 1

	Min. card. pathset	Min. card. cutset	Rel. poly.	Rel. approx.
undirected and directed two-terminal (f)	* [3]	! TH	! [12]	! TH
undirected network (g)	* [10]†	* [3]	! TH§	! TH
directed network (g)	* [10]†	! TH	! TH§¶	! TH

Either the appropriate reference is given or TH which indicates the result is contained in the theorems given in this paper; * implies polynomial; ! implies # P-complete.

† Reference [10] reduces the problem to computing the determinant of a matrix. It is (now) well known that determinants can be computed in polynomial time.

§ These results have recently been proven independently by Jerrum [9].

¶ This result has recently been proven independently by Hagstrom [6].

complexity results for all of these problems. It uses the generic term pathsets to refer to both spanning trees and (s, t)-paths and cutsets refer to both (s, t)-cuts and network cuts. Columns 1 and 2 refer to the problems of determining the number of minimum cardinality pathsets and cutsets respectively, column 3 to the problem of determining the polynomial f or g, and column 4 to the approximation problem defined in parts 9 and 10 of the theorem.

2. Proof of the theorem. The format for establishing a function f as # P-complete is as follows. We first establish that f is in # P by showing that, for any input z, there exists a polynomial algorithm for recognizing structures associated with the input z whose number is $f(z)$. In the context of the functions given in the theorem this is a trivial matter, since virtually all the functions count easily recognizable objects in the graph G associated with the input z. To show that f is # P-complete, we start with a known # P-complete function g, and show that there exists an algorithm which, for any z, evaluates $g(z)$ using a polynomial number of evaluations of f. In many cases this simply involves altering the input z (here the graph G) in polynomial time to a new input z' (here a new graph G') for which $g(z)=f(z')$. In some cases, however, we must evaluate f for a number of inputs $z_1, \cdots, z_n$, that number being polynomial in the size of z. We then relate the values $f(z_i)$, $i=1, \cdots, n$ to the value of g by

equations of the form

$$v_i = f(z_i) = \sum_{j=1}^{k} a_{ij} b_j, \qquad i = 1, \cdots, k \tag{1}$$

where the a_{ij} are known and $g(z)$ is some simple function of the b_j. If we can show that the $k \times k$ matrix of the coefficients a_{ij} for (1) is nonsingular, we can perform k evaluations of f, and then solve the linear system to obtain the values of b_j, and hence the value of $g(z)$.

Valiant, in [12], has made use of a special class of matrices to produce the desired nonsingular systems discussed above. A *Vandermonde matrix* is an $(n+1)\times(n+1)$ matrix of the form

$$\Delta = \begin{pmatrix} 1 & \mu_0 & \mu_0^2 & \cdots & \mu_0^n \\ 1 & \mu_1 & \mu_1^2 & \cdots & \mu_1^n \\ \vdots & & & & \\ 1 & \mu_n & \mu_n^2 & \cdots & \mu_n^n \end{pmatrix}$$

(or its transpose), where $\mu_0, \cdots, \mu_n$ are arbitrary real numbers. A well-known fact about these matrices (see, for example [7, § 5.1]) is $\det \Delta = \prod_{i>j} (\mu_i - \mu_j)$. We have immediately from the previous discussion:

LEMMA. *Suppose we have v_i and b_i, $i = 1, \cdots, n+1$, related by the equation*

$$v_i = \sum_{j=1}^{n+1} a_{ij} b_j, \qquad i = 1, \cdots, n+1.$$

Further, suppose that the matrix of coefficient (a_{ij}) is Vandermonde, with parameters $\mu_0, \cdots, \mu_n$ which are distinct. Then, given values for $v_1, \cdots, v_{n+1}$, we can obtain the values $b_1, \cdots, b_{n+1}$ in time polynomial in n.

We will make repeated use of this lemma throughout the proof of the theorem.

It is easy to see that the problems 1–10 of the theorem are in #P. To show that they are #P-complete, we establish a sequence of reductions, starting with the following counting problem.

CARDINALITY VERTEX COVER

Input: *graph $G = (V, E)$, integer k*

Output: $|\{S \subseteq V$: *S is a vertex cover for G and $|S| = k$*$\}|$.

This problem is known to be #P-complete (see [5, p. 169]). We also define one intermediate problem for purposes of the proof, namely,

0. VERTEX COVER

Input: *graph $G = (V, E)$*

Output: $|\{S \subseteq V$: *for each $e = (u, v) \in E$, $u \in S$ or $v \in S$*$\}|$.

We now give the reductions.

0. CARDINALITY VERTEX COVER $\propto$ VERTEX COVER. Given $G = (V, E)$, for $l = 1, \cdots, m = |V|$, construct graph $G'(l)$ with vertex set $V'(l) = \{v_i' : v \in V, i = 1, \cdots, l\}$ and edge set $E'(l) = \{(u_i', v_j')$: $(u, v) \in E$, $i = 1, \cdots, l$, $j = 1, \cdots, l\}$. This construction is illustrated in Fig. 1. Now every cover C' of $G'(l)$ has the property that if $(u, v) \in E$ then $\{u_1', \cdots, u_l'\} \subseteq C'$ or $\{v_1', \cdots, v_l'\} \subseteq C'$. Therefore, for each cover C of G there corresponds a class $\Omega(C)$ of covers of $G'(l)$ with elements of the form $\cup_{v \in V} S_v'$, where $S_v' = \{v_1', \cdots, v_l'\}$ if $v \in C$ and $S_v' \subsetneq \{v_1', \cdots, v_l'\}$ if $v \notin C$. The class $\Omega(C)$ consists of $(2^l - 1)^{m-|C|}$ covers, and the classes $\{\Omega(C)$:C a cover of $G\}$ partition

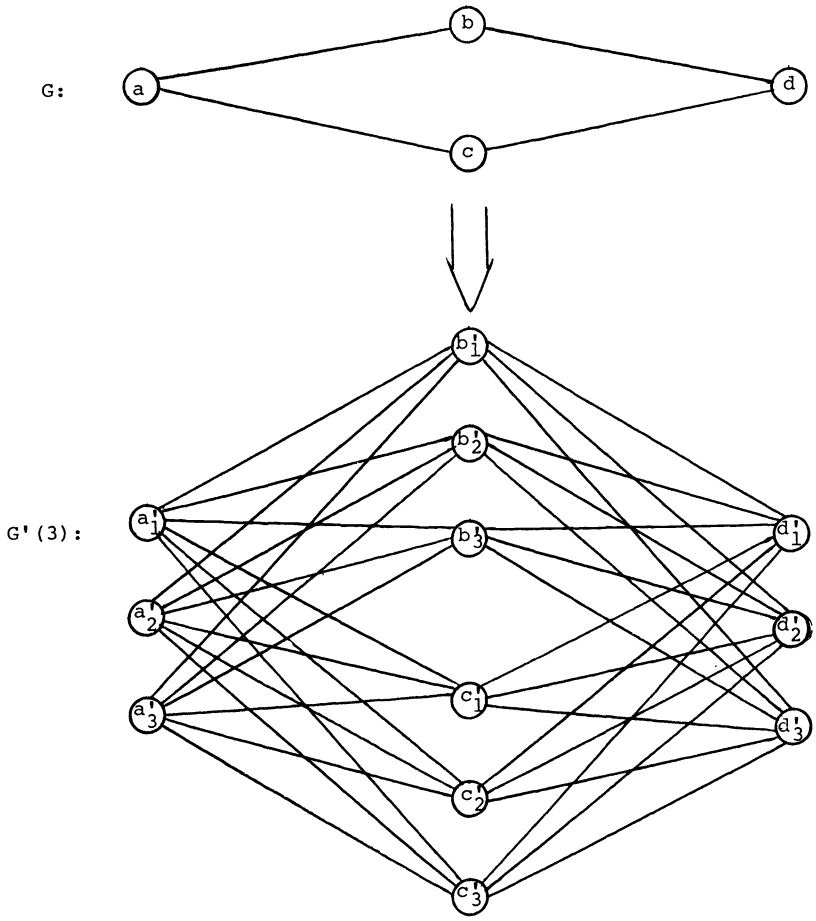


FIG. 1. *Example of transformation used in reduction* 0.

the covers of $G'(l)$. The number of covers of $G'(l)$ is therefore

$$\Gamma(l) = \sum_{i=0}^{m-1} A_i(2^l - 1)^i, \tag{2}$$

where A_i is the number of covers of G of cardinality $m-i$, $i=0,\cdots,m-1$. Now the $m \times m$ matrix $B=(b_{jl})$ with entries $b_{jl}=(2^l-1)^{j-1}$ $j=1,\cdots,m$, $l=1,\cdots,m$, is Vandermonde with $\mu_l = 2^l - 1$ distinct for $l=1,\cdots,m$. Therefore, by the lemma we can solve (2) to obtain each A_i, and hence solve the cardinality vertex cover problem.

