SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics

INTEGRATION IN FINITE TERMS WITH SPECIAL FUNCTIONS: THE LOGARITHMIC INTEGRAL*

G. W. CHERRY†

Abstract. Since R. Risch published an algorithm for calculating symbolic integrals of elementary functions in 1969 (Trans. Amer. Math. Soc., 139 (1969), pp. 167-189), there has been an interest in extending his methods to include nonelementary functions. In this paper, we use the framework of differential algebra to make precise the notion of integration in terms of elementary functions and logarithmic integrals. Basing our work on a recent extension of Liouville's theorem on integration in finite terms, we then describe a decision procedure for determining if a given element in a transcendental elementary field has an integral which can be written in terms of elementary functions and logarithmic integrals. This algorithm first examines the structure of the integrand in order to limit the logarithmic integrals which could appear in the integral to a finite number. This allows us to write a general expression for the integral and then use techniques similar to those employed by Risch to calculate the undetermined parts.

Key words. integration in finite terms, special functions, logarithmic integral

1. Introduction. In 1969 R. Risch gave a decision procedure for determining whether an expression involving an algebraically independent set of logarithmic and exponential expressions (Risch's "monomials"), possesses an elementary integral [Risch69]. Although this result was seen as the culmination of a long history of research on symbolic integration, there are still a number of open questions in this area. The work presented here is an extension of the Risch algorithm to include a well-known yet nonelementary function, the logarithmic integral, into the integral.

Let F be a differential field of characteristic zero with subfield of constants C and let E be a differential extension of F. An element θ in E such that $\theta' = a'/a$ for some a in F is called a *logarithm of* a and we write $\theta = \log(a)$. Similarly, an element θ such that $\theta' = \theta a'$ for some a in F is called an *exponential of* a and we write $\theta = \exp(a)$. We say that a differential field, F, is *elementary* if $F = C(x, \theta_1, \cdots, \theta_n)$ where C is a field of constants, x is a solution to $z' - 1 = 0$ and each θ_i is an algebraic, exponential or logarithm over $C(x, \theta_1, \cdots, \theta_{i-1})$. F is further called *transcendental* if, for each i, θ_i is not algebraic over $C(x, \theta_1, \cdots, \theta_{i-1})$.

The Risch algorithm determines if an element of a transcendental elementary field, F, has an integral in an elementary extension of F. If the integral exists then it will be calculated and if not, then the algorithm terminates. An important class of functions which will cause such termination are the logarithmic integrals: $\operatorname{li}(u) = \int (u'/\log(u))\, dx$. The algorithm presented here considers the same set of integrands as those considered by Risch but attempts to write the integrals of these in terms of elementary functions and logarithmic integrals.

Example 1.1. $\int (x/\log(x)^2)\, dx$ is not elementary yet

$$\int \frac{x}{\log(x)^2}\, dx = 2\operatorname{li}(x^2) - \frac{x^2}{\log(x)}. \qquad \square$$

Section 2 will formalize these notions and present an extension to Liouville's theorem on integration in finite terms. The classical Liouville theorem describes the

* Received by the editors November 22, 1983, and in revised form August 16, 1984. This work was supported, in part, by the System Development Foundation under grant 301.

† Department of Computer and Information Science, University of Delaware, Newark, Delaware 19711. Present address, Computer Research Laboratories, Tektronix, Inc., Beaverton, Oregon 97077.

structure an expression must exhibit in order to have an elementary integral and was the basis for Risch's work. The extension presented here, which includes the logarithmic integral, will be used in a similar manner in § 5 to prove the main result of the paper. In § 3 various types of transcendental elementary field descriptions are discussed. Section 4 is concerned with Σ-decompositions. These decompositions will be useful in § 5.

2. Li-elementary extensions. We begin by making precise the notions of li-elementary extensions and li-elementary fields. Let F be a differential field of characteristic zero with derivation $'$ and constants C. We say that a differential extension E of F is a *li-elementary extension of* F if $F = F_0 \subseteq F_1 \subseteq \cdots \subseteq F_n = E$ such that $F_i = F_{i-1}(\theta_i)$ where for each i, $1 \leqq i \leqq n$, one of the following holds:

(i) θ_i is algebraic over F_{i-1};

(ii) $\theta_i' = u'\theta_i$ for some u in F_{i-1} (i.e. $\theta_i = \exp(u)$);

(iii) $\theta_i' = u'/u$ for some nonzero u in F_{i-1} (i.e. $\theta_i = \log(u)$);

(iv) $\theta_i' = u'/v$ for some nonzero u and v in F_{i-1} such that $v' = u'/u$. In this case we write $\theta_i = \mathrm{li}(u)$.

Cases (i), (ii) and (iii) describe the elementary functions. Case (iv) extends this notion and allows us to adjoin nonelementary elements of the form $\int (u'/\log(u))\,dx$ onto these fields. By *li-elementary field* we mean any field which can be described as an li-elementary extension of $C(x)$ where C is the field of constants and x is a solution to $z' - 1 = 0$.

Example 2.1. Let R be the set of real numbers and let $F = R(x)$ be the set of rational functions with coefficients in R. Then F is a differential field under the usual derivative: d/dx. The real-valued function e^x is an exponential over F and so $R(x, e^x)$ is an elementary extension of F. Next consider $g(x) = \int_0^x (e^t/t)\,dt$. This function is a logarithmic integral over $F(x, e^x)$, $(g(x) = \mathrm{li}(e^x))$, and hence $R(x, e^x, g(x))$ is a li-elementary field. □

The decision procedure given here is based on an extension of Liouville's theorem on integration in finite terms published recently by Singer, Saunders and Caviness [Scs81]. We shall state and prove a refinement of this theorem for logarithmic integrals. In order to do this we first need the following simple lemma.

LEMMA 2.1. *Let K be a field containing the nth roots of unity and let E be an algebraic extension of K. If v is an element of E such that v^n is in K, then either v is in K or the trace of v in E with respect to K is zero.*

Proof. First note that v satisfies a pure equation and therefore has a cyclic Galois group: $\{\sigma, \sigma^2, \cdots, \sigma^r\}$ [Vdw50]. Next, let $\sigma(v) = \xi^k v$ where ξ is a primitive nth root of unity and write the conjugates of v as

$$\begin{aligned} \sigma(v) &= \xi^k v, \\ \sigma^2(v) &= \xi^{2k} v, \\ &\vdots \\ \sigma^r(v) &= \xi^{rk} v. \end{aligned}$$

Now since $\sigma = \sigma^{r+1}$, ξ^k is an rth root of unity and so either $r = 1$ or

$$\mathrm{Tr}(v) = (\xi^k + \cdots + \xi^{rk})v = 0 \cdot v = 0. \qquad \square$$

THEOREM 2.2. *Let F be a liouvillian extension of its field of constants C. Assume C is algebraically closed and has characteristic zero and let γ be an element of F which has an integral in some li-elementary extension of F. Then there exist constants c_i and d_i*

in C and elements w_i, u_i and v_i in F such that

$$\gamma = w_0' + \sum c_i \frac{w_i'}{w_i} + \sum d_i \frac{u_i'}{v_i} \tag{2.1}$$

where $v_i' = u_i'/u_i$.

Proof. A direct application of the main theorem from [Ssc81] implies that there exist elements w_i, u_i and v_i algebraic over F satisfying (2.1). We only need to show that these elements are in F. Since $v_i' = u_i'/u_i$ we have by the lemma in [Rosi77, p. 338] that v_i, and hence $1/v_i$, is in F and that $u_i^{n_i}$ is in F for positive integers n_i. Now let E be a normal extension of F containing the w_i and u_i and take the trace in E on both sides of (2.1) over the field F. This yields

$$m\gamma = (\mathrm{Tr}(w_0))' + \sum c_i \frac{(N(w_i))'}{N(w_i)} + \sum d_i \frac{(\mathrm{Tr}(u_i))'}{v_i}$$

where m is a positive integer and $N(w_i)$ is the norm of w_i. The proof is completed by noting that, by Lemma 2.1, $\mathrm{Tr}(u_i) = 0$ for all u_i not in F. □

There is a technicality implicit in the statement of this theorem, (and, in fact, in the remainder of this paper), which involves transcendental constants. Consider the following example.

Example 2.2. Let $\bar{Q}$ denote the algebraic closure of the rationals and let $F = \bar{Q}(x, \ln(x))$ where $\ln(x)$ is the classical logarithm (i.e. the unique solution to $y' = 1/x$, $y(1) = 0$). Then $\gamma = 1/(\ln(x)+1)$ has an antiderivative in some li-elementary extension of F since (2.1) is satisfied with $u_1 = x$, $v_1 = \ln(x)+1$, $d_1 = 1$ and $w_i = c_i = 0$. Notice that by introducing transcendental constants we can write $\int \gamma$ as $1/e \int ((ex)'/\ln(ex))\, dx = 1/e\ \mathrm{li}(ex)$ which has the added property that $v_1 = \ln(u_1)$. □

As this example indicates, equation (2.1) is not a unique representation for γ. In fact we can make the following observation: Let F be as in Theorem 2.2 and let γ be an element of F such that (2.1) holds for some w_i, u_i, v_i in F and constants c_i and d_i. Consider an element $\bar{u}_i$ of a differential extension E of F where $v_i' = \bar{u}_i'/\bar{u}_i$. It is easy to show that $\bar{u}_i = \lambda_i u_i$ for some nonzero λ_i in the constant field of E. Furthermore we have

$$\gamma = w_0' + \sum c_i \frac{w_i'}{w_i} + \sum \frac{d_i}{\lambda_i} \frac{\bar{u}_i'}{v_i}.$$

Therefore, an integration algorithm for integrating with li-elementary functions over some classical field F, can first attempt to satisfy (2.1) with some w_i, u_i, v_i, c_i and d_i in F and then, with the possible inclusion of transcendental constants, adjust u_i and v_i in order to satisfy the condition $v_i = \ln(u_i)$. For other examples of this see Examples 5.3 and 5.4.

3. Towers of elementary fields. An important phase of any symbolic integration algorithm is the formulation of a suitable field for describing the integrand. In this section various notions are defined which shall be used later in the construction of these fields.

Let $F = C(x, \theta_1, \cdots, \theta_n)$ be a transcendental elementary extension of $C(x)$ where C is the field of constants and x is a solution to $x' = 1$. Rearrange the θ's into a tower $C(x) = F_0 \subseteq F_1 \subseteq \cdots \subseteq F_r = F$ where $F_i = F_{i-1}(\theta_{i1}, \cdots, \theta_{im_i})$ for $i = 1, \cdots, r$ and where one of the following holds for each θ_{ij}:

(i) $\theta_{ij}' = a_{ij}'/a_{ij}$ for some nonzero a_{ij} in F_{i-1} where a_{ij} is not in F_{i-2};

(ii) $\theta_{ij}' = \theta_{ij} a_{ij}'$ for some a_{ij} in F_{i-1} where a_{ij} is not in F_{i-2}.

We now define the rank of a tower of transcendental elementary fields $F = C(x, \theta_1, \cdots, \theta_n)$, denoted rank (F), to be the tuple $(m_r, \cdots, m_1, 1)$. Notice that rank (F) depends on the particular monomials chosen in the definition of F.

Example 3.1. The fields $E_1 = C(x, \exp(x), \exp(\exp(x)+x), \exp(\exp(x)+x^2))$ and $E_2 = C(x, \exp(x), \exp(x^2), \exp(\exp(x)))$ are isomorphic fields yet rank $(E_1) = (2, 1, 1)$ and rank $(E_2) = (1, 2, 1)$. □

We can also define the rank of a particular element in F. Let $F_0, \cdots, F_r$ be as above. An element a in F has rank k if a is an element of F_k and a is not an element of F_{k-1}.

Given two sequences $(m_r, \cdots, m_1, 1)$ and $(\tilde{m}_s, \cdots, \tilde{m}_1, 1)$ we say that $(m_r, \cdots, m_1, 1) < (\tilde{m}_s, \cdots, \tilde{m}_1, 1)$ if $r < s$ or if $r = s$ and $(m_r, \cdots, m_1, 1) < (\tilde{m}_s, \cdots, \tilde{m}_1, 1)$ in the usual lexicographic ordering. Notice that, with this ordering, the set of all tuples of nonnegative integers is a well ordered set with (1) being the first element. We can therefore prove theorems using induction on rank (F). A similar notion of rank and a further discussion can be found in [Ssc81].