1. VERTEX COVER $\propto$ BIPARTITE VERTEX COVER. Given $G=(V,E)$, for $l=0,\cdots,N=\binom{m+2}{2}-1$ construct bipartite graph $G'(l)$ by replacing each edge (u,v) in G by the subgraph shown in Fig. 2. (Note that when $l=0$, the graph $\Gamma'(l)$ has no edges at all.) This subgraph has the property that the number of vertex covers containing neither u nor v is 2^l, the number of covers containing a particular one of u or v is 3^l and the number of covers containing both u and v is 5^l. Thus, the number of covers of $G'(l)$ is

$$\Gamma'(l) = \sum_{\substack{i+j+k=n \\ i,j,k \geqq 0}} A_{ijk}(2^l)^i(3^l)^j(5^l)^k = \sum_{\substack{i+j+k=n \\ i,j,k \geqq 0}} A_{ijk}(2^i 3^j 5^k)^l, \tag{3}$$

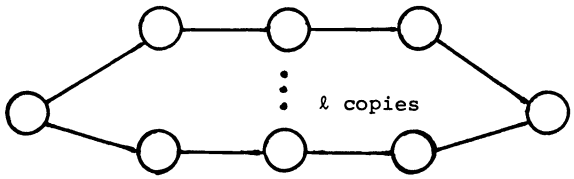


FIG. 2

where A_{ijk} is the number of sets S of vertices in G for which i edges of G have neither vertex in S, j edges have exactly one vertex in S, and k edges have both vertices in S. The $N \times N$ matrix $B = (b_{ql})$ defined

$$b_{ql} = (2^{i_q}3^{j_q}5^{k_q})^l, \qquad q = 1, \cdots, N, \qquad l = 0, \cdots, N-1,$$

where (i_q, j_q, k_q) are all triples summing to n, is Vandermonde. Further, $\mu_q = 2^{i_q}3^{j_q}5^{k_q} = 2^{i_r}3^{j_r}5^{k_r} = \mu_r$ if and only if $i_q = i_r$, $j_q = j_r$, and $k_q = k_r$. Therefore, the μ_q are distinct and by the lemma we can solve (3) to obtain each A_{ijk}, for $i+j+k = n$, $i \geqq 0$, $j \geqq 0$, $k \geqq 0$. In particular, we can obtain

$$A = \sum_{\substack{j+k=n \\ j,k \geqq 0}} A_{0jk},$$

which is the number of sets of vertices of G for which no edge of G is uncovered, that is, the number of covers of G.

2. BIPARTITE VERTEX COVER $\propto$ BIPARTITE INDEPENDENT SET. Given $G = (V, E)$ we note that $C \subseteq V$ is a cover for G if and only if $V - C$ is an independent set in G. The reduction follows.

3. BIPARTITE INDEPENDENT SET $\propto$ ANTICHAIN. Given bipartite graph $G = (V, E)$ with $V = V_1 \cup V_2$ and $E \subseteq V_1 X V_2$, define partial order $(X, \leqq)$ with $X = V$ and order defined for $x \neq y \in X$: $x \leqq y$ if and only if $x \in V_1$, $y \in V_2$, and $(x, y) \in E$. $(X, \leqq)$ is trivially transitive and antisymmetric. Further, a set $S \subseteq X$ is an antichain in $(X, \leqq)$ if and only if it is independent in G. The reduction follows.

4. BIPARTITE VERTEX COVER $\propto$ MINIMUM CARDINALITY BIPARTITE VERTEX COVER (MAXIMUM CARDINALITY BIPARTITE INDEPENDENT SET, MAXIMUM CARDINALITY ANTICHAIN, RESPECTIVELY). Given bipartite graph $G = (V, E)$, construct bipartite graph $G' = (V', E')$ by adding vertices $\{v' : v \in V\}$ to V and pendant edges $M = \{(v, v') : v \in V\}$ to E. Now since M consists of m disjoint edges that cover all vertices of G (a perfect matching), it follows that a minimum cardinality vertex cover of G' is of cardinality m. Furthermore, there is a 1–1 correspondence between vertex covers of G and minimum cardinality vertex covers of G' obtained by associating with cover C of G the cardinality m cover

$$C' = \{v : v \in C\} \cup \{v' : v \notin C\}.$$

In view of the discussion in reductions 2 and 3, it follows easily that the bipartite vertex cover problem reduces to any of the three given minimum or maximum cardinality problems.

5. BIPARTITE VERTEX COVER $\propto$ BIPARTITE 2-SAT WITH NO NEGATIONS. Given bipartite graph $G = (V, E)$ with $V = V_1 \cup V_2 V_1 = \{u_1, \cdots, u_k\}$,

$V_2 = \{v_1, \cdots, v_l\}$ define Boolean expression in $x_1, \cdots, x_k, y_1, \cdots, y_l$ by

$$f(x_1, \cdots, x_k, y_1, \cdots, y_l) = \bigwedge_{e=(u_i, y_j) \in E} (x_i \vee y_j).$$

Then $f(x_1, \cdots, x_k, y_1, \cdots, y_l)$ is true if and only if $\{u_i : x_i = T\} \cup \{v_j : y_j = T\}$ forms a cover of G. The reduction follows.

6. BIPARTITE INDEPENDENT SET $\propto$ MINIMUM CARDINALITY (s, t)-CUT. Given a bipartite graph $G = (V, E)$, $V = V_1 \cup V_2$, $E \subseteq V_1 \times V_2$, construct the graph G' with vertices $V \cup \{s, t\}$ and edges consisting of E along with sets M'_v of multiple edges of the type (s, v), $v \in V_1$ or (v, t), $v \in V_2$ with multiplicity equal to the degree of v in G. An example of this construction is given in Fig. 3. Now a minimum

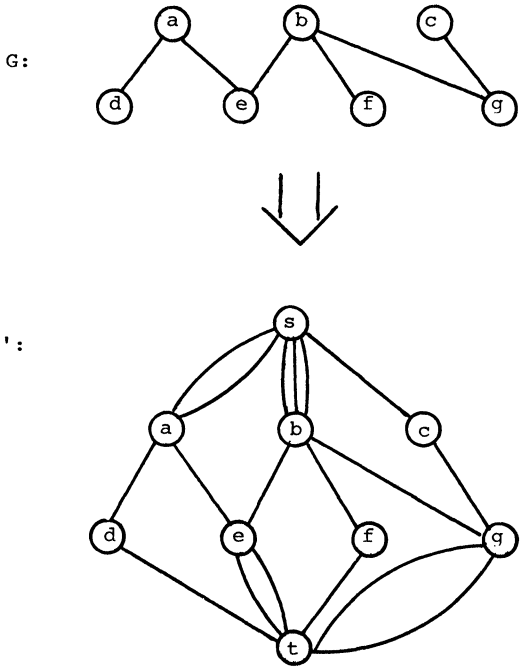


FIG. 3. *Example of transformation used in reduction* 6.

cardinality (s, t)-cut in G' is of cardinality $|E|$, since (a) E is an (s, t)-cut and (b) an (s, t)-flow of size $|E|$ can be obtained by directing all edges from s to t and giving each a flow of 1. It is clear that if a minimum cardinality (s, t)-cut C' of G' contains one edge of a set M'_v then it must contain every edge in M'_v. Further, if the edges (s, v) and (w, t) are in C', then (v, w) cannot be an edge, since we can obtain a cut with one less edge by replacing M'_v and M'_w with all edges in E adjacent to either v or w. Thus, the sets M'_v of C' have ends in G which are independent in G and the remaining edges in C' must be all those edges in E which do not have a vertex in common with these sets. Conversely, any set of edges of this type must be a minimum cardinality (s, t)-cut in G'. Thus, there is a one to one correspondence between minimum cardinality (s, t)-cuts in G' and independent sets in G. The reduction is now complete. Note that this reduction applies in both the directed and undirected cases.

The use of multiple edges could have been avoided but we omit the argument for the sake of simplicity.

7. DIRECTED MINIMUM CARDINALITY (s, t)-CUT $\propto$ MINIMUM CARDINALITY DIRECTED NETWORK CUT. Given directed graph $G = (V, E)$, $s, t \in V$, let k be the cardinality of a minimum cardinality (s, t)-cut. (It is well known that k can be calculated in polynomial time using a network flow algorithm.) Construct directed graph G' from G by adding multiple edges of the form (t, v) with multiplicity $k+1$ for each $v \in V - \{s, t\}$. Fig. 4 illustrates this transformation. Now any

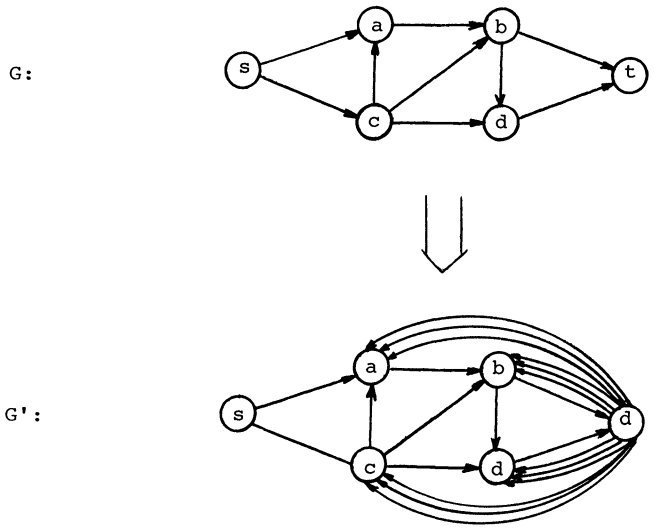


FIG. 4. *Example of transformation used in reduction* 7.

minimum cardinality (s, t)-cut in G remains a network cut in G' since all of the added edges point out of t. Thus, the size of a minimum cardinality network cut in G' is at most k. But since removal of any set S of at most k edges from E' must leave at least one edge from t to every vertex $x \neq s$ in V, then S is a network cut in G' if and only if S is an (s, t)-cut in G. Therefore, the minimum cardinality network cuts for G' are of cardinality k, and they consist precisely of sets of edges which are (s, t)-cuts for G. This completes the reduction. As in the previous argument, the use of multiple edges could have been avoided.

8A. MINIMUM CARDINALITY DIRECTED NETWORK $\propto$ DIRECTED CONNECTEDNESS RELIABILITY. Given $G = (V, E)$, we write, as in § 1,

$$g(G, s; p) = \sum_{j=0}^{n} g_j p^j (1-p)^{n-j} = (1-p)^n \sum_{j=0}^{n} g_j \left(\frac{p}{1-p}\right)^j, \tag{4}$$

where g_j is the number of sets of edges of cardinality j whose complement admits a path from s to every other vertex in G. Thus, $\bar{g}_j = \binom{n}{j} - g_j$ is the number of sets of edges of cardinality j that contain a directed network cut. Further, the matrix $B = (b_{ij})$ with $b_{ij} = (p_i/(1-p_i))^j$ for $i = 0, \cdots, m$, $j = 0, \cdots, n$ is Vandermonde for any choice $0 < p_0 \cdots < p_n < 1$. Therefore, by evaluating $g(G, s; p)/(1-p)^n$ for $i = 0, \cdots, n$, and

solving (4) we can obtain g_j, and hence $\bar{g}_j$ for $j=0,\cdots,n$. The value of the first nonzero $\bar{g}_j$ then solves the minimum cardinality directed network cut problem.

8B. MINIMUM CARDINALITY UNIDIRECTED (s,t)-CUT $\propto$ UNIDIRECTED CONNECTEDNESS RELIABILITY. Given undirected graph G, vertices s, t, write the network reliability polynomial of G with respect to s as above

$$g(G,s;p)=(1-p)^n\sum_{i=0}^{n} g_i\left(\frac{p}{1-p}\right)^i,$$

where g_i is the number of sets of cardinality i whose complement admits a path from every vertex to s. Consider now the graph G' obtained from G by replacing the vertices s and t with the vertex v'_{st} in every edge in which either appears. The network reliability polynomial of G' with respect to v'_{st} is

$$g(G',v'_{st};p)=\sum_{i=0}^{n} g'_i p^i(1-p)^{n-i}=(1-p)^n\sum_{i=0}^{n} g'_i\left(\frac{p}{1-p}\right)^{n-i}.$$

Now g'_i is the number of sets of edges in G' of cardinality i whose complement admits a path from every vertex of G' to v'_{st}, or equivalently, the number of sets of edges in G of cardinality i whose complement admits a path from every vertex of G to either s or t. Therefore, g'_i-g_i is the number of sets of edges in G of cardinality i whose complement admits a path from every vertex to s or t but does not admit a path from every vertex to *both* s and t. Such a set in particular contains an (s,t)-cut. Let k be the cardinality of a minimum cardinality (s,t)-cut. Then the complement of any set of k edges that contains an (s,t)-cut must allow a path from every vertex to either s or t (otherwise, an edge could be added to the component containing a vertex not connected to either s or t and still not allow a path from s or t). Thus g'_k-g_k is the number of minimum cardinality (s,t)-cuts in G. As in problem 8A, by evaluating $g(G,s;p_i)$ and $g(G',v'_{st};p_i)$ for $0<p_0<\cdots<p_n$, we can obtain g_i and g'_i for $i=0,\cdots,n$, and in particular, the value g'_k-g_k. This completes the reduction.

9. MINIMUM CARDINALITY NETWORK CUT $\propto$ CONNECTEDNESS RELIABILITY APPROXIMATION. Suppose we are given $G=(V,E)$ and $s\in V$. We produce this reduction by showing how to compute the g_i successively for $i=0,1,\cdots$, using as a subroutine an algorithm for the connectedness reliability approximation problem. Suppose we have computed g_i for $i=0,1,\cdots,k-1$; define

$$\alpha=\sum_{j=0}^{k-1} g_j p^j(1-p)^{n-j};$$

then for $0<p<1$ we have

$$\begin{aligned} g(G,s;p)-\alpha &= \sum_{j=k}^{n} g_j p^j(1-p)^{n-j} \\ &= p^k(1-p)^{n-k}\left[g_k+\frac{p}{1-p}\sum_{i=k+1}^{n} g_i\left(\frac{p}{1-p}\right)^{i-k-1}\right]. \end{aligned}$$

Using the fact that $0\leqq g_i\leqq\binom{n}{i}$ for $i=k+1,\cdots,n$, we obtain the inequalities

$$\frac{g(G,s;p)-\alpha}{p^k(1-p)^{n-k}}\geqq g_k$$

and

$$\frac{g(G,s;p)-\alpha}{p^k(1-p)^{n-k}}$$

$$\leqq g_k+\frac{p}{1-p}\sum_{i=k+1}^{n}\binom{n}{i}\left(\frac{p}{1-p}\right)^{i-k-1}$$

$$=g_k+\frac{p}{1-p}\sum_{i=k+1}^{n}\left[\frac{n!}{(n-k-1)!}\right]\left[\frac{(i-k-1)!}{i!}\right]\left[\binom{n-k-1}{i-k-1}\left(\frac{p}{1-p}\right)^{i-k-1}\right]$$

$$\leqq g_k+\frac{p}{1-p}\left[\frac{n!}{(n-k-1)!}\right]\left[\frac{1}{(k+1)!}\right]\left[\frac{1}{(1-p)^{n-k-1}}\right]$$

$$=g_k+\binom{n}{k+1}\frac{p}{(1-p)^{n-k}}.$$

Now if r is an ε-approximation to g, it follows for $0<p<1$ that

$$g_k\leqq\frac{r+\varepsilon-\alpha}{p^k(1-p)^{n-k}}=\frac{(r-\varepsilon)-\alpha+2\varepsilon}{p^k(1-p)^{n-k}}\leqq\frac{g(G,s;p)-\alpha+2\varepsilon}{p^k(1-p)^{n-k}}$$

$$\leqq g_k+\binom{n}{k+1}\frac{p}{(1-p)^{n-k}}+\frac{2\varepsilon}{p^k(1-p)^{n-k}}$$

$$=g_k+\frac{1}{(1-p)^{n-k}}\left[\binom{n}{k+1}p+\frac{2\varepsilon}{p^k}\right],$$

so that, if we choose

$$p=\min\left\{1-2^{-1/(n-k)},\frac{1}{2}\binom{n}{k+1}^{-1}\right\}$$

and $\varepsilon=p^k/4$, then

$$\frac{1}{(1-p)^{n-k}}\left[\binom{n}{k+1}p+\frac{2\varepsilon}{p^k}\right]<\frac{1}{1/2}\left[\binom{n}{k+1}\frac{1}{2}\binom{n}{k+1}^{-1}+\frac{p^k/2}{p^k}\right]=2\left[\frac{1}{2}+\frac{1}{2}\right]=1.$$

Hence,

$$g_k=\left\lfloor\frac{r+\varepsilon-\alpha}{p^k(1-p)^k}\right\rfloor.$$

The proof is now complete.