Let $F = C(x, \theta_1, \cdots, \theta_n)$ be an elementary transcendental extension of $C(x)$. We shall call F *factored* if for each logarithmic monomial $\theta_i = \log(a_i)$, a_i is an irreducible polynomial in $C[x, \theta_1, \cdots, \theta_{i-1}]$. An easy induction shows that given any transcendental elementary extension $F = C(x, \theta_1, \cdots, \theta_n)$ of $C(x)$, one can construct a factored field $\tilde{F} = C(x, \tilde{\theta}_1, \cdots, \tilde{\theta}_m)$ such that F is differentially isomorphic to a subfield of $\tilde{F}$. We may assume therefore that the fields that define our integrands are factored.

Let F be as above and let $\theta_i = \exp(a_i)$ be an exponential monomial of rank k. Suppose that $a_i = \sum (p_j/q_j)\theta_j + \gamma$, where p_j and q_j are integers, the θ_j are logarithmic monomials of rank $k-1$ and rank $(\gamma) < k-1$. We shall call such monomials *normalized* if $0 < p_j/q_j < 1$ and say that F is normalized if each exponential monomial with the above format is normalized. □

Example 3.2. Let $F = C(x, \log(x), \exp(\frac{5}{2}\log(x)+x))$. Although this is not a normalized field, it is differentially isomorphic to the field $\tilde{F} = C(x, \log(x), \exp(\frac{1}{2}\log(x)+x))$ which is normalized. □

As the above example indicates, one can always replace the monomials $\theta_1, \cdots, \theta_n$ with another set of monomials $\tilde{\theta}_1, \cdots, \tilde{\theta}_n$ so that the field $\tilde{F} = C(x, \tilde{\theta}_1, \cdots, \tilde{\theta}_n)$ is normalized and differentially isomorphic to F. Also note that in $\tilde{F}$ the isomorphic image of θ_i will have a representation of the form $\tilde{A}_i\tilde{\theta}_i$, where $\tilde{A}_i$ is in $C(x, \tilde{\theta}_1, \cdots, \tilde{\theta}_{i-1})$. Moreover, it is not hard to show that rank $(\tilde{F}) \leqq$ rank (F), (where $\leqq$ is the reflexive closure of $<$).

We shall make two assumptions concerning the field of constants, C. We first assume that polynomials with coefficients in C can be factored into irreducible factors in a finite number of steps. Note that this implies, ([Vdw50]), that polynomials in $C(x, \theta_1, \cdots, \theta_{k-1})[\theta_k]$ can also be factored in a finite number of steps over $C(x, \theta_1, \cdots, \theta_{k-1})$. Secondly, we will assume that c is algebraically closed. These two assumptions hold for the algebraic closure of the rationals and the algebraic closure of any finitely generated extension of this field [Davtr81].

4. Σ-decompositions. Let K be a field of characteristic zero and let $\Sigma = (f_1, \cdots, f_m)$ be a sequence of distinct and irreducible elements of $K[x]$, where no f_i is in K. Given Φ in $K(x)$ we say that Φ has a *Σ-decomposition over* K if there exist b_i in K, integers α_{ij}, and a natural number n so that

$$\Phi = \sum_{i=1}^{n} b_i \prod_{j=1}^{m} f_j^{\alpha_{ij}}.$$

We are interested in this section in the existence, uniqueness, and construction of Σ-decompositions.[1] The following examples show that neither existence nor uniqueness are guaranteed unless we restrict the decompositions in some manner.

Example 4.1. It is easy to show with simple degree arguments that x has no Σ-decomposition over the rationals if $\Sigma=(x^2+1)$. □

Example 4.2. Again let K be the rationals and let $\Sigma=(x, x+1)$. Then 0 has an infinite number of Σ-decompositions since

$$0=(x+1)^n-\sum_{k=1}^{n} C(n,k)x^{n-k} \quad \text{for } n=1,2,\cdots. \qquad \square$$

Let T be a subset of Z and let $g: T\to Z^m$. We say that Φ in $K(x)$ has a *Σ-decomposition restricted by g* if

$$\Phi=\sum_{i=1}^{n} b_i \prod_{j=1}^{m} f_j^{\alpha_{ij}}$$

where for all i, α_{i1} is in T, $g(\alpha_{i1})=(\alpha_{i1}, \alpha_{i2}, \cdots, \alpha_{im})$ and $b_i\neq 0$ except in the trivial case where $\Phi=0$ and $n=1$. Notice that if Φ has a Σ-decomposition restricted by g, then we may assume that $\alpha_{11}<\alpha_{21}<\cdots<\alpha_{n1}$.

Let f be an irreducible element of $K[x]$ and let Φ be a nonzero element of $K(x)$. By the unique factorization in $K[x]$ we may write $\Phi=f^{\alpha}p/q$ where f, p, q are pairwise relatively prime elements of $K[x]$, q is monic and α is an integer. Moreover, this representation is unique. We then refer to α as the *multiplicity* of f in Φ.

LEMMA 4.1. *Let Σ and g be as above. Let Φ be a nonzero element of $K(x)$ and let $\sum_{i=1}^{n} b_i \prod f_j^{\alpha_{ij}}$ be a Σ-decomposition of Φ restricted by g where $\alpha_{11}<\alpha_{21}<\cdots<\alpha_{n1}$. Then α_{11} is the multiplicity of f_1 in Φ.*

Proof. The proof is trivial if $n=1$. Therefore assume that $n>1$. Letting $\Phi=f_1^{\alpha}p/q$ where f_1, p and q are pairwise relatively prime elements of $K[x]$ and q is monic, we have

$$f_1^{\alpha}\frac{p}{q}=b_1 f_1^{\alpha_{11}}\prod_{j=2}^{m} f_j^{\alpha_{1j}}+\sum_{i=2}^{n} b_i f_1^{\alpha_{i1}}\prod_{j=2}^{m} f_j^{\alpha_{ij}},$$

and so

$$f_1^{\alpha}\frac{p}{q}=f_1^{\alpha_{11}}\left(b_1\prod_{j=2}^{m} f_j^{\alpha_{1j}}+\sum_{i=2}^{n} b_i f_1^{\alpha_{i1}-\alpha_{11}}\prod_{j=2}^{m} f_j^{\alpha_{ij}}\right). \tag{4.1}$$

Since f_j is relatively prime to f_1 for $j\geqq 2$, we have $b_1\prod_{j=2}^{m} f_j^{\alpha_{1j}}=p_1/q_1$ for some p_1 and q_1 in $K[x]$ where f_1, p_1, and q_1 are pairwise relatively prime. Also since $\alpha_{i1}-\alpha_{11}>0$ for all $i\geqq 2$, we can write

$$\sum_{i=2}^{n} b_i f_1^{\alpha_{i1}-\alpha_{11}}\prod_{j=2}^{m} f_j^{\alpha_{ij}}=f_1^{\bar{\alpha}}\frac{p_2}{q_2}$$

where f_1, p_2 and q_2 are pairwise relatively prime and $\bar{\alpha}>0$. Substituting these into (4.1) yields

$$f_1^{\alpha}\frac{p}{q}=f_1^{\alpha_{11}}\left(\frac{p_1}{q_1}+f_1^{\alpha}\frac{p_2}{q_2}\right)=f_1^{\alpha_{11}}\left(\frac{p_1q_2+f_1^{\bar{\alpha}}p_2q_1}{q_1q_2}\right)=f_1^{\alpha_{11}}\frac{p_3}{q_3},$$

where f_1, p_3 and q_3 are pairwise relatively prime elements of $K[x]$ and q_3 is monic. Thus $\alpha=\alpha_{11}$. □

[1] We shall assume throughout this section that K is computable. That is, the field operations can be effectively carried out in K.

As a consequence of this lemma, we have the following uniqueness theorem for restricted Σ-decompositions.

THEOREM 4.2. *Let Σ and g be as above and suppose Φ has a Σ-decomposition restricted by g. Then this decomposition is unique except for the possible reordering of terms and combining of like terms.*

Proof. Let $\Phi = \sum_{i=1}^{n} b_i \prod f_j^{\alpha_{ij}}$ and $\Phi = \sum_{i=1}^{\bar{n}} \bar{b}_i \prod f_j^{\bar{\alpha}_{ij}}$ be two Σ-decompositions of Φ restricted by g. Thus

$$\sum_{i=1}^{n} b_i \prod f_j^{\alpha_{ij}} - \sum_{i=1}^{\bar{n}} \bar{b}_i \prod f_j^{\bar{\alpha}_{ij}} = 0.$$

Now reorder and combine the terms in this equation to obtain a restricted Σ-decomposition for 0:

$$\sum_{i=1}^{r} c_i \prod f_j^{\delta_{ij}} = 0$$

where $\delta_{11} < \delta_{21} < \cdots < \delta_{r1}$. We wish to show that $r = 1$ and $c_1 = 0$. Assuming that $r > 1$ yields

$$-c_1 \prod f_j^{\delta_{1j}} = \sum_{i=2}^{r} c_i \prod f_j^{\delta_{ij}}.$$

The multiplicity of f_1 in the left side of this equation is δ_{11} and, by Lemma 4.1, the multiplicity of f_1 in the right side is δ_{21}. But $\delta_{11} < \delta_{21}$, thus proving that $r = 1$. From this it follows that $c_1 = 0$. □

The lemma also yields an algorithm for calculating Σ-decompositions under the assumption that the decomposition exists.

THEOREM 4.3. *Let Σ and g be as above. Let Φ be an element of $K(x)$ and suppose that Φ has a Σ-decomposition restricted by g. Further assume that for any α in T we can calculate $g(\alpha)$ in a finite number of steps. Then we can calculate the Σ-decomposition of Φ in a finite number of steps.*

Proof. Let $\Phi = \sum_{i=1}^{n} b_i \prod f_j^{\alpha_{ij}}$ be the Σ-decomposition of Φ restricted by g. Using an inductive argument on n, we need only show that $b_1, \alpha_{11}, \cdots, \alpha_{1m}$ can be calculated in a finite number of steps. (Once these values are known we can apply the induction hypothesis to $\Phi - b_1 \prod f_j^{\alpha_{1j}}$). But by Lemma 4.1, α_{11} is the multiplicity of f_1 in Φ and, since g is computable, this allows us to compute $\alpha_{12}, \cdots, \alpha_{1m}$. We can then calculate relatively prime polynomials p and q such that q is monic, $\gcd(f_1, q) = 1$ and

$$\frac{p}{q} = \frac{\Phi}{\prod f_j^{\alpha_{1j}}} = b_1 + \sum_{i=2}^{n} b_i \prod f_j^{\alpha_{ij} - \alpha_{1j}}$$

which implies that

$$p = b_1 q + q \sum_{i=2}^{n} b_i \prod f_j^{\alpha_{ij} - \alpha_{1j}}. \tag{4.2}$$

Now since p and $b_1 q$ are in $K[x]$, we have $q \sum_{i=2}^{n} b_i \prod f_j^{\alpha_{ij} - \alpha_{1j}}$ in $K[x]$. Then since $\alpha_{i1} - \alpha_{11} > 0$ for each $i > 1$, we can write (4.2) as

$$p = b_1 q + f_1 Q \tag{4.3}$$

with Q in $K[x]$. Finally, (4.3) implies that $b_1 = (p \bmod f_1)/(q \bmod f_1)$. □

The only question remaining concerning restricted Σ-decompositions is the following: Given an arbitrary element Φ in $K(x)$ and a function g, can we determine in a

finite number of steps if Φ has a Σ-decomposition restricted by g? A partial solution is to apply the above algorithm to Φ. The problem, of course, is guaranteeing termination in the case where Φ has no Σ-decomposition restricted by g.

Example 4.3. Let $\Phi = 1/(1-x)$, $\Sigma = (x)$, $T = Z$ and $g(\alpha) = (\alpha)$. Then the above calculations will generate $b_1 = 1$, $b_2 = 1, \cdots$, and will not terminate. □

The theorem below gives conditions under which a termination strategy can be adopted. Let $\Phi = \sum_{i=1}^{n} b_i \prod f_j^{\alpha_{ij}}$ be a Σ-decomposition of Φ and, for each j, let $p_j(\alpha)$ be the unique polynomial of degree less than n for which $p_j(\alpha_{i1}) = \alpha_{ij}$, $i = 1.2, \cdots, n$. We then define the *degree* of this decomposition to be the maximum of the degrees of the p_j. Although the Σ-decompositions which occur in § 5 are only of degree one the added generality here and in the next theorem comes at little cost and, in fact, appears to be useful when designing algorithms for integrating in terms of other special functions (e.g., the error function [Cher83]).

THEOREM 4.4. *Let Σ and g be as above and let d be a positive integer. Suppose that for any integer, α, we can determine in a finite number of steps if α is in T and, if so, can calculate $g(\alpha)$. Then given Φ in $K(x)$, we can determine in a finite number of steps if Φ has a Σ-decomposition restricted by g with degree less than d.*

Proof. We first calculate α_{11} as the multiplicity of f_1 in Φ. If α_{11} is not in T we stop and say that Φ has no such Σ-decomposition. Otherwise we calculate $\alpha_{12}, \cdots, \alpha_{1m}$, form $\Phi / \prod f_j^{\alpha_{1j}} = p/q$, and let $b_1 = p \bmod f_1 / q \bmod f_1$. If b_1 is not in K we again stop. If b_1 is in K we let $\Phi_2 = \Phi - b_1 \prod f_j^{\alpha_{1j}}$ and calculate α_{21} as the multiplicity of f_1 in Φ_2. If α_{21} is in T and $\alpha_{21} > \alpha_{11}$ then continue with the calculation of $(b_2, \alpha_{21}, \cdots, \alpha_{2m})$. In this manner, we compute $(b_3, \alpha_{31}, \cdots, \alpha_{3m}), \cdots, (b_d, \alpha_{d1}, \cdots, \alpha_{dm})$ and, with this data, form $p_1(\alpha), \cdots, p_m(\alpha)$. Once these polynomials are known, we can find an integer α^* such that $p_j(\alpha)$ is monotone on the interval (α^*, ∞) for each j. Note that if p_j has degree 0 for some j then we may replace Φ_{d+1} with $\Phi_{d+1} / f_j^{\alpha_{dj}}$ thereby insuring that the p_j are strictly monotone on this interval. Now continue with the iterations as before until $\alpha_{k1} > \alpha^*$ for some $k \geqq d$. If at any point $\alpha_{ij} \neq p_j(\alpha_{i1})$, then there is no such decomposition. Otherwise we have an element Φ_{k+1} which we wish to write as $\sum_{k+1}^{n} b_i \prod f_j^{\alpha_{ij}}$ where the sequences $(\alpha_{k+1,j}, \cdots, \alpha_{nj})$ are strictly monotone for each j. There are two cases:

(i) Suppose the jth sequence is monotone decreasing. Let r be the multiplicity of f_j in Φ_{k+1}. By examining the partial fraction decomposition of Φ_{k+1} we see that $\alpha_{nj} = r$. Therefore we may terminate our computations if for some $i > k$ we have $\alpha_{ij} < r$.

(ii) Assume that all the sequences are monotone increasing. In this case we iterate until $\alpha_{r1}, \cdots, \alpha_{rm}$ are all positive for some r. If Φ_r is not in $K[x]$ we terminate with failure. Otherwise we have $\deg(\prod f_j^{\alpha_{nj}}) = \deg(\Phi_r)$ and can terminate the computations if for some $i > r$ we have $\deg(\prod f_j^{\alpha_{ij}}) > \deg(\Phi_r)$. □

These results generalize easily to the multivariate case. That is, let $\Sigma = (f_1, \cdots, f_m)$ be a sequence of pairwise relatively prime, irreducible elements of $K[x_1, \cdots, x_r]$ where f_i is not in K for each i. Σ-decompositions, restricted Σ-decompositions, and the degree of a decomposition are all defined as before.

In order to determine if Φ in $K(x_1, \cdots, x_r)$ has a Σ-decomposition restricted by g of degree d over K we first single out x_k such that f_1 is not in $K[x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_r]$. Then form $\Sigma' = (f_1, f_{k1}, \cdots, f_{ks})$ where f_{kj} is not in $K[x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_r]$ for each j. Now use Theorem 4.4 to calculate (if they exist) elements B_i in $K(x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_r)$ and integers α_{ij} so that

$$\Phi = \sum B_i f_1^{\alpha_{i1}} \prod f_{kj}^{\alpha_{ij}}. \tag{4.4}$$

If Φ has a Σ-decomposition restricted by g of degree d over K then (4.4) will be a

Σ'-decomposition restricted by a projection of g also of degree d over $K(x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_r)$. At this point we need only factor each B_i over $K(x_1, \cdots, x_{k-1}, x_{k+1}, \cdots, x_r)$ and examine these factors and multiplicities to determine if Φ has a decomposition over K. Also this decomposition is unique since (4.4) is unique and each factorization of B_i is unique.

5. The logarithmic integral. In this section a decision procedure for calculating antiderivatives of transcendental elementary functions in terms of elementary functions and logarithmic integrals is described. A useful operation when integrating with special functions is to partition the integrand into a number of simpler expressions. The following lemma describes how to do this with logarithmic integrals.

LEMMA 5.1. *Let F be as in Theorem 2.2 and let θ be an exponential monomial over F. Let γ be an element of $F(\theta)$ and, using a partial fraction decomposition, write $\gamma = A_m\theta^m + \cdots + A_{\bar{m}}\theta^{\bar{m}} + A_0 + P(\theta)/Q(\theta)$ where A_k is in F for all $\bar{m} \leqq k \leqq m$, p, q are in $F[\theta]$, $\deg(P(\theta)) < \deg(Q(\theta))$ and $\gcd(Q(\theta), \theta) = 1$. Then γ has an antiderivative in some li-elementary extension of $F(\theta)$ if and only if each of terms $A_m\theta^m, \cdots, A_{\bar{m}}\theta^{\bar{m}}, A_0 + P(\theta)/Q(\theta)$ has such an antiderivative. In this case, for every k, the terms u_i'/v_i involved in the integration of $A_k\theta^k$ are each of the form $f_i\theta^k$ where f_i is in F. The terms u_i'/v_i involved in the integration of $A_0 + P(\theta)/Q(\theta)$ are in F.*

Proof. If each of the terms $A_m\theta^m, \cdots, A_{\bar{m}}\theta^{\bar{m}}, A_0 + P(\theta)/Q(\theta)$ has an antiderivative in some li-elementary extension of $F(\theta)$ then the sum of these is an antiderivative of γ.

Conversely, suppose γ has an antiderivative in some li-elementary extension of $F(\theta)$. From Theorem 2.2 we have constants c_i, d_i and w_i, u_i, and v_i, in $F(\theta)$ such that (2.1) holds. We shall first show that for each i we have $u_i'/v_i = f_i\theta^{r_i}$ for some integer r_i and some f_i in F. Applying [Ros76, Thm. 2], we have that v_i is algebraic over F and that $u_i = \bar{u}_i\theta^{r_i}$ for some rational r_i and some $\bar{u}_i$ algebraic over F. But this in conjunction with Theorem 2.2 implies that r_i is an integer and that f_i is in F.

Equation (2.1) therefore becomes

$$\gamma = w_0' + \sum c_i \frac{w_i'}{w_i} + \sum d_i f_i \theta^{r_i}.$$

Now write both sides of this equation explicitly in terms of θ over the field F. This yields

$$\begin{aligned} &A_m\theta^m + \cdots + A_{\bar{m}}\theta^{\bar{m}} + A_0 + \frac{P(\theta)}{Q(\theta)} \\ &= \left(B_m\theta^m + \cdots + B_{\bar{m}}\theta^{\bar{m}} + B_0 + \frac{R(\theta)}{S(\theta)}\right)' + \sum c_i \frac{w_i'}{w_i} + \sum d_i f_i \theta^{r_i} \end{aligned} \tag{5.1}$$

where B_i is in F, $R(\theta)$ and $S(\theta)$ are in $F[\theta]$ and $\deg(R(\theta)) < \deg(S(\theta))$. Differentiating and comparing coefficients yields

$$\begin{aligned} A_m\theta^m &= (B_m\theta^m)' + \sum_m \bar{d}_i f_i \theta^m, \\ &\vdots \\ A_{\bar{m}}\theta^{\bar{m}} &= (B_{\bar{m}}\theta^{\bar{m}})' + \sum_{\bar{m}} \bar{d}_i f_i \theta^{\bar{m}}, \\ A_0 + \frac{P(\theta)}{Q(\theta)} &= \left(B_0 + \frac{R(\theta)}{S(\theta)}\right)' + \sum \bar{c}_i \frac{w_i'}{w_i} + \sum_0 \bar{d}_i f_i \theta^0, \end{aligned}$$

where the notation $\sum_k$ implies that the sum includes those terms in (5.1) for which $r_i = k$.

This completes the proof. □

We shall also need the following.

LEMMA 5.2. *Let K be a computable field and let p and q be relatively prime elements of $K[x]$. Given f in $K[x]$ with $\deg(f) = n > 0$, one can determine in a finite number of steps if there exists a polynomial d in $K[x]$ with degree less than n such that f divides $p + dq$. Moreover this polynomial, if it exists, it unique and can be calculated.*

Proof. First note that if such a d exists then f and q must be relatively prime, for a common factor of f and q would also be a common factor of p and q. Therefore polynomials x and y can be calculated such that $xf + yq = 1$. We claim now that $d = -py \bmod f$ is the unique solution to the problem. It is a solution since

$$p + dq \equiv p - pyq = p(1 - yq) = pxf \equiv 0 \quad (\text{modulo } f).$$

To show that it is unique let d_1 and d_2 be two solutions such that

$$fg_1 = p + d_1 q \quad \text{and} \quad fg_2 = p + d_2 q.$$

Subtracting yields

$$f(g_1 - g_2) = q(d_1 - d_2).$$

Since f and q are relatively prime, f must divide $d_1 - d_2$ which contradicts the degree conditions. □

Although Theorem 5.4 is the main result in this section, most of the work is done in the following theorem. Much of the proof of the main theorem is a reduction to this case.

THEOREM 5.3. *Let $E = C(x, \theta_1, \cdots, \theta_n)$ be a differential field of characteristic zero with algebraically closed subfield of constants C. Assume x is transcendental over C and a solution to $x' = 1$, θ_i is a monomial over $C(x, \theta_1, \cdots, \theta_{i-1})$ for each i, and $E = C(x, \theta_1, \cdots, \theta_n)$ is factored and normalized. Further assume that θ_n is exponential over $C(x, \theta_1, \cdots, \theta_{n-1})$, the rank of θ_n is r and the rank of E is $(1, m_{r-1}, \cdots, 1)$. Then given A in $C(x, \theta_1, \cdots, \theta_{n-1})$, one can determine in a finite number of steps if $A\theta_n$ has an antiderivative in some li-elementary extension of E.*

Proof. First some notation. Let $F = C(x, \theta_1, \cdots, \theta_{n-1})$ and let e and l be indexing sets for the exponential and logarithmic monomials in F:

$$e = \{i \mid \theta_i \text{ is an exponential monomial}\},$$

$$l = \{i \mid \theta_i \text{ is a logarithmic monomial}\}.$$

Also let $\theta_i' = \theta_i \alpha_i'$ for all i in e, $\theta_i' = a_i'/a_i$ for all i in l and let $\theta_n = \exp(a_n)$.