10. MINIMUM CARDINALITY (s,t)-CUT $\propto$ (s,t)-CONNECTEDNESS RELIABILITY APPROXIMATION. The reduction here is identical to that in problem 9. This completes the proof of the theorem.

3. Further discussion. We remark that problems 9 and 10 easily show the #P-completeness of the *α-approximation problem* (see [11], called the *point estimate* problem in [1] for the functions g and f. This problem is: given $\alpha<1$, $0\leqq p\leqq 1$, find a number r such that $\alpha r<g(G,s;p)$ (respectively $f(G,s,t;p))<r/\alpha$. We should note that a seemingly more difficult unsolved problem involves the case where α (or ε) is constant, i.e. is not allowed to vary as part of the input list.

We complete our discussion by considering the complexity of certain reliability and counting problems for two special classes of graphs. One class is that of directed

acyclic graphs, that is, graphs that have no closed (directed) paths. For these graphs the minimum cardinality (s, t)-cut problem (6) still remains # P-complete since the network constructed in the proof of the theorem is acyclic; hence, the (s, t)-connectedness problem (10) for acyclic graphs remains # P-complete. The directed network cut problem (7), however, is polynomial, and, in fact, the connectedness reliability problems (8 and 9) are also polynomial (see [3]). The second class of graphs is that of planar graphs (directed and undirected). Here, both the minimum cardinality network cut problem and the minimum cardinality (s, t)-cut problem are polynomial (see also [3]). The complexity of the reliability problems, however, are open questions. Table 2 summarizes known results for these classes of graphs.

TABLE 2

	Min. card. pathset	Min. card cutset	Rel. poly.	Rel. approx.
directed acyclic two terminal	* [3]	! TH	! TH	! TH
directed acyclic network	* [3]	* [3]	* [3]	* [3]
undirected and directed planar two terminal	* [3]	* [3]	?	?
undirected and directed planar network	* [10]	* [3]	?	?

The table entries have the same interpretation as those in Table 1.

REFERENCES

[1] M. O. BALL, *The complexity of network reliability computations*, Networks, 10 (1980), pp. 153–165.

[2] M. O. BALL AND J. S. PROVAN, *Bounds on the reliability polynomial for shellable independence systems*, SIAM J. Alg. Discr. Meth., 3 (1982), pp. 166–181.

[3] ———, *Calculating bounds on reachability and connectedness in stochastic networks*, Networks, 13 (1983), pp. 253–278.

[4] Z. GALIL, *On some direct encodings of nondeterministic Turing machines operating in polynomial time into* P*-complete problems*, SIGACT News, 6, 1 (1974), pp. 19–24.

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

[6] J. N. HAGSTROM, *Computing rooted communication reliability is* # P*-complete*, unpublished manuscript, 1981.

[7] G. H. HARDY, J. E. LITTLEWOOD AND G. POLYA, *Inequalities*, Cambridge Univ. Press, Cambridge, 1952.

[8] A. S. HOUSEHOLDER, *Principles of Numerical Analysis*, McGraw-Hill, New York, 1953.

[9] M. JERRUM, *On the complexity of evaluating multivariate polynomials*, Ph.D. thesis, Tech. Rep. CST-11-81, Dept. Computer Science, Univ. Edinburgh, 1981.

[10] G. KIRCHHOFF, *Über die Auflosung der Gleichungen, auf welche man sei der Untersuchung der linearen Verteilung Galvanischer Strome gerfuhrt wird*, Poggendorf's Ann. Phy. Chem., 72 (1847), pp. 497–508; English translation, IRE Trans. Circuit Theory, 5 (1958), pp. 4–8.

[11] A. ROSENTHAL, *Computing the reliability of complex networks*, SIAM J. Applied Math, 32 (1977), pp. 384–393.

[12] L. G. VALIANT, *The complexity of enumeration and reliability problems*, this Journal, 8 (1979), pp. 410–421.

SIAM J. COMPUT.
Vol. 12, No. 4, November 1983

0097-5397/83/1204-0014 501.25/0

ASYMPTOTIC EXPANSIONS OF MOMENTS OF THE WAITING TIME IN A SHARED-PROCESSOR OF AN INTERACTIVE SYSTEM*

DEBASIS MITRA† AND J. A. MORRISON†

Abstract. An interactive computer system's service is characterized by the random waiting (or response) time perceived by users. This paper presents a novel solution to the problem of efficiently computing the second moment of the *waiting time* of a class of *large* interactive systems. The physical system consists of a bank of terminals, each of which asynchronously alternates between "thinking" and waiting for service from a CPU which operates under the processor-sharing discipline. The problem of obtaining higher order waiting time moments is quite different from that of obtaining CPU queue statistics. Only the first moment of the waiting time is obtainable from the CPU queue statistics.

The technique for arriving at the second moment of the waiting time consists of developing an asymptotic expansion in inverse powers of the number of terminals. Hence, as the system grows, quite fortuitously, fewer terms of the series require computation to achieve the desired accuracy. Beside its numerical advantages, the results give new insight since the leading terms of the series, which contain most of the information, are obtained explicitly. The novelty also rests on the fact that instead of solving matrix equations, the problem is turned into one of solving a second order differential equation. A simple two-dimensional recursion yields all the terms of the asymptotic expansion.

Key words. queueing networks, queueing theory, asymptotic expansions, waiting time moments

1. Introduction. An interactive computer system's service is characterized by the random waiting (or response) time perceived by users of the system. This paper presents a novel approach to the problem of efficiently computing the first and second moments of the waiting time for *large* interactive systems. The physical system, see Fig. 1, consists of a bank of user terminals in series with a CPU which feeds back to the terminals. Each user spends alternating time periods in the "think" mode and the "waiting" mode; in the former, the user takes an independent amount of time to generate jobs with random service time requirements, while in the waiting mode the job, now transferred to the CPU, contends with other jobs for service. On completion of service the job returns to the terminal and a new cycle resumes.

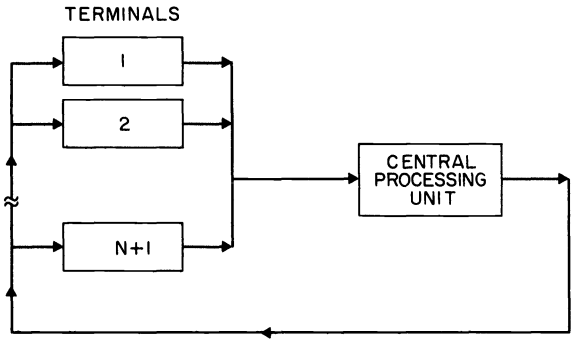
The waiting time distribution for such a model has been considered in [1] and the moments have been given there in terms of the solution of a matrix equation. These equations have dimension $(N+1)$, where $(N+1)$ is the number of user terminals. Practical interest is focused on large systems, i.e. large N, and in this case the equations pose a computational challenge which is compounded by the equation's worsening conditioning with increasing system usage. Also, insight into the nature of solutions is less readily forthcoming. In this paper we give a quite novel technique for arriving at the second moment $E[W^2]$. Given here is an asymptotic expansion for $E[W^2]$ in inverse powers of N, so that the larger N is, fewer terms of the series require computation to achieve the desired degree of accuracy. Also, it is possible to give explicitly and simply the leading terms which contain most of the information and are most amenable to interpretation.

The novelty of the technique also rests on the fact that, instead of inverting matrices, we transform the problem into a differential equation for the generating

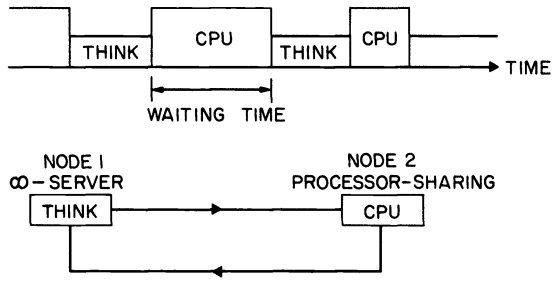
*Received by the editors September 1, 1982, and in revised form December 21, 1982. This paper was typeset by Carmela Patuto at Bell Laboratories, Murray Hill, New Jersey, using the troff program running under the Unix™ operating system. Final copy was produced on April 4, 1983.

†Bell Laboratories, Murray Hill, New Jersey 07974.

function of the matrix quantities. The differential equation is linear and only of second order, but is still not nice (a reflection of the nontrivial nature of the problem) in that its solution is not a classical special function. Nevertheless, we are able to use



(a) THE PHYSICAL SYSTEM



(b) JOBS SPEND ALTERNATE PERIODS OF TIME IN THE THINK NODE AND PROCESSOR-SHARING CPU NODE.

FIG. 1

certain properties of the solution to obtain its asymptotic expansion. The simplicity of the solution technique obtained here is reflected in the fact that a two-dimensional linear recursion yields the coefficients of the expansion.

1.1 The model, parameters and stationary distribution. The model and problem have three distinguishing features; first, the CPU's discipline is "processor sharing" [2]-[4], [8]-[12]; second, the network is closed [2]-[4]; finally, the problem concerns not the stationary distribution of jobs in the CPU, but the equilibrium waiting time distribution, where waiting time is defined to be the interval from time of entry to the CPU to time of exit.

In the processor sharing discipline there is no overt queueing because all, say n, jobs present in the CPU simultaneously receive $1/n$ times the rate of service which a solitary job in the CPU would receive. Therefore, the rate of service received by a specific job fluctuates with time and, importantly, its waiting time depends not only on the jobs in the CPU at its time of arrival there, but also on subsequent arrivals. This

makes the processor sharing discipline intrinsically harder to analyze than, say, the first-come-first-served discipline.

The network is closed so that the total number of jobs in the think and the processing states is constant at $(N+1)$, the number of terminals. We refer to $(N+1)$ as the population. Notice that on account of the network closure the job stream arriving at the CPU is not characterized by any nice stochastic properties.

The required service times for all jobs are independent, identically and exponentially distributed. The unit of time is selected to give unity as the mean required service time. The "think" times are also independent, identically and exponentially distributed with mean $1/p$. Thus

(1) $$p = \text{Mean required service time/Mean think time.}$$

We also define

(2) $$\rho \triangleq Np,$$

and assume throughout this paper that

(3) $$\rho < 1.$$

This assumption is equivalent to the "normal usage" assumption in [5].

We give next the known results on the equilibrium distribution of jobs in the CPU. As the analysis of the waiting time problem for a population of $(N+1)$ requires as a prerequisite the stationary distribution of jobs with a population of N (cf. [1]), it is the latter that is given here.

For $i = 0,1,\cdots,N$ let

π_i = equilibrium probability of i jobs being in CPU for population of N

(4) $$= \binom{N}{i} i!p^i/G(N),$$

where

(5) $$G(N) = \sum_{j=0}^{N} \binom{N}{j} j!p^j$$

is the normalizing constant. It is also referred to as the *partition function* (of N).

2. Moments of the waiting time distribution.

2.1 Known results. It is proven in [1] that

(6) $$\frac{1}{2} E[W^2] = \mathbf{c}'\mathbf{b},$$

where $\mathbf{c}$ is the solution of the following matrix equation,

(7) $$\mathbf{c}'[\mathbf{I}-\mathbf{A}] = \boldsymbol{\pi}'\mathbf{B}.$$

Here $\boldsymbol{\pi}' = [\pi_0,\pi_1,\cdots,\pi_N]$ and $\mathbf{b}' = [1,2,\cdots,N+1]$; $\mathbf{B}$ and $\mathbf{I}$ are $(N+1)$-dimensional matrices where $\mathbf{B} = \text{diag}\{\mathbf{b}\}$ and $\mathbf{I}$ is the identity matrix. The matrix $\mathbf{A}$ is a generator of a birth-and-death Markov process and is therefore tridiagonal. (The reader is cautioned that this birth-and-death process is not the natural process associated with

the jobs in the CPU, cf. [1].) For $i = 0,1,\cdots,N$,

$$A_{i,i-1} = i \quad , \quad A_{i,i+1} = \left(1 - \frac{i}{N}\right)(i+1)\rho,$$

(8)

$$A_{i,i} = -\{A_{i,i-1} + A_{i,i+1}\}.$$

The matrix [I-A] in (7) is invertible by Proposition 4 of [1]. Consequently, there exists a unique solution $\mathbf{c}$ to (7). This uniqueness property will play an important role in §4.2.

It is also known [1] that

$$E[W] = \mathbf{c}'\mathbf{1}. \tag{9}$$

Recently Gaver, Jacobs and Latouche [6] have considered the model described above and its extensions. The central theme of their work is that in heavy traffic the response time for a long job is approximately normally distributed.

The open network version of our problem in which a Poisson stream of jobs is submitted to a processor-sharing CPU is analyzed in [8]. Also of interest in this connection are [9] and [10]. Certain important natural generalizations of the processor-sharing discipline may be found in [11] and [12].

2.2 Generating functions. Let us define the following generating function of the stationary probabilities:

$$G(z;N) = \left(\sum_{i=0}^{N} \pi_i z^i\right) G(N), \tag{10}$$

so that in particular,

$$G(1;N) = G(N). \tag{11}$$

Thus the above generating function is a natural generalization of the partition function. A methodology for analyzing partition functions of a large class of networks is given in [5]. Using certain notions from [5] we find that the generating function has the following integral representation:

$$G(z;N) = \int_0^\infty e^{-u}\left(1 + \frac{\rho}{N}zu\right)^N du. \tag{12}$$

To arrive at (12) substitute

$$i! = \int_0^\infty e^{-u}u^i\, du \tag{13}$$

in (4) and use the binomial theorem.

It will also be convenient to define another related generating function,

$$H(z;N) = zG(z;N). \tag{14}$$

Note that

$$G(N) = H(1;N) \tag{15}$$

and

$$H'(z;N) = \left\{\sum_{i=0}^{N} (i+1)\pi_i z^i\right\} G(N) \tag{16}$$

where, as in the rest of the paper, the derivative is with respect to the first argument.

Now let

$$C(z;N) = \left\{\sum_{i=0}^{N} c_i z^i\right\} G(N)\,, \tag{17}$$

where $\mathbf{c}$ is the sought after solution of (7). By multiplying the i^{th} component equation of (7) by z^i and summing with respect to i we obtain the differential equation which will provide the basis for most of the analysis in this paper:

$$\frac{\rho}{N}(1-z)z^2C''(z;N) + (1-z)\left\{1-\rho z+\frac{2\rho z}{N}\right\}C'(z;N) - \{1+\rho(1-z)\}C(z;N) = -H'(z;N)\,. \tag{18}$$

The homogeneous version of the above differential equation has singularities at 0,1 and ∞ of ranks 1,0 and 0, respectively [7]. For this reason we are unable to express the solution of even the homogeneous version of (18) in terms of standard special functions.

We note that the first two moments of the waiting time are easily obtained from the solutions of (18): from (6) and (9),

$$\begin{aligned} \frac{1}{2}E[W^2] &= \{C(1;N) + C'(1;N)\}/G(N); \\ E[W] &= C(1;N)/G(N). \end{aligned} \tag{19}$$

Moreover, we note from (14) and (18) that

$$C(1;N) = H'(1;N) = G(1;N) + G'(1;N)\,.$$

The above concludes for the present the discussion on the generating function $C(z;N)$ of the vector $\mathbf{c}$. We return now to $G(z;N)$, defined in (10) as the generating function of $\boldsymbol{\pi}$, and show that it too satisfies a related differential equation.

A lemma in [1] states that $\boldsymbol{\pi}'\mathbf{A} = \mathbf{0}'$, which may be compared to (7). By a derivation analogous to that of (18), it follows that

$$\frac{\rho}{N}z^2G''(z;N) + \left[1-\rho z+\frac{2\rho z}{N}\right]G'(z;N) - \rho G(z;N) = 0.$$

This equation may be integrated once to obtain

$$\frac{\rho}{N}z^2G'(z;N) + (1-\rho z)G(z;N) = 1, \tag{20}$$

where we have used the fact that $G(0;N) = \pi_0 G(N) = 1$. Equation (20) may also be obtained directly from the birth-and-death equations given as $\boldsymbol{\pi}'\mathbf{F} = \mathbf{0}'$ in [1].