Now assume that $A\theta_n$ has an antiderivative in some li-elementary extension of E. From Theorem 2.2 and the proof of Lemma 5.1, we have

$$A\theta_n = (B\theta_n)' + \sum d_i \frac{u_i'}{v_i} \tag{5.2}$$

where B is in F, the d_i are constants, $v_i' = u_i'/u_i$ and u_i, v_i are in E. We first take a closer look at the structure of the u_i and v_i. Since u_i is an algebraic exponential over E, we have by [Roca79, Thm. 3.1] that there exists rational numbers r_i and r_{ij} and a constant v_i so that

$$v_i = r_i a_n + \sum_e r_{ij} a_j + \sum_l r_{ij} \theta_j + v_i.$$

Then since u_i is an exponential of v_i, we also have

$$u_i = \eta_i \prod_l a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}} \theta_n^{r_i} \tag{5.3}$$

where η_i is in C. Thus we may rewrite equation (5.2) as

$$A\theta_n = (B\theta_n)' + \sum d_i \frac{(r_i a_n + \sum_e r_{ij} a_j + \sum_l r_{ij}\theta_j + v_i)'}{(r_i a_n + \sum_e r_{ij} a_j + \sum_l r_{ij}\theta_j + v_i)} \eta_i \prod_l a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}} \theta_n^{r_i}.$$

We may assume by comparing coefficients that $r_i = 1$. Also since different values for η_i will change only the constant d_i, we may let $\eta_i = 1$ for every i. Finally all of the factors in (5.3) are pairwise relatively prime elements of $C[x, \theta_1, \cdots, \theta_n]$. Hence, u_i in E implies r_{ij} is an integer for all i and j.

The main idea in what follows is to reconstruct the elements v_i by examining the denominator of A. Once these values are known we apply the Main Theorem, part (b) from [Risch69]. This theorem reduces the calculation of the element B and the constants d_i to the solution of a system of linear equations with coefficients in C.

We have, from the conditions on the rank of E and the rank of θ_n, that rank $(a_n) = r-1$ and rank $(a_j) < r-1$ for all a_j, j in e. Thus there exists a monomial, say θ, of rank $r-1$ so that a_n involves θ and the other a_j are free of θ. Let $F = D(\theta)$ and write $a_n = p(\theta)/q(\theta)$ where $p(\theta)$ and $q(\theta)$ are relatively prime elements of $D[\theta]$ and $q(\theta)$ is monic. We then have

$$A\theta_n = (B\theta_n)' + \sum d_i \frac{(p(\theta)/q(\theta) + \sum_e r_{ij} a_j + \sum_l r_{ij}\theta_j + v_i)'}{(p(\theta)/q(\theta) + \sum_e r_{ij} a_j + \sum_l r_{ij}\theta_j + v_i)} \prod_l a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}} \theta_n. \tag{5.4}$$

The remainder of the proof is broken down into two cases.

Case (a). $\theta = \exp(a)$ or $\theta = x$. We have

$$v_i = \frac{p(\theta)}{q(\theta)} + \beta_i = \frac{p(\theta) + \beta_i q(\theta)}{q(\theta)}$$

where $\beta_i = \sum_e r_{ij} a_j + \sum_l r_{ij}\theta_j + v_i$ is in D. Since $p(\theta)$ and $q(\theta)$ are relatively prime the right side of this expression is the unique rational expression with monic denominator for v_i in the field $D(\theta)$. Thus for each i, the denominator of v_i is known. To determine the numerators first note that if $p(\theta) + \beta_0 q(\theta)$ is in D for some β_0 in D then such a β_0 is unique and is easily calculated. Therefore assume that $p(\theta) + \beta_i q(\theta)$ is not in D and let $\xi_i f_{i1}^{m_{i1}} \cdots f_{ik_i}^{m_{ik_i}}$ be its factorization where ξ_i is in D and each f_{ij} is monic and irreducible. Equation (5.4) then becomes

$$A\theta_n = (B\theta_n)' + \sum d_i \frac{(\xi_i f_{i1}^{m_{i1}} \cdots f_{ik_i}^{m_{ik_i}} q(\theta)^{-1})'}{(\xi_i f_{i1}^{m_{i1}} \cdots f_{ik_i}^{m_{ik_i}} q(\theta)^{-1})} \prod_l a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}} \theta_n$$

or

$$A\theta_n = (B\theta_n)' + \sum d_i \left(\frac{\xi_i'}{\xi_i} - \frac{q(\theta)'}{q(\theta)} + \sum_{j=1}^{k_i} m_{ij} \frac{f_{ij}'}{f_{ij}} \right) \prod_l a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}} \theta_n.$$

We claim now that each f_{ij} must divide the denominator of A unless $f_{ij} = \theta$. First note that if $f_{ij} \neq \theta$ then f_{ij} does not divide $f_{ij}'\theta^k$ for any $k \geqq 0$. Also by Lemma 5.2 each f_{ij} determines β_i uniquely and by [Roca79, Thm. 3.1] each β_i determines r_{ij} and v_i uniquely.[2] Hence each $f_{ij} \neq \theta$ must occur as a simple factor of the denominator of $\sum d_i u_i'/v_i$. Now consider the term $(B\theta_n)'$. It is easy to show by considering the partial fraction decomposition of the above expressions that f_{ij} is either a multiple factor or

[2] The uniqueness of r_{ij} and v_i follows from [Roca79, Thm. 3.1] as follows: If $\beta_i = \sum_e r_{ij} a_j + \sum_l r_{ij}\theta_j + v_i$ and $\beta_i = \sum_e \tilde{r}_{ij} a_j + \sum_l \tilde{r}_{ij}\theta_j + \tilde{v}_i$, then $0 = \sum_e (r_{ij} - \tilde{r}_{ij}) a_j + \sum_l (r_{ij} - \tilde{r}_{ij})\theta_j + (v_i - \tilde{v}_i)$. By [Roca79, Thm. 3.1], such a nontrivial representation for 0 contradicts the assumption that E is a transcendental elementary field.

is not a factor of the denominator of $(B\theta_n)'$. In either case f_{ij} must divide the denominator of A, proving our claim.

Now with each irreducible polynomial, p_i, in the denominator of A, we apply Lemma 5.2 and determine if a value β_i exists such that p_i divides $p(\theta)+\beta_i q(\theta)$. At this time we also try $p_i=\theta$. With these values for $\beta_1, \beta_2, \cdots$ and the value for β_0 calculated earlier we next determine if there exist integers r_{ij} and a constant v_i such that $\beta_i=\sum_e r_{ij}a_j+\sum_l r_{ij}\theta_j+v_i$.

In this way one can calculate, for a given A, all of the terms u_i'/v_i which could occur in (5.2). We may now apply [Risch69, Main Theorem, part (b)] in order to determine if there exist constants d_i and an element B satisfying (5.2).

Case (b). $\theta=\log(a)$. Although the strategy applied here will be the same as Case (a) it will not be as easy to implement. This is because the expression $\sum_e r_{ij}a_j+\sum_l r_{ij}\theta_j+v_i$ is no longer in D but instead is linear in θ.

Letting $l'=\{\theta_i|$ is a logarithmic monomial in $D\}$ we have

$$v_i=\frac{p(\theta)}{q(\theta)}+r_i\theta+\beta_i=\frac{p(\theta)+r_i\theta q(\theta)+\beta_i q(\theta)}{q(\theta)}$$

where $\beta_i=\sum_e r_{ij}a_j+\sum_{l'} r_{ii}\theta_j+v_i$. As in the previous case this determines the denominator of v_i for each i. There are two subcases.

Case (b1). $q(\theta)\neq 1$ or $\deg(p(\theta))>1$. First determine if there exist values r_0 and β_0 such that $p(\theta)+r_0\theta q(\theta)+\beta_0 q(\theta)$ is in D. This yields $r_0=-\mathrm{lc}(p(\theta))$, (where lc denotes the leading coefficient), and $\beta_0=-\mathrm{lc}(p(\theta)+r_0\theta q(\theta))$. We note that the only case where these values are not determined uniquely, where $q(\theta)=1$ and $p(\theta)+r_0\theta$ is in D, does not occur in this case. Therefore assume that $p(\theta)+r_i\theta q(\theta)+\beta_i q(\theta)$ is not in D and let $\xi_i f_{i1}^{m_{i1}}\cdots f_{ik_i}^{m_{ik_i}}$ be its factorization where ξ_i is in D and each f_{ij} is monic and irreducible. Again we rewrite (5.4) as

$$A\theta_n=(B\theta_n)'+\sum d_i\left(\frac{\xi_i'}{\xi_i}-\frac{q(\theta)'}{q(\theta)}+\sum_{j=1}^{k_i} m_{ij}\frac{f_{ij}'}{f_{ij}}\right)\prod_l a_j^{r_{ij}}\prod_e \theta_j^{r_{ij}}\theta_n.$$

Notice that in this case $\prod_l a_j^{r_{ij}}\prod_e \theta_j^{r_{ij}}$ is in D and $\deg_\theta(f_{ij}')<\deg_\theta(f_{ij})$ for each i. Consideration of the partial fraction decomposition of $(B\theta_n)'$ again implies that each f_{ij} must divide the denominator of A unless some cancellation occurs among the terms in $\sum d_i u_i'/v_i$. However such cancellation would imply an equation of the form

$$m_1\frac{f'}{f}\prod_l a_j^{r_{1j}}\prod_e \theta_j^{r_{1j}}+\cdots+m_t\frac{f'}{f}\prod_l a_j^{r_{tj}}\prod_e \theta_j^{r_{tj}}=0$$

or

$$m_1 a^{r_1}\prod_{l'} a_j^{r_{1j}}\prod_e \theta_j^{r_{1j}}+\cdots+m_t a^{r_t}\prod_{l'} a_j^{r_{tj}}\prod_e \theta_j^{r_{tj}}=0$$

where $m_i\neq 0$ is in C and F divides $p(\theta)+r_i\theta q(\theta)+\beta_i q(\theta)$ for all $1\leqq i\leqq t$. Since $\{a\}\cup\{a_j\}_{l'}\cup\{\theta_j\}_e$ is a set of relatively prime irreducible elements of $C[x,\theta_1,\cdots,\theta_{n-1}]$, this expression is a Σ-decomposition (where $\Sigma=(a,\cdots,a_i,\cdots,\theta_j,\cdots)$, $i\in l', j\in e$), for 0. Furthermore given r_i we can, by Lemma 5.2 and [Roca79, Thm. 3.1], determine r_{ij} uniquely for any i. By Theorem 4.2 such decompositions are unique and therefore $t=1$ and $m_1=0$, a contradiction. Thus we have shown that every factor of the numerator of v_i is also a factor of the denominator of A.

We can now reconstruct all values v_i which appear in (5.2) as follows: The values v_i with a linear numerator must have $r_i=0$ or $r_i=-\mathrm{lc}(p(\theta))$. With these values for r_i calculate, using Lemma 5.2, all values β_i such that $p(\theta)+r_i\theta q(\theta)+\beta_i q(\theta)$ is divisible

by a linear factor of the denominator of A. Next calculate all v_i with a numerator of degree greater than one. These can be determined directly from the denominator of A again using Lemma 5.2. Finally, with each value for β_i, we determine if there exist integers r_{ij} and constants v_i such that $\beta_i = \sum_e r_{ij}a_j + \sum_{l'} r_{ij}\theta_j + v_i$. Thus all terms u_i'/v_i which could appear in (5.2) have again been determined and we apply [Risch69, Main Theorem, part (b)].

Case (b2). $p(\theta) = r\theta + s$ and $q(\theta) = 1$. This case is handled differently from the previous cases since here one can not calculate all candidates for v_i by simply examining the denominator of A.

First break (5.2) into partial fractions with respect to θ over the field D to obtain

$$\left[A_{m+1}\theta^{m+1} + \cdots + A_0 + \sum_{i=1}^{K}\sum_{j=1}^{k_i+1}\frac{A_{ij}}{p_i^j}\right]\theta_n$$

$$= \left[\left(\sum_{i=0}^{m} B_i\theta^i + \sum_{i=1}^{K}\sum_{j=1}^{k_i}\frac{B_{ij}}{p_i^j}\right)' + (r\theta+s)'\left(\sum_{i=0}^{m} B_i\theta^i + \sum_{i=1}^{K}\sum_{j=1}^{k_i}\frac{B_{ij}}{p_i^j}\right)\right]\theta_n$$

$$+ \sum d_i\left[\frac{(r+r_i)'}{(r+r_i)} + \frac{(\theta+(s+\beta_i)/(r+r_i))'}{(\theta+(s+\beta_i)/(r+r_i))}\right]\prod_l a_j^{r_{ij}}\prod_e \theta_j^{r_{ij}}\theta_n$$

where p_i is monic and irreducible for every i. Notice that $r + r_i \neq 0$ since r_i is a nonzero integer and E is normalized. Differentiating and comparing terms yields $A_{1k_1+1} \equiv -k_1 B_{1k_1} p_1'$ (modulo p_1). We now proceed as in [Risch69, p. 180]: calculate polynomials R and S so that $Rp_1 + Sp_1' = A_{1k_1+1}$ and $\deg(S) < \deg(p_1)$. Then $B_{1k_1} = -S/k_1$. In this way we also determine $B_{2k_2}, \cdots, B_{Kk_K}$. Now replace $A\theta_n$ with $A\theta_n - [(B_{1k_1}/p_1^{k_1} + \cdots + B_{Kk_K}/p_K^{k_K})\theta_n]'$ and a repeat this process. Eventually B is reduced to a polynomial. That is

$$A_{m+1}\theta^{m+1} + \cdots + A_0 + \frac{\bar{A}_{11}}{p_1} + \cdots + \frac{\bar{A}_{K1}}{p_K}$$

$$= (B_m\theta^m + \cdots + B_0)' + (r\theta+s)'(B_m\theta^m + \cdots + B_0)$$

$$+ \sum d_i\left[\frac{(r+r_i)'}{(r+r_i)} + \frac{(\theta+(s+\beta_i)/(r+r_i))'}{(\theta+(s+\beta_i)/(r+r_i))}\right]\prod_l a_j^{r_{ij}}\prod_e \theta_j^{r_{ij}}$$

where $\bar{A}_{k1}$ is in $D(\theta)$ and $\deg_\theta(\bar{A}_{k1}) < \deg_\theta(p_k)$ for each $1 \leqq k \leqq K$.

This yields, for each k,

$$\frac{\bar{A}_{k1}}{p_k} = \sum_{pk} d_i \frac{(\theta+(s+\beta_i)/(r+r_i))'}{(\theta+(s+\beta_i)/(r+r_i))}\prod_l a_j^{r_{ij}}\prod_e \theta_j^{r_{ij}} \tag{5.5}$$

where the notation $\sum_{p_k}$ implies that the summation is taken only over terms where $p_k = \theta + (s+\beta_i)/(r+r_i)$. From (5.5) we have

$$\frac{\bar{A}_{k1}}{p_k'} = \sum_{p_k} d_i a^{r_i}\prod_{l'} a_j^{r_{ij}}\prod_e \theta_j^{r_{ij}}. \tag{5.6}$$

Constants r_{ij}, r_i and v_i can now be calculated for each k using the results in § 4. First note that equation (5.6) is a Σ-decomposition for $\bar{A}_{k1}/p_k'$, (where $\Sigma = (a, \cdots, a_i, \cdots, \theta_j, \cdots)$, $i \in l', j \in e$). Furthermore, we have

$$\beta_i = (r+r_i)\delta_k - s, \tag{5.7}$$

where $p_k = \theta + \delta_k$, and so given r_i one can determine β_i and hence r_{ij} and v_i for all j.

Therefore (5.6) is a Σ-decomposition restricted by a computable function. The proof shall be completed by showing that the Σ-decompositions in (5.6) are of degree one.

Clearly any Σ-decomposition with only one term has degree one. Therefore assume that (5.7) holds for distinct r_1 and r_2. Now define the set S as the collection of all elements in D which can be written as $\sum_e r_j a_j + \sum_{l'} r_j \theta_j + v$ for some constant v and rationals r_j. Notice that S is closed under addition, subtraction and multiplication by a rational number. From (5.7) we have

$$(r+r_1)\delta_k - s \tag{5.8}$$

and

$$(r+r_2)\delta_k - s$$

in S. Subtracting and multiplying by $1/(r_1-r_2)$ we have δ_k in S. Equation (5.8) now implies that $r\delta_k - s$ is in S. Thus there exist rational numbers w_{1j}, w_{0j} and constants w_1 and w_0 such that

$$\delta_k = \sum_e w_{1j} a_j + \sum_{l'} w_{1j}\theta_j + w_1$$

and

$$r\delta_k - s = \sum_e w_{0j} a_j + \sum_{l'} w_{0j}\theta_j + w_0.$$

Substituting these into (5.7) and collecting terms yields

$$\sum_e r_{ij} a_j + \sum_{l'} r_{ij}\theta_j + v_i = \sum_e (w_{1j} r_i + w_{0j}) a_j + \sum_{l'} (w_{1j} r_i + w_{0j})\theta_j + (w_1 r_i + w_0).$$

Since this representation for elements in S is unique we have for each j, $r_{ij} = w_{1j} r_i + w_{0j}$, proving that the degree of the Σ-decomposition in (5.6) is equal to one.

We can now determine if d_i and r_{ij} exist satisfying (5.6). If these values exist then we can again determine all u_i and v_i which occur in (5.2) and use [Risch69, Main Theorem, part (b)] to calculate B and the constants d_i. This concludes Case (b2) and concludes the proof. □

THEOREM 5.4. *Let $C(x)$ be a differential field of characteristic zero where x is transcendental over C, a solution to $x' = 1$, and C is an algebraically closed subfield of constants. Let $E = C(x, \theta_1, \cdots, \theta_n)$, $n \geqq 0$, be a transcendental elementary extension of $C(x)$ that is factored and normalized. Given γ in E, one can decide in a finite number of steps if γ has an antiderivative in some li-elementary extension of E, and if so, find constants c_i and d_i and elements w_i, u_i, v_i in E such that*

$$\gamma = w_0' + \sum c_i \frac{w_i'}{w_i} + \sum d_i \frac{u_i'}{v_i} \tag{5.9}$$

where $v_i' = u_i'/u_i$.

Proof. Assuming that γ has such an antiderivative, the existence of constants c_i and d_i and elements w_i, u_i, v_i satisfying the conditions of the theorem is guaranteed by Theorem 2.2. It remains to show that these elements can be constructed in a finite number of steps. Denote the rank of E by $(m_r, \cdots, m_1, 1)$. Our construction, which is an induction on the rank of E, is arranged so that if γ has no antiderivative in some li-elementary extension of E then this will be determined and the algorithm will be terminated.

(i) For the base case we take rank $(E) = (1)$ and so γ is an element of $C(x)$. It is well-known that γ has an antiderivative in an elementary extension of $C(x)$ (i.e. logarithmic integrals are not needed). Also there are well-known methods, such as

given in [Risch69], for constructing this antiderivative. This observation concludes the base case.

(ii) Assume the theorem is true for factored and normalized fields with rank less than $(m_r, \cdots, m_1, 1)$ and consider γ in $E = C(x, \theta_1, \cdots, \theta_n)$ where E is factored and normalized and the rank of E is $(m_r, \cdots, m_1, 1)$. The strategy adopted here is to reduce all cases to either Theorem 5.3 or the induction hypothesis. There are two basic cases.

Case (a). There is a logarithmic monomial of rank r. Call it θ with $\theta' = a'/a$ and let $E = F(\theta)$. Since v_i is a logarithm of u_i we have from [Roca79, Thm. 3.1] that

$$v_i = r_i\theta + \sum_e r_{ij}a_j + \sum_{l'} r_{ij}\theta_j + v_i$$

where $l' = \{j | \theta_j = \log(a_j)$ is a logarithmic monomial in $F\}$, $e = \{j | \theta_j = \exp(a_j)$ is an exponential monomial in $F\}$, r_{ij} are rational numbers and v_i is in C. We also have

$$u_i = \eta_i a^{r_i} \prod_{l'} a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}}$$

where η_i is in C. We may assume without loss of generality that $\eta_i = 1$ for all i.

Now break (5.9) into partial fractions with respect to θ over the field F. This yields

$$A_m\theta^m + \cdots + A_0 + \sum_{i=1}^{K} \sum_{j=1}^{k_i+1} \frac{A_{ij}}{p_i^j}$$

$$= \left[B_{m+1}\theta^{m+1} + \cdots + B_0 + \sum_{i=1}^{K} \sum_{j=1}^{k_i} \frac{B_{ij}}{p_i^j} + \sum_{w_i \text{ in } F} c_i \int \frac{w_i'}{w_i} + \sum_{w_i \text{ not in } F} c_i \int \frac{w_i'}{w_i} \right.$$

$$\left. + \sum_{r_i=0} d_i \int \frac{u_i'}{v_i} + \sum_{r_i \neq 0} d_i \int \frac{(\theta + \beta_i/r_i)'}{(\theta + \beta_i/r_i)} a^{r_i} \prod_{l'} a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}} \right]'$$

where $\beta_i = \sum_e r_{ij}a_j + \sum_{l'} r_{ij}\theta_j + v_i$ and the p_i are monic, irreducible elements of $F[\theta]$. We may assume by the uniqueness of the partial fraction decomposition that each w_i not in F equals some p_i, $1 \leqq i \leqq K$, and that each $\theta + \beta_i/r_i$ equals some p_i, $1 \leqq i \leqq K$.

We now proceed as in [Risch69] and calculate $B_{m+1}, \cdots, B_1$. This reduces the polynomial part of the problem to deciding if an expression of the form $A_0 - \bar{B}_1(a'/a)$, where $\bar{B}_1$ is known, has an antiderivative in some li-elementary extension of F. By the induction hypothesis this can be decided, i.e. apply the algorithm recursively to $A_0 - \bar{B}_1(a'/a)$ which is in F.

Next calculate $B_{1k_1}, \cdots, B_{11}, \cdots, B_{Kk_K}, \cdots, B_{K1}$ using the Hermite reduction scheme. This yields an equation of the form

$$\left(\frac{\bar{A}_{11}}{p_1} + \cdots + \frac{\bar{A}_{K1}}{p_K} \right) = \sum c_i \frac{p_i'}{p_i} + \sum d_i \frac{(\theta + \beta_i/r_i)'}{(\theta + \beta_i/r_i)} a^{r_i} \prod_{l'} a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}}$$

where $\bar{A}_{11}, \cdots, \bar{A}_{K1}$ are in $F[\theta]$ and $\deg_\theta(\bar{A}_{i1}) < \deg_\theta(p_i)$ for all i. By the uniqueness of the partial fraction decomposition, for each p_k, $1 \leqq k \leqq K$, there are two possibilities. If $\deg_\theta(p_k) > 1$ then $\bar{A}_{k1} = c_k p_k'$ must hold for some constant c_k, otherwise the integral does not exist. Consider therefore the case where p_k is linear in θ. Collecting together all terms with a denominator of p_k yields

$$\frac{\bar{A}_{k1}}{p_k} = c_k \frac{p_k'}{p_k} + \sum_{p_k} d_i \frac{p_k'}{p_k} a^{r_i} \prod_{l'} a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}}$$

where $p_k = \theta + \beta_i/r_i$ for each i in the above sum. Therefore

$$\frac{\bar{A}_{k1}}{p_k'} = c_k + \sum_{p_k} d_i a^{r_i} \prod_{l'} a_j^{r_{ij}} \prod_e \theta_j^{r_{ij}} \tag{5.10}$$

and so

$$\frac{(\bar{A}_{k1}/p_k')'}{p_k'} = \sum_{p_k} d_i r_i a^{r_i} \prod_{l'} a_j^{r_{ij}} \prod_{e} \theta_j^{r_{ij}}. \tag{5.11}$$

We can now determine r_i, r_{ij} and d_i using the results of § 4. We need only show that (5.11) is a restricted Σ-decomposition (where $\Sigma = (a, \cdots, a_i, \cdots, \theta_j, \cdots)$, $i \in l', j \in e$), and bound the degree. However β_i/r_i does not depend on i and can be written as $\sum_e R_j a_j + \sum_{l'} R_j \theta_j + v$ where $R_j = r_{ij}/r_i$ and $v = v_i/r_i$ for all i and j. Hence $r_{ij} = R_j r_i$ proving that (5.11) is a restricted Σ-decomposition of degree one. Once these values are known we determine c_k with (5.10). This concludes Case (a).