We remark that (20) leads directly to the integral representation (12) for $G(z;N)$. Equation (20) is interesting in its own right for it suggests a new avenue for

calculating the partition function. Indeed, some of the relevant asymptotic expansions in §3, which rely for their derivation on the integral representation, are also obtained in §4.4 from (20).

2.3 An outline of the procedure for the moments' asymptotic expansions. An outline of the procedure is given here and subsequent sections will provide the necessary clarifications.

(i) Since we are interested in the behavior of $C(z;N)$ for z in the neighborhood of 1, it is convenient to make the transformation

$$f(u;N) \triangleq C\left(1-\frac{u}{N};N\right). \tag{21}$$

From (18),

$$\rho\left(1-\frac{2u}{N}+\frac{u^2}{N^2}\right)uf''(u;N) - \left\{(1-\rho)+\frac{\rho}{N}(2+u)-\frac{2\rho u}{N^2}\right\}uf'(u;N) - \left(1+\frac{\rho u}{N}\right)f(u;N) = -H'\left(1-\frac{u}{N};N\right). \tag{22}$$

(ii) The right-hand side has an asymptotic expansion

$$H'\left(1-\frac{u}{N};N\right) \sim \sum_{r=0}^{\infty} J_r(u)/N^r \quad \text{as } N \to \infty. \tag{23}$$

The results in §§3, 4.1 and 4.4 give $J_r(u)$ explicitly. It is shown that $J_r(u)$ is a polynomial of degree r in u.

(iii) Let

$$f(u;N) \sim \sum_{r=0}^{\infty} f_r(u)/N^r \quad \text{as } N \to \infty. \tag{24}$$

On substituting (23) and (24) in (22) and upon matching coefficients of $1/N^r$ we obtain the following infinite set of differential equations to be satisfied by $\{f_r(u)\}$:

$$\rho u f_0'' - (1-\rho)uf_0' - f_0 = -J_0(u), \tag{25}$$

$$\rho u f_1'' - 2\rho u^2 f_0'' - (1-\rho)uf_1' - \rho(2+u)uf_0' - f_1 - \rho u f_0 = -J_1(u), \tag{26}$$

and, for $r \geqslant 2$,

$$\rho u f_r'' - 2\rho u^2 f_{r-1}'' + \rho u^3 f_{r-2}'' - (1-\rho)uf_r' - \rho(2+u)uf_{r-1}' + 2\rho u^2 f_{r-2}' - f_r - \rho u f_{r-1} = -J_r(u). \tag{27}$$

(iv) In §4.2 we show that the solution $f_r(u)$, like $J_r(u)$, is a polynomial in u of degree r:

$$f_r(u) = \sum_{l=0}^{r} b_{rl} u^l \quad , \quad r = 0,1,2, \cdots \quad . \tag{28}$$

On substituting this form and the previously derived polynomial form for $J_r(u)$ in (25)-(27) and then matching the coefficients of powers of u, a set of relations is obtained which must be satisfied by $\{b_{rl}\}$. These key relations given in (59) are in the form of recursions and, being complete, they constitute a procedure for computing $\{b_{rl}\}$.

(v) Finally, from (19), (21), (24) and (28),

$$\frac{1}{2} E[W^2] \sim \frac{1}{G(N)} \sum_{r=0}^{\infty} (b_{r0} - b_{r+1,1})/N^r, \tag{29}$$

$$E[W] \sim \frac{1}{G(N)} \sum_{r=0}^{\infty} b_{r0}/N^r. \tag{30}$$

The asymptotic expansion for $G(N)$, which is known from previous results [5], is also available from the generalized framework of §3. This completes the procedure for generating the asymptotic expansions for the first and second moments.

3. Asymptotic expansion of the generating function of the stationary distribution. The reader should view this section as a digression from the main theme of this paper. However, the results obtained here are essential for the subsequent derivation of the main result of this paper. We have also chosen to collect certain results here since they follow in a natural manner from the framework established in [5]. Indeed these results represent a conceptual generalization of the framework in [5]. However, the results in [5] are for a large class of networks while here we have under consideration the network in Fig. 1.

3.1 Generating functions. Briefly, a procedure is given in [5] for obtaining the asymptotic expansion of the partition function $G(N)$, i.e. a procedure for generating the coefficients $\{G_r\}$ where

$$G(N) \sim \sum_{r=0}^{\infty} G_r/N^r \quad , \quad \text{as} \quad N \to \infty. \tag{31}$$

The method of [5] is applicable for the case of "normal usage", which here is equivalent to $\rho < 1$, as in (3). Here we use these techniques to arrive at the asymptotic expansion

$$H(z;N) = zG(z;N) \sim \sum_{r=0}^{\infty} H_r(z)/N^r, \quad \text{as} \quad N \to \infty. \tag{32}$$

The above subsumes (31) since $G_r = H_r(1)$. We will find no need for detailed proofs regarding asymptoticity since the proofs in [5] will suffice.

Recall from (12) and (14) that

$$H(z;N) = z \int_0^{\infty} e^{-u} \left(1 + \frac{\rho}{N} uz\right)^N du \tag{33}$$

$$= z \int_0^{\infty} e^{-\alpha(z)u} \left\{ e^{-\rho z u} \left(1 + \frac{\rho z u}{N}\right)^N \right\} du \tag{34}$$

where we have introduced an important parameter

$$\alpha(z) \triangleq 1 - \rho z. \tag{35}$$

We shall require that

$$\alpha(z) > 0\,, \tag{36}$$

which however will allow $z \in [0,1]$, since $\rho < 1$. Note that the parameter α of [5] is simply $\alpha(1)$.

A change of variables in (34) to $v = \alpha(z)u$ gives

$$H(z;N) = \frac{z}{\alpha(z)} \int_0^\infty e^{-v} h(v,z,N^{-1})\, dv\,, \tag{37}$$

where

$$h(v,z,N^{-1}) = e^{-\tilde{\Gamma}(z)v} \left(1 + \frac{\tilde{\Gamma}(z)v}{N}\right)^N, \tag{38}$$

and

$$\tilde{\Gamma}(z) = \frac{\rho z}{\alpha(z)}\,. \tag{39}$$

Again $\tilde{\Gamma}(z)$ is the natural generalization of $\tilde{\Gamma}$ in [5]:

$$\tilde{\Gamma} = \tilde{\Gamma}(1) = \frac{\rho}{1-\rho}\,. \tag{40}$$

The parameter $\tilde{\Gamma}$ will be familiar from single-server queueing theory.

The procedure for obtaining the asymptotic expansion for $H(z;N)$ as $N \to \infty$, namely,

$$H(z;N) \sim \sum_{r=0}^{\infty} H_r(z)/N^r\,, \tag{41}$$

consists of first obtaining a power series for $h(v,z,N^{-1})$,

$$h(v,z,N^{-1}) = \sum_{r=0}^{\infty} h_r(v,z)/N^r\,, \tag{42}$$

and then integrating term by term to obtain

$$H_r(z) = \frac{z}{\alpha(z)} \int_0^\infty e^{-v} h_r(v,z)\, dv. \tag{43}$$

The functions $h_r(v,z)$ are the coefficients in a Taylor series in N^{-1} of $h(v,z,N^{-1})$ which is obtained directly from (38). It may be verified that

$$\begin{aligned} &h_0(v,z) \equiv 1\,, \\ &h_{r+1}(v,z) = -\frac{1}{r+1} \sum_{m=0}^{r} \frac{(r+1-m)}{(r+2-m)} \{-\tilde{\Gamma}(z)v\}^{r+2-m} h_m(v,z), \quad r = 0,1,2,\cdots\,. \end{aligned} \tag{44}$$

This recursive formula shows that $h_r(v,z)$ is a simple polynomial of degree $2r$ in v

and, consequently, the integration in (43) is easy to carry out. This enables $H_r(z)$ to be obtained in a straightforward manner. In particular,

$$
\begin{aligned}
H_r(z) &= \tilde{\Gamma}(z)/\rho, \quad r = 0 \\
&= -\tilde{\Gamma}^3(z)/\rho, \quad r = 1 \\
&= \{2\tilde{\Gamma}^4(z)+3\tilde{\Gamma}^5(z)\}/\rho, \quad r = 2 \\
&= -\{6\tilde{\Gamma}^5(z)+20\tilde{\Gamma}^6(z)+15\tilde{\Gamma}^7(z)\}/\rho, \quad r = 3.
\end{aligned} \tag{45}
$$

A pattern in the above expressions which prevails generally is that $H_r(z)$, $r \geqslant 0$, is a polynomial in $\tilde{\Gamma}(z)$ of degree $(2r+1)$, with coefficients which are inversely proportional to ρ.