Case (b). All the monomials of rank r are exponential. Choose one, say θ, and let $E = F(\theta)$. Break γ into partial fractions to obtain

$$\gamma = A_m \theta^m + \cdots + A_{\bar{m}} \theta^{\bar{m}} + A_0 + \frac{P(\theta)}{Q(\theta)}$$

where $\bar{m} \leqq m$, $\deg_\theta (P) < \deg_\theta (Q)$ and $\gcd(\theta, Q(\theta)) = 1$. If γ has an antiderivative in some li-elementary extension of E then by Lemma 5.1 each of the terms $A_m \theta^m, \cdots, A_{\bar{m}} \theta^{\bar{m}}, A_0 + P(\theta)/Q(\theta)$ must have such an antiderivative.

Consider first the term $A_0 + P(\theta)/Q(\theta)$. From Theorem 2.2 we have

$$A_0 + \frac{P(\theta)}{Q(\theta)} = w_0' + \sum c_i \frac{w_i'}{w_i} + \sum d_i \frac{u_i'}{v_i}$$

where c_i, d_i are in C, and w_i, u_i, v_i are in E. By Lemma 5.1, u_i and v_i must be in F and we can therefore proceed exactly as in the exponential case of [Risch69]. This reduces the problem of deciding if $A_0 + P(\theta)/Q(\theta)$ has an antiderivative in some li-elementary extension of E to the problem of deciding if an expression of the form $A_0 + \sum \delta_i$, where the δ_i are known, has an antiderivative in some li-elementary extension of F. This is decided using the induction hypothesis.[3]

Consider next the term $A_j \theta'$, where $j \neq 0$. Two different types of reductions will be employed to integrate this expression depending on whether $m_r > 1$ or $m_r = 1$.

First suppose that $m_r > 1$. This implies that there is another exponential monomial of rank r. Denote it by ψ and let $F = D(\psi)$. (We know have $E = F(\theta) = D(\psi)(\theta)$.) Do a partial fraction decomposition of A_j with respect to ψ over the field D and write

$$A_j = A_{m_j} \psi^{m_j} + \cdots + A_{\bar{m}_j} \psi^{\bar{m}_j} + A_{0_j} + \frac{P_j(\psi)}{Q_j(\psi)}$$

where the A_{i_j} are in D, $\deg_\psi (P_j) < \deg_\psi (Q_j)$ and $\gcd(\psi, Q_j) = 1$. From this we see that the partial fraction decomposition of $A_j \theta^j$ over the field $D(\theta)$ with respect to ψ is

$$A_j \theta^j = A_{m_j} \theta^j \psi^{m_j} + \cdots + A_{\bar{m}_j} \theta^j \psi^{\bar{m}_j} + \theta^j A_{0_j} + \frac{\theta^j P_j(\psi)}{Q_j(\psi)}.$$

Applying Lemma 5.1 again we have that $A_j \theta^j$ will have an antiderivative in some li-elementary extension of $E = D(\theta)(\psi)$ if and only if each of the terms

$$A_{m_j} \theta^j \psi^{m_j}, \cdots, A_{\bar{m}_j} \theta^j \psi^{\bar{m}_j}, \theta^j A_{0_j} + \frac{\theta^j P_j(\psi)}{Q_j(\psi)}$$

[3] A careful analysis of this case reveals that, in fact, $P(\theta)/Q(\theta)$ must be elementary over F and, hence, the induction hypothesis can be applied to A_0. Notice also that this allows one to apply the Rothstein algorithm [Roth76] to $\int (P(\theta)/Q(\theta))$.

has such an antiderivative. The term $\theta^j A_{0_j} + \theta^j P_j(\psi)/Q_j(\psi)$ can be handled in the same manner as before. That is, follow the exponential case of the Risch algorithm and reduce the problem to deciding if a known expression in $D(\theta)$ has an antiderivative in a li-elementary extension of $D(\theta)$. This can be decided using the induction hypothesis. For the other terms, consider a general representative $A_{i_j}\theta^j\psi^{i_j}$. This expression will have an antiderivative in some li-elementary extension of E if and only if $A_{i_j}\Theta_{i_j}$, where $\Theta_{i_j} = \theta^j\psi^{i_j}$, has an antiderivative in some li-elementary extension of $D(\Theta_{i_j})$. This is true because E is an elementary extension of $D(\Theta_{i_j})$. Notice that, in the field $D(\Theta_{i_j})$, rank $(\Theta_{i_j}) \leqq r$ and therefore rank $(D(\Theta_{i_j})) \leqq (m_r - 1, \cdots, m_1, 1)$. Following our discussion of ranked and normalized fields from § 3 we replace $D(\Theta_{ij})$ with $\tilde{D}_{i_j}$ so that $\tilde{D}_{i_j}$ is isomorphic to $D(\Theta_{i_j})$ and $\tilde{D}_{i_j}$ is normalized. We then have

$$\text{rank}\,(\tilde{D}_{i_j}) \leqq \text{rank}\,(D(\Theta_{i_j})) \leqq (m_r - 1, \cdots, m_1, 1) < \text{rank}\,(E).$$

Now suppose that in the field $\tilde{D}_{i_j}$, $A_{i_j}\Theta_{i_j}$ has the representation $\tilde{A}_{i_j}$. Since rank $(\tilde{D}_i) <$ rank (E) we can apply the induction hypothesis to determine if $\tilde{A}_{i_j}$ has an antiderivative in some li-elementary extension of $\tilde{D}_{i_j}$ and then map the result back to E. This reduction is applied to each of the terms $A_{m_j}\theta^j\psi^{m_j}, \cdots, A_{\bar{m}_j}\theta^j\psi^{\bar{m}_j}$, concluding case where $m_r > 1$.

We may now assume that $m_r = 1$. Finding an antiderivative of $A_j\theta^j$ in some li-elementary extension of E is equivalent to finding an antiderivative of $A_j\theta^{(j)}$, where $\theta^{(j)} = \theta^j$, in some li-elementary extension of $F(\theta^{(j)})$. Again this is true because E is an elementary extension of $F(\theta^{(j)})$. Replace $\theta^{(j)}$ with $\tilde{\theta}^{(j)}$ so that $F(\tilde{\theta}^{(j)})$ is normalized and isomorphic to $F(\theta^{(j)})$. Now in the field $F(\tilde{\theta}^{(j)})$, $A_j\theta^j$ will have a representation of the form $\tilde{A}_j\tilde{\theta}^{(j)}$ where $\tilde{A}_j$ is some element of F. Moreover the rank of $F(\tilde{\theta}^{(j)})$ is either less than or equal to the rank of E. If rank $(F(\tilde{\theta}^{(j)})) <$ rank (E) then by the induction hypothesis we can determine if $\tilde{A}_j\tilde{\theta}^{(j)}$ has an antiderivative in some li-elementary extension of $F(\tilde{\theta}^{(j)})$. Otherwise rank $(F(\tilde{\theta}^{(j)})) =$ rank $(E) = (1, m_{r-1}, \cdots, 1)$ and in this case Theorem 5.3 is applicable. This concludes the case where $m_r = 1$. □

Example 5.1. Consider $\int (x^3/\log(x^2-1))\,dx$. First the integrand must be rewritten as $x^3/(\log(x+1)+\log(x-1))$ with the factored tower of monomials $C(x, \theta_1 = \log(x-1), \theta_2 = \log(x+1))$. We then follow Case (a) from the proof of Theorem 5.4 with $\theta = \theta_2$, $K = 1$, $\bar{A}_{11} = x^3$ and $p_1 = \log(x+1)+\log(x-1)$. With these values the Σ-decomposition in equation (5.11) becomes

$$\frac{(\bar{A}_{11}/p_1')'}{p_1'} = \frac{2x^4-3x^2+1}{2} = \sum d_i r_i (x+1)^{r_i}(x-1)^{r_{i1}}.$$

Also, since $\beta_i/r_i = R_1 \log(x-1) + v = \log(x-1)$, we have that the above Σ-decomposition is restricted by the function $g(r_i) = (r_i, R_1 r_i) = (r_i, r_i)$. Next follow the calculations described in the proof of Theorem 4.4. We begin with $\Phi_1 = (2x^4-3x^2+1)/2$ and so $r_1 = 1$, $r_{11} = 1$, $d_1 r_1 = \frac{1}{2}$. Form $\Phi_2 = \Phi_1 - d_1 r_1 (x+1)^{r_1}(x-1)^{r_{11}} = x^4 - 2x^2 + 1$ and calculate $r_2 = 2$, $r_{21} = 2$ and $d_2 r_2 = 1$. Since $\Phi_3 = \Phi_2 - (x+1)^2(x-1)^2 = 0$, we are done and conclude that

$$\int \frac{x^3}{\log(x^2-1)}\,dx = \frac{1}{2}\,\text{li}\,(x^4-2x^2+1) + \frac{1}{2}\,\text{li}\,(x^2-1). \qquad \square$$

Example 5.2. Consider $\int (x^2/\log(x^2-1))\,dx$. Proceeding as above we have

$$\int \frac{x^2}{\log(x^2-1)}\,dx = \int \frac{x^2}{\log(x+1)+\log(x-1)}\,dx$$

and

$$\frac{(\bar{A}_{11}/p_1')'}{p_1'} = \frac{3x^4-4x^2+1}{4x} = \sum d_i r_i (x+1)^{r_i}(x-1)^{r_{i1}}$$

where the Σ-decomposition is restricted by the function $g(r_i) = (r_i, r_i)$. In the application of Theorem 4.4 we have $\alpha_{11} = r_1 = 1$. Since $g(r_i)$ is strictly monotone in each of its components, $r_1 = 1 > 0$, and Φ_1 is not a polynomial in x, we are able to apply the termination strategy in Case (ii) and conclude that the above Σ-decomposition does not exist. Therefore, $\int (x^2/\log(x^2-1))\,dx$ can not be written in terms of elementary functions and logarithmic integrals. □

Example 5.3. Consider

$$\int \left[\frac{2x+3}{3\log(x)+2x} e^{\log(x)/2+x} + \frac{1}{x+1}(e^{\log(x)/2+x})^2 \right] dx.$$

The tower of monomials here is $C(x, \log(x), e^{\log(x)/2+x})$ and so, in the proof of Theorem 5.4, Case (b) applies. In this case, Lemma 5.1 allows us to consider the problem in two parts.

(i) $\int ((2x+3)/(3\log(x)+2x)\, e^{\log(x)/2+x})\,dx$. This is an instance of the reduced problem described in Theorem 5.3. In the proof of that theorem, we apply Case (b2), where $r = \frac{1}{2}$, $s = x$, $K = 1$, $\bar{A}_{11} = (2x+3)/3$ and $p_1 = \log(x) + 2x/3$. The Σ-decomposition in equation (5.6) becomes

$$\frac{\bar{A}_{11}}{p_1'} = x = \sum d_i x^{r_i}.$$

We have $d_1 = r_1 = 1$ and so

$$\int \frac{2x+3}{3\log(x)+2x} e^{\log(x)/2+x}\,dx = d_1 \operatorname{li}(x^{r_1} e^{\log(x)/2+x}) = \operatorname{li}(x\, e^{\log(x)/2+x}).$$

(ii) $\int 1/(x+1)(e^{\log(x)/2+x})^2\,dx$. Here the problem is posed in the field $C(x, \theta_1 = \log(x), \theta_2 = e^{\log(x)/2+x})$, which has rank (1, 1, 1). We first define $\theta_2^{(2)}$ as θ_2^2 and write the integrand as $\theta_2^{(2)}/(x+1)$ in the field $C(x, \theta_1 = \log(x), \theta_2^{(2)} = e^{\log(x)+2x})$. Next replace these monomials with a normalized tower of monomials yielding the integrand $x/(x+1)\, e^{2x}$ in the field $C(x, \theta_1 = \log(x), \tilde{\theta}_2^{(2)} = e^{2x})$. The rank of this field is (2, 1) which allows us to apply the induction hypothesis. Since $\theta_1 = \log(x)$ does not occur in the integrand, we shall consider $\int x/(x+1)\, e^{2x}\,dx$ in the field $C(x, e^{2x})$. This is an instance of Case (a) of the reduced problem (Theorem 5.3). In this case $\theta = x$, $p(\theta)/q(\theta) = 2x$, $A = x/(x+1)$ and β is a constant. The only factor of the denominator of A is $x+1$ and, by Lemma 5.2, we let $\beta = 2$ so that $x+1$ divides $2x+\beta$. Hence, $u_1 = e^{2x}$ and $v_1 = 2x+2$ are the only values generated for u_1 and v_1. (Notice at this point that $v_1 = \log(u_1)$ yet $v_1 \neq \ln(u_1)$.) [Risch69 Main Theorem, part (b)] is applied next to solve

$$\frac{x}{x+1} e^{2x} = (B\, e^{2x})' + d_1 \frac{(2x)'\, e^{2x}}{2x+2}.$$

This yields $B = \frac{1}{2}$ and $d_1 = -1$. Finally, we replace u_1 with $\bar{u}_1 = e^{2x+2}$ and d_1 with $\bar{d}_1 = -1/e^2$ so that $v_1 = \ln(\bar{u}_1)$ (see the discussion on transcendental constants in § 2) and write

$$\int \frac{x}{x+1} e^{2x} = \frac{e^{2x}}{2} - e^{-2} \operatorname{li}(e^{2x+2}).$$

□

Example 5.4. Consider

$$\int \left(\frac{2x^3 - x^2 - 6x}{x^2 + 3x + 2} + \frac{2x - 3}{\log(x) + 1} \right) e^{x \log(x) + x} \, dx.$$

This is again an example of Case (b2), Theorem 5.3. We have $r = x$, $s = x$, $\bar{A}_{11} = 2x - 3$ and $p_1 = \log(x) + 1$. The Σ-decomposition in (5.6) becomes

$$\frac{\bar{A}_{11}}{p_1'} = 2x^2 - 3x = \sum d_i x^{r_i}.$$

This yields $d_1 = -3$, $r_1 = 1$, $d_2 = 2$, $r_2 = 2$, and so

$$u_1 = x\, e^{x \log(x) + x}, \qquad v_1 = x \log(x) + x + \log(x) + 1$$

and

$$u_2 = x^2\, e^{x \log(x) + x}, \qquad v_2 = x \log(x) + x + 2 \log(x) + 2.$$

We next apply [Risch69, part (b)], and see that

$$\left(\frac{2x^3 - x^2 - 6x}{x^2 + 3x + 2} + \frac{2x - 3}{\log(x) + 1} \right) e^{x \log(x) + x} = -3 \frac{u_1'}{v_1} + 2 \frac{u_2'}{v_2}.$$

Finally, letting $\bar{u}_1 = e u_1$ and $\bar{u}_2 = e^2 u_2$ yields

$$\int \left(\frac{2x^3 - x^2 - 6x}{x^2 + 3x + 2} + \frac{2x - 3}{\log(x) + 1} \right) e^{x \log(x) + x} \, dx$$

$$= -\frac{3}{e} \operatorname{li}\left(e^{x \log(x) + x + \log(x) + 1} \right) + \frac{2}{e^2} \operatorname{li}\left(e^{x \log(x) + x + 2 \log(x) + 2} \right). \qquad \square$$

6. Conclusions. The importance of this work depends, in part, on whether the ideas presented here can be readily extended to other special functions. In [Cher83] a decision procedure, based on similar techniques, for integrating a large class of transcendental elementary expressions in terms of elementary functions and error functions:

$$\operatorname{erf}(u) = \int u'\, e^{-u^2} \, dx$$

is given (also see [Checa83]). Without any further work, however, Theorem 5.4 can be applied to a few other special functions due to the following relationships.

(i) The exponential integral:

$$\operatorname{ei}(u) = \int \frac{u' \exp(u)}{u} \, dx = \operatorname{li}(\exp(u)).$$

(ii) The sine and cosine integrals:

$$\operatorname{si}(u) = \int \frac{u' \sin(u)}{u} \, dx = \frac{1}{2i} [\operatorname{ei}(iu) - \operatorname{ei}(-iu)],$$

$$\operatorname{ci}(u) = \int \frac{u' \cos(u)}{u} \, dx = \frac{1}{2} [\operatorname{ei}(iu) + ei(-iu)].$$

For example

$$\int \frac{\cos(x)^2}{x^3}\,dx = -\frac{2x^2\,\mathrm{ei}(2ix)+2x^2\,\mathrm{ei}(-2ix)-2x\sin(2x)+\cos(2x)+1}{4x^2}$$

$$= -\mathrm{ci}(2x)+\frac{\sin(2x)}{2x}-\frac{\cos(2x)}{4x^2}-\frac{1}{4x^2}.$$

Appendix. An implementation of an algorithm for integrating a class of elementary expressions in terms of logarithmic integrals. In the following Macsyma demonstration, the command "int" will accept as input any element γ, of a transcendental elementary field F, and will attempt to determine if γ has an antiderivative in some li-elementary extension of F. This procedure is not a complete implementation of the decision procedure described in § 5. There are two exceptions:

(i) We have only implemented the base case of Risch's Main Theorem, part (b). Therefore in Theorem 5.3 only Case (a) where $\theta = x$ is covered.

(ii) We have not assumed that the constant field is algebraically closed. Thus the exponential case demonstrated below will not always recognize logarithmic integrals involving new algebraic constants. An example of this will be given.

```
/*Some Examples of Integration with Logarithmic Integrals*/
(c1) int (x/log (x)^2, x);
```

Time = 1216 msec.

$$\int \frac{x}{\log(x)^2}\,dx = 2\,\mathrm{li}(x^2)-\frac{x^2}{\log(x)} \tag{d1}$$

```
(c2)/* The following example shows how transcendental constants can be introduced. Int first
      calculates the values u1 = x, v1 = log (x)+3 and d1 = 1. In the last stage of the computations
      u1 is replaced with ū1 = λ1 u1 = e^3 x and d1 is replaced with d1/λ1 = 1/e^3, */
int (1/(log (x)+3), x);
```

Time = 983 msec.

$$\int \frac{1}{\log(x)+3}\,dx = e^{-3}\,\mathrm{li}(e^3x) \tag{d2}$$

```
(c3)/* Writing 3x^3+5x^2+2x as a Σ-decomposition where Σ = (x+1) (see (5.11)), yields the following
      expansion of logarithmic integrals: */
  int (x^2/log (x+1), x);
```

Time = 1900 msec.

$$\int \frac{x^2}{\log(x+1)}\,dx = \mathrm{li}(x^3+3x^2+3x+1)-2\,\mathrm{li}(x^2+2x+1)+\mathrm{li}(x+1) \tag{d3}$$

```
(c4)/* It follows from Theorem 5.5 that any expression of the form ∫ u'g(log (u)) dx where g is a
      rational function with constant coefficients can be integrated in terms of elementary functions
      and logarithmic integrals. For example */
int ((log (x)^2+3)/(log (x)^2+3* log (x)+2), x);
```

Time = 2233 msec.

$$\int \frac{\log(x)^2+3}{\log(x)^2+3\log(x)+2}\,dx = -7e^{-2}\,\mathrm{li}(e^2x)+4e^{-1}\,\mathrm{li}(ex)+x \tag{d4}$$

(c5)/* Macsyma can also differentiate expressions involving logarithmic integrals. ("%" refers to the previous expression) */

li (x/log (x)* exp (x));

Time = 50 msec.

$$\mathrm{li}\left(\frac{xe^x}{\log(x)}\right) \tag{d5}$$

(c6) ratsimp (diff(%, x));

Time = 516 msec.

$$\frac{(x+1)e^x \log(x) - e^x}{\log(x)^2 \log(xe^x/\log(x))} \tag{d6}$$

(c7) /* Now integrate this expression: */

int (%, x);

Time = 8266 msec.

$$\int \frac{(x+1)e^x \log(x) - e^x}{\log(x)^2 \log(xe^x/\log(x))}\,dx = \mathrm{li}\left(\frac{xe^x}{\log(x)}\right) \tag{d7}$$

(c8)/* Since li (u) = ei(e^u) (cf. § 6), int will use the "ei" notation in these situations. For example */

int (exp (x)/(x+1)^2, x);

Time = 3350 msec.

$$\int \frac{e^x}{(x+1)^2}\,dx = e^{-1}\,\mathrm{ei}(x+1) - \frac{e^x}{x+1} \tag{d8}$$

(c9)/* The sine integral can be integrated in terms of logarithmic integrals: */

int (sin (x)/x, x);

Time = 8216 msec.

$$\int \frac{\sin(x)}{x}\,dx = -\frac{i\,\mathrm{ei}(ix) - i\,\mathrm{ei}(-ix)}{2} \tag{d9}$$

(c10)/* as well as a number of other trigonometric integrands: */

int (cos (x)^2 / x^3, x);

Time = 8750 msec.

$$\int \frac{\cos(x)^2}{x^3}\,dx = -\frac{2x^2\,\mathrm{ei}(2ix) + 2x^2\,\mathrm{ei}(-2ix) - 2x\sin(2x) + \cos(2x) + 1}{4x^2} \tag{d10}$$

(c11)/* Any expression of the form $\int u'e^u g(u)\,dx$ where g is a rational function with constant coefficients can be integrated in terms of logarithmic integrals: */

int ((x^2+3)/(x^2+3*x+2)* exp (x), x);

Time = 6800 msec.

$$\int \frac{(x^2+3)e^x}{x^2+3x+2}\,dx = -7e^{-2}\,\mathrm{ei}(x+2) + 4e^{-1}\,\mathrm{ei}(x+1) + e^x \tag{d11}$$

(c12)/* From deficiency (ii) above, however, int will miss some logarithmic integrals: */

int ((x^2+1)/(x^2+x+1)* exp (x), x);

Time = 7100 msec.

$$\int \frac{(x^2+1)e^x}{x^2+x+1}\,dx = \int \frac{(x^2+1)e^x}{x^2+x+1}\,dx \tag{d12}$$

Acknowledgments. Bob Caviness suggested the original problem and I am grateful for the many helpful discussions we have had during the course of this work. I have also benefited greatly from the suggestions of Michael Singer, David Saunders and James Davenport.

REFERENCES

[Bate53] H. BATEMAN, *Higher Transcendental Functions*, McGraw-Hill, New York, 1953.

[Cher83] G. W. CHERRY, *Algorithms for integrating transcendental elementary functions in terms of logarithmic integrals and error functions*, Ph.D. dissertation, Univ. Delaware, Newark, 1983.

[Checa83] G. W. CHERRY AND B. F. CAVINESS, *Integration in finite terms with special functions: a progress report* in Proc. 1984 ACM Symposium on Symbolic and Algebraic Computing, J. Fitch, ed.

[Davtr81] J. H. DAVENPORT AND B. M. TRAGER, *Factorization over finitely generated fields* in Proc. 1981 ACM Symposium on Symbolic and Algebraic Computation, P. S. Wang, ed.

[Risch69] R. RISCH, *The problem of integration in finite terms*, Trans. Amer. Math. Soc., 139 (1969), pp. 167–189.

[Roca79] M. ROTHSTEIN AND B. F. CAVINESS, *A structure theorem for exponential and primitive functions*, this Journal, 8 (1979), pp. 357–367.

[Ros76] M. ROSENLICHT, *On Liouville's theory of elementary functions*, Pacific J. Math., 65 (1976), pp. 485–492.

[Rosi77] M. ROSENLICHT AND M. F. SINGER, *On elementary, generalized elementary and Liouvillian extension fields* in Contributions to Algebra, Bass, Cassidy and Kovacic, eds., Academic Press, New York, 1977, pp. 329–342.

[Roth76] M. ROTHSTEIN, *Aspects of symbolic integration and simplification of exponential and primitive functions*, Ph.D. dissertation, Univ. Wisconsin, Madison, 1976.

[Ssc81] M. F. SINGER, B. D. SAUNDERS AND B. F. CAVINESS, *An extension of Liouville's theorem on integration in finite terms*, this Journal 14 (1985), pp. 966–990. Extended Abstract in Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, P. S. Wang, ed.