This completes the description of the procedure for systematically generating the asymptotic expansion for $H(z;N)$.

We conclude by observing that later in the paper we shall require quantities of the form $H_r^{(k)}(1)$, i.e.

$$
\begin{array}{l}
H_0^{(1)}(1) \\
H_1^{(1)}(1)\ H_0^{(2)}(1) \\
H_2^{(1)}(1)\ H_1^{(2)}(1)\ H_0^{(3)}(1) \\
\quad .\qquad .\qquad .
\end{array} \tag{46}
$$

Of course computing the above array to the depth desired is straightforward once the functions $\{H_r(z)\}$ have been obtained. In particular,

$$
\begin{aligned}
&H_0^{(k)}(1) = \frac{k!}{\rho^2}\tilde{\Gamma}^{k+1}, \quad k \geqslant 1, \\
&H_1^{(1)}(1) = -\frac{3}{\rho^2}\tilde{\Gamma}^4, \quad H_1^{(2)}(1) = -\frac{6}{\rho^2}\tilde{\Gamma}^4 - \frac{12}{\rho^2}\tilde{\Gamma}^5, \\
&H_2^{(1)}(1) = \frac{8}{\rho^2}\tilde{\Gamma}^5 + \frac{15}{\rho^2}\tilde{\Gamma}^6.
\end{aligned} \tag{47}
$$

In §4.4 we will give an alternate, and also complete *recursive* procedure for generating the above quantities. This procedure treats $G(z;N)$ as given by (20) in a manner similar to the treatment in §2.3 for $C(z;N)$ as given by (18).

3.2 Partition function. We have already stated that the above procedure for generating the asymptotic expansion for the function $H(z;N)$ subsumes the procedure for obtaining the asymptotic expansion for the partition function $G(N)$. To make this quite explicit and also because the asymptotic expansion for $G(N)$ is required later in the paper, we now give the leading terms.

Recall from (32) that

$$
G(N) = H(1,N) \sim \sum_{r=0}^{\infty} H_r(1)/N^r, \quad \text{as } N \to \infty \tag{48}
$$

where, from (45),

$$
\begin{aligned}
H_r(1) &= \tilde{\Gamma}/\rho, \quad r = 0 \\
&= -\tilde{\Gamma}^3/\rho, \quad r = 1 \\
&= (2\tilde{\Gamma}^4+3\tilde{\Gamma}^5)/\rho, \quad r = 2 \\
&= -(6\tilde{\Gamma}^5+20\tilde{\Gamma}^6+15\tilde{\Gamma}^7)/\rho, \quad r = 3.
\end{aligned} \tag{49}
$$

4. Asymptotic expansions for waiting time moments. This section presents the main results of the paper.

4.1 The forcing terms. Consider in detail the term $H'(1-u/N;N)$ in (22). Recall that $H(z;N)$ is a polynomial of degree $(N+1)$ in z but $H_r(z)$ is not.

$$
\begin{aligned}
H'\left(1-\frac{u}{N};N\right) &= \sum_{k=0}^{N} \frac{H^{(k+1)}(1;N)}{k!}\left(\frac{-u}{N}\right)^k = \sum_{k=0}^{\infty} \frac{H^{(k+1)}(1;N)}{k!}\left(\frac{-u}{N}\right)^k \\
&\sim \sum_{k=0}^{\infty} \frac{1}{k!}\left(\frac{-u}{N}\right)^k \sum_{s=0}^{\infty} \frac{H_s^{(k+1)}(1)}{N^s} \\
&\sim \sum_{r=0}^{\infty} \frac{1}{N^r}\left\{\sum_{k=0}^{r} \frac{(-1)^k H_{r-k}^{(k+1)}(1)}{k!} u^k\right\},
\end{aligned} \tag{50}
$$

i.e

$$
H'\left(1-\frac{u}{N};N\right) \sim \sum_{r=0}^{\infty} J_r(u)/N^r, \tag{51}
$$

where

$$
J_r(u) = \sum_{k=0}^{r}\left\{\frac{(-1)^k H_{r-k}^{(k+1)}(1)}{k!}\right\} u^k, \quad r = 0,1,2,\cdots. \tag{52}
$$

It has been shown in §3.1 that the quantities $H_{r-k}^{(k+1)}(1)$ may be systematically obtained. Thus the r^{th} degree polynomials $J_r(u)$, $r = 0,1,2,\cdots$ are known quantities.

4.2 The recursions on the expansion coefficients. The nonhomogeneous differential equations in (25)-(27) may be rendered into the homogeneous form by transforming the dependent variables from $\{f_r(u)\}$ to

$$
f_r(u) - \sum_{l=0}^{r} b_{rl} u^l \tag{53}
$$

by an appropriate choice of the constants $\{b_{rl}\}$ which will depend linearly on the coefficients of the polynomials $\{J_r(u)\}$. These observations are simple corollaries of the form in (25)-(27) and the fact that $J_r(u)$ is a polynomial in u of degree r.

The homogenized differential equations certainly admit the trivial solutions

$$f_r(u) - \sum_{l=0}^{r} b_{rl}u^l \equiv 0. \tag{54}$$

We now claim this as the correct solution because of the uniqueness property, discussed previously in §2.1, of the solution vector c in the original equation (7). We also arrive at this conclusion from the fact that the nontrivial solutions of the homogenized equation are inadmissible. This is so because it can be shown that one solution of the homogenized version of (25) grows exponentially with u, and the other solution has an infinite derivative at $u = 0$. We have arrived at an important conclusion, namely, $f_r(u)$, like $J_r(u)$, is a polynomial in u of degree r.

The following procedure allows the quantities $\{b_{rl}\}$ in (54) to be identified. If we substitute in (25)-(27) the forms for $\{f_r(u)\}$ and $\{J_r(u)\}$ in (54) and (52) respectively, we obtain polynomials which must vanish for all u. This procedure is undertaken below.

From (25),

$$b_{00} = H_0^{(1)}(1). \tag{55}$$

From (26),

$$b_{10} = H_1^{(1)}(1), \tag{56}$$

$$b_{11} = -\frac{1}{2-\rho}\left\{H_0^{(2)}(1)+\rho H_0^{(1)}(1)\right\}. \tag{57}$$

From (27), for $r \geqslant 2$,

$$\begin{aligned}
&\rho u \sum_{l=0}^{r} l(l-1)b_{rl}u^{l-2} - 2\rho u^2 \sum_{l=0}^{r-1} l(l-1)b_{r-1,l}u^{l-2} + \rho u^3 \sum_{l=0}^{r-2} l(l-1)b_{r-2,l}u^{l-2} \\
&- (1-\rho)u \sum_{l=0}^{r} lb_{rl}u^{l-1} - \rho(2u+u^2) \sum_{l=0}^{r-1} lb_{r-1,l}u^{l-1} + 2\rho u^2 \sum_{l=0}^{r-2} lb_{r-2,l}u^{l-1} \\
&- \sum_{l=0}^{r} b_{rl}u^l - \rho u \sum_{l=0}^{r-1} b_{r-1,l}u^l = \sum_{k=0}^{r} \left\{\frac{(-1)^{k+1}}{k!}H_{r-k}^{(k+1)}(1)\right\} u^k.
\end{aligned} \tag{58}$$

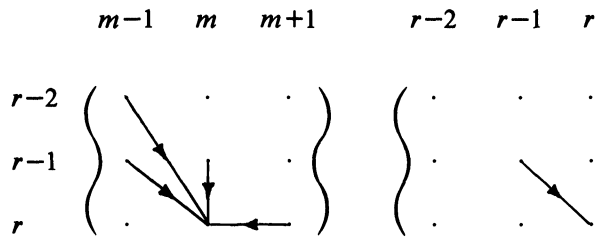
After rearranging and equating coefficients of u^m, $m = 0,1,\cdots,r$ in (58) we obtain, for $r \geqslant 2$,

$$\{1+(1-\rho)r\}b_{rr} = -\rho r b_{r-1,r-1} + \frac{(-1)^r}{r!} H_0^{(r+1)}(1);$$

$$\begin{aligned}
\{1+(1-\rho)m\}b_{rm} &= \rho m(m+1)b_{r,m+1} - 2\rho m^2 b_{r-1,m} + \rho m(m-1)b_{r-2,m-1} \\
&\quad -\rho m b_{r-1,m-1} + \frac{(-1)^m}{m!} H_{r-m}^{(m+1)}(1), \quad m = 0,1,2,\cdots,r-1.
\end{aligned} \tag{59}$$

The above recursions together with the end values in (55)-(57) form a complete system. Notice that (55)-(57) imply that (59) holds also for $r = 0$ and $r = 1$.

We may schematically represent the dependencies in the above recursions thus:



4.3 The dominant terms in the moments' asymptotic expansion. Recall from the outline in §2.3 and (29)-(30) in particular that

$$\frac{1}{2}E[W^2] \sim \frac{1}{G(N)} \sum_{r=0}^{\infty} \{b_{r0}-b_{r+1,1}\}/N^r , \tag{29}$$

$$E[W] \sim \frac{1}{G(N)} \sum_{r=0}^{\infty} b_{r0}/N^r . \tag{30}$$

Also, from (48),

$$G(N) \sim \sum_{r=0}^{\infty} H_r(1)/N^r.$$

In particular therefore,

$$\frac{1}{2}E[W^2] \sim \frac{1}{H_0(1)} \left[(b_{00}-b_{11}) + \frac{1}{N}\left\{(b_{10}-b_{21}) - \frac{H_1(1)}{H_0(1)}(b_{00}-b_{11})\right\}\right], \tag{60}$$

and

$$E[W] \sim \frac{1}{H_0(1)} \left[b_{00} + \frac{1}{N}\left\{b_{10} - \frac{H_1(1)}{H_0(1)} b_{00}\right\}\right]. \tag{61}$$

After some straightforward calculations we obtain

$$\frac{1}{2}E[W^2] \sim \frac{2}{(1-\rho)^2(2-\rho)} - \frac{1}{N}\frac{2\rho^2(13-8\rho)}{(1-\rho)^4(2-\rho)^2} , \tag{62}$$

and

$$E[W] \sim \frac{1}{1-\rho} - \frac{1}{N}\frac{2\rho^2}{(1-\rho)^3} . \tag{63}$$

We note that the first term in (62) and the first term in (63) give the exact known results for the open network [8] in which a Poisson stream of jobs with rate parameter ρ is offered to the processor-sharing CPU. This observation is in agreement with the previously established [1, §3] correspondence between open networks and a limiting case of the closed network.

4.4 An alternate derivation of the forcing terms. We now consider an alternative to the procedure in §§3.1 and 4.1 for calculating the coefficients in the polynomials $J_r(u)$ defined by (23). As given by (52), these coefficients involve the quantities $H_{r-k}^{(k+1)}(1)$, to obtain which the procedure in §3.1 has been presented. In the technique given here, these quantities are obtained from a simple two-dimensional linear recursion. Thus, the primary recursion in (59) may be coupled with the recursion given below in (73); together with (76), they form an alternate, completely recursive solution.

The starting point is (20), in which we let

$$g(u;N) \triangleq G\left[1-\frac{u}{N};N\right], \tag{64}$$

so that

$$\rho\left[1-\frac{2u}{N}+\frac{u^2}{N^2}\right]g'(u;N)-\left[1-\rho+\frac{\rho u}{N}\right]g(u;N) = -1. \tag{65}$$

Let

$$g(u;N) \sim \sum_{r=0}^{\infty} g_r(u)/N^r \quad \text{as } N \to \infty. \tag{66}$$

Then, from (65), we obtain the following infinite set of differential equations to be satisfied by $\{g_r(u)\}$:

$$\rho g_0' - (1-\rho)g_0 = -1; \tag{67}$$

$$\rho g_1' - (1-\rho)g_1 = 2\rho u g_0' + \rho u g_0; \tag{68}$$

and, for $r \geqslant 2$,

$$\rho g_r' - (1-\rho)g_r = 2\rho u g_{r-1}' + \rho u g_{r-1} - \rho u^2 g_{r-2}'. \tag{69}$$

A solution of (67) is

$$g_0(u) = \frac{1}{(1-\rho)} = \frac{\tilde{\Gamma}}{\rho}. \tag{70}$$

Since the solution of the homogeneous equation grows exponentially, (70) is the desired solution. It may be shown by induction that $g_r(u)$ is a polynomial in u of degree r:

$$g_r(u) = \sum_{l=0}^{r} g_{rl}u^l. \tag{71}$$

It follows from (68), (70) and (71) that

$$g_{00} = \tilde{\Gamma}/\rho \ , \ g_{11} = -\tilde{\Gamma}^2/\rho, \ g_{10} = -\tilde{\Gamma}^3/\rho. \tag{72}$$

Also, from (69), for $r \geqslant 2$ we obtain

$$g_{rr} = -\tilde{\Gamma} g_{r-1,r-1};$$

$$(73)\quad g_{rm} = \tilde{\Gamma}[(m+1)g_{r,m+1} - 2mg_{r-1,m} - g_{r-1,m-1} + (m-1)g_{r-2,m-1}], \quad m = 1, \cdots, r-1;$$

$$g_{r0} = \tilde{\Gamma} g_{r1}.$$

The above recursions together with the end values in (72) form a complete system.

Now, from (14), (23), (64) and (66), it follows that

$$(74)\qquad J_r(u) = ug_r'(u) + g_r(u) - g_{r+1}'(u).$$

Hence, from (71), we obtain

$$(75)\qquad J_r(u) = \sum_{k=0}^{r} (k+1)(g_{rk} - g_{r+1,k+1})u^k,$$

and (52) implies that

$$(76)\qquad (-1)^k H_{r-k}^{(k+1)}(1) = (k+1)!(g_{rk} - g_{r+1,k+1}).$$

Thus we have a recursive procedure for calculating these quantities, which are required in the recursions (59). We also note, from (48), (64), (66) and (71) that

$$(77)\qquad H_r(1) = g_{r0}, \; r = 0,1,2, \cdots .$$

We have verified that (76) and (77) are consistent with (47) and (49).

REFERENCES

[1] D. MITRA, *Waiting time distribution from closed queueing network models of shared-processor systems*, Proc. Eighth Intl. Symp. on Computer Performance, Modeling, Measurement and Evaluation, Performance 81, F.J. Kylstra, ed., North-Holland, Amsterdam, 1981, pp. 113-131.

[2] F. P. KELLY, *Reversibility and Stochastic Networks*, John Wiley, New York, 1979.

[3] F. BASKETT, K. M. CHANDY, R. R. MUNTZ, AND F. G. PALACIOS, *Open, closed and mixed networks of queues with different classes of customers*, J. ACM, 22 (1975), pp. 248-260.

[4] L. KLEINROCK, *Queueing Systems, Vol. II: Computer Applications*, John Wiley, New York, 1976.

[5] J. MCKENNA AND D. MITRA, *Integral representations and asymptotic expansions for closed markovian queueing networks: normal usage*, Bell System Tech. J., 61 (1982), pp. 661-683.

[6] D. P. GAVER, P. A. JACOBS AND G. LATOUCHE, *The normal approximation and queue control for response times in a processor-shared computer system model*, Proc. Intl. Sem. on Modelling and Performance Evaluation Methodology, INRIA, Paris, January 24-26, 1983, pp. 25-41.

[7] F. W. J. OLVER, *Asymptotics and Special Functions*, Academic Press, New York, 1974, p. 148.

[8] E. COFFMAN, R. R. MUNTZ AND H. TROTTER, *Waiting time distributions for processor-sharing systems*, J. ACM, 17 (1970), pp. 123-130.

[9] M. SAKOTA, S. NOGUCHI AND J. OIZUMI, *Analysis of processor-shared queueing model for time sharing systems*, Proc. Second Hawaii Intl. Conf. System Sciences, B.S.M. Granborg, ed., Western Periodicals, 1969, pp. 625-627.

[10] R. MUNTZ, *Waiting time distributions for round robin queueing systems*, Proc. Symp. Computer-Communications Networks and Teletraffic, J. Fox, ed., Poly. Inst. of Brooklyn, New York, 1972.

[11] J. W. COHEN, *The multiple phase service network with generalized processor sharing*, Acta Informatica, 12 (1979), pp. 245-284.

[12] G. FAYOLLE, I. MITRANI AND R. IASNOGORODSKI, *Sharing a processor among many job classes*, J. ACM, 27 (1980), pp. 519-532.