[Vdw50] B. L. VAN DER WAERDEN, *Modern Algebra*, second ed., Fredrick Ungar, New York, 1950.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
002

AN AMORTIZED ANALYSIS OF INSERTIONS INTO AVL-TREES*

KURT MEHLHORN† AND ATHANASIOS TSAKALIDIS†

Abstract. We analyse the amortized behavior of AVL-trees under sequences of insertions. We show that the total rebalancing cost (=balance changes) for a sequence of n arbitrary insertions is at most $2.618n$. For random insertions the bound is improved to $2.26n$. We also show that the probability that t or more balance changes are required decreases exponentially with t.

Key words. balance search trees, insertions, AVL-trees, balance changes, amortized number, expected number

1. Introduction. As for many balanced tree schemes an insertion into an AVL-tree consists of a search down the tree followed by a rebalancing phase which works its way back to the root. Rebalancing is usually restricted to a terminal segment of the search paths. Experiments (cf. [5]) suggest that the expected length of this terminal segment is less than two, however, there is no theoretical evidence to support that claim. A first attempt of an analysis was made by C. C. Foster (see [3], cf. also Knuth [6, p. 462]). Although his analysis predicts the expected length of the terminal segment fairly well (he predicts expected length 1.85), it is not precise in a mathematical sense.

In this paper we give a *rigorous* analysis. More precisely, we prove in § 3, that the amortized length of the terminal segment is at most 2.618. That is, if we consider an arbitrary (not random) sequence of n insertions into an initially empty AVL-tree then the total length (summed over the n insertions) of the terminal segments is at most $2.618n$. We also prove that this bound is sharp by exhibiting a sequence of n insertions where the total number of balance changes is essentially $2.618n$. Our main insight is to concentrate on amortized behavior rather than expected behavior. This led us to stronger results (our results hold for arbitrary not just random sequences of insertions) and suggested to use combinatorial (not probabilistic) methods of analysis.

For sequences of random insertions we can slightly improve the bound in § 4. The expected length of the terminal segment is at least 1.47 and at most 2.26. Finally in § 5 we prove a result on the distribution of the length of the terminal path. We show that the probability that the length exceeds t decreases exponentially in t. Section 2 contains some definitions.

We close the introduction with a brief discussion of related work. Brown [2] and Mehlhorn [7], [8] studied the expected number of balanced nodes in random AVL-trees. We use their results in § 4. The amortized cost of insertions and deletions into (a, b)-trees was studied by Huddleston and Mehlhorn [4]. They prove that the amortized number of node splittings and fusings is $0(1)$ provided that $b \geqq 2a$. Finally, we want to mention that Tsakalidis [9] proves a result which is analogous to the one presented here for sequences of deletions. He shows that the total number of balance and structural changes in a sequence of deletions applied to an AVL-tree with n leaves is $1.618n$.

2. Definitions. AVL-trees were introduced by Adel'son-Vel'skii and Landis [1] in 1962. AVL-trees are binary trees in which nodes either have two sons or no sons. The

* Received by the editors August 31, 1983, and in revised form September 1, 1984.

† Fachbereich 10, Angewandte Mathematik und Informatik, Universität des Saarlandes, D-6600 Saarbrücken, West Germany.

latter nodes are called leaves. A binary search tree is AVL if the heights of the subtrees at each node differ by at most one, where the height Height (v) of node v is equal to the length of the longest path from v to a leaf.

Let $L(v)$ [$R(v)$] be the left [right] subtree of the tree with root v. For every node v we define its height balance $hb(v)$ by

$$hb(v) = \text{Height}(R(v)) - \text{Height}(L(v)).$$

Hence the height balance can be $+1$, 0, or -1. We call a node balanced (unbalanced) if its height balance is 0 (± 1).

For every insertion we define the *critical node* as the last unbalanced node on the search path. We give the last definition more formally:

Let $v_0, v_1, \cdots, v_k$ be a path from the root v_0 to a leaf v_k of an AVL-tree. Let i be minimal such that $hb(v_i) = hb(v_{i+1}) = \cdots = hb(v_k) = 0$. Then node v_{i-1} is called the *critical node* of the path (if $i \geqq 1$) and $v_i, \cdots, v_k$ is called the *critical path*. The *length* of the critical path is $k - i$ (and this is equal to the height of node v_i). For $i = 0$ no critical node is defined since this insertion causes a height increase of the tree.

We use the insertion algorithm described in Knuth [6, p. 455]. It can be summarized as follows:

At first the leaf at which the new element is to be inserted is located and this leaf is replaced by a tree with two leaves. Next the height balance of nodes on the critical path are changed from 0 to ± 1 (this corresponds to step A_6 in Knuth [6]). Finally rebalancing is completed by absorption, single rotation or double rotation at the critical node or a height increase of the tree if no critical node exists.

Let T_0 be the empty AVL-tree, i.e. the AVL-tree with 0 leaves. We consider sequences of n insertions into T_0. Let T_i, $0 \leqq i \leqq n$, be the AVL-tree obtained after the ith insertion (and completed rebalancing). We are interested in the following quantities:

x_1, the total number of balance changes $0 \to \pm 1$ in step A_6,

x_2, the total number of absorptions,

x_3, the total number of single rotations,

x_4, the total number of double rotations,

x_5, the total number of height increases.

We will also refer to the 5 operations (balance change, absorption, single rotation, double rotation, height increase) as $Op_1, Op_2, \cdots, Op_5$. Since every insertion terminates in an absorption, single or double rotation or height increase we clearly have

$$x_2 + x_3 + x_4 + x_5 = n.$$

Quantity x_1 is harder to estimate. Let l_i, $1 \leqq i \leqq n$, be the length of the critical path for the ith insertion. Then

$$x_1 = \sum_{i=1}^{n} l_i.$$

We will estimate x_1 for arbitrary sequences of insertions in § 3 and for random sequences in § 4. Finally let Val (T_i) be the total number of unbalanced nodes in AVL-tree T_i.

3. The total number of balance changes in arbitrary sequences of insertions. Our main tool for getting bounds on x_1, the number of balance changes $0 \to \pm 1$, is Lemma 1 below. In this lemma we relate quantities x_1, x_3, x_4, x_5, the number of insertions n and the value Val (T_n) of the final tree.

LEMMA 1. *Consider an arbitrary sequence of n insertions into an initially empty tree. Then*

$$x_1 = \text{Val}(T_n) + n + x_3 + x_4 - x_5.$$

Proof. We exactly estimate the rate of the increase of Val (T_i) during the transition from T_{i-1} to T_i with respect to the alternative operations.

We give the scheme how the node's balance on the critical path will be changed. In our diagrams we will always show an insertion into the left subtree of the critical node.

Case 1. Absorption.

In the figures ± 1 denotes a node with balance $+1$ or -1 and $\square$ denotes a leaf. The figure shows an insertion into the left subtree of the critical node, the other case being symmetric.

With respect to Fig. 1 the following holds:

$$\text{Val}(T_i) = \text{Val}(T_{i-1}) + (l_i - 1). \tag{1}$$

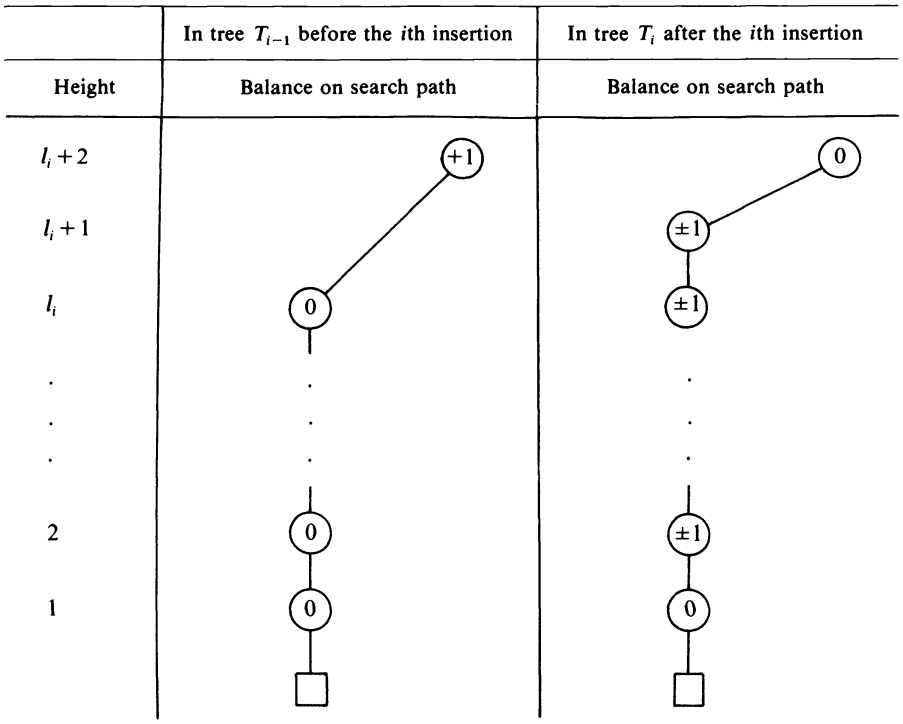


FIG. 1

Case 2. Reconstruction of the tree.

Case 2.1. Single rotation.

Fig. 2 illustrates this case and the following holds:

$$\text{Val}(T_i) = \text{Val}(T_{i-1}) + (l_i - 2). \tag{2}$$

Case 2.2. Double rotation is analogous to Case 2.1. Among the 3 top nodes of the reconstructed subtree there are exactly 2 nodes with Balance 0.

Case 3. Height increase (l_i is the height of the root), i.e. no critical node exists.

Fig. 3 illustrates this case and the following holds:

$$\text{Val}(T_i) = \text{Val}(T_{i-1}) + l_i. \tag{3}$$

We apply the recursive equations (1), (2) and (3) n times, and we get with respect to

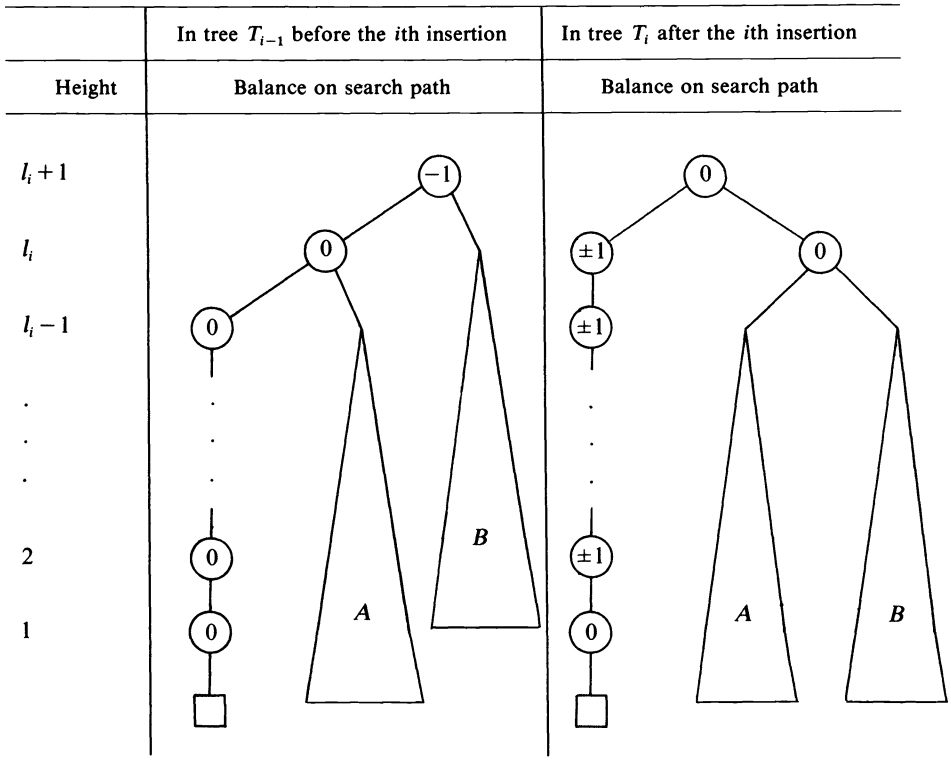

In tree T_{i-1} before the ith insertion
In tree T_i after the ith insertion
Height
Balance on search path
Balance on search path
l_i+1
l_i
l_i-1
2
1
−1
0
±1
A
B

FIG. 2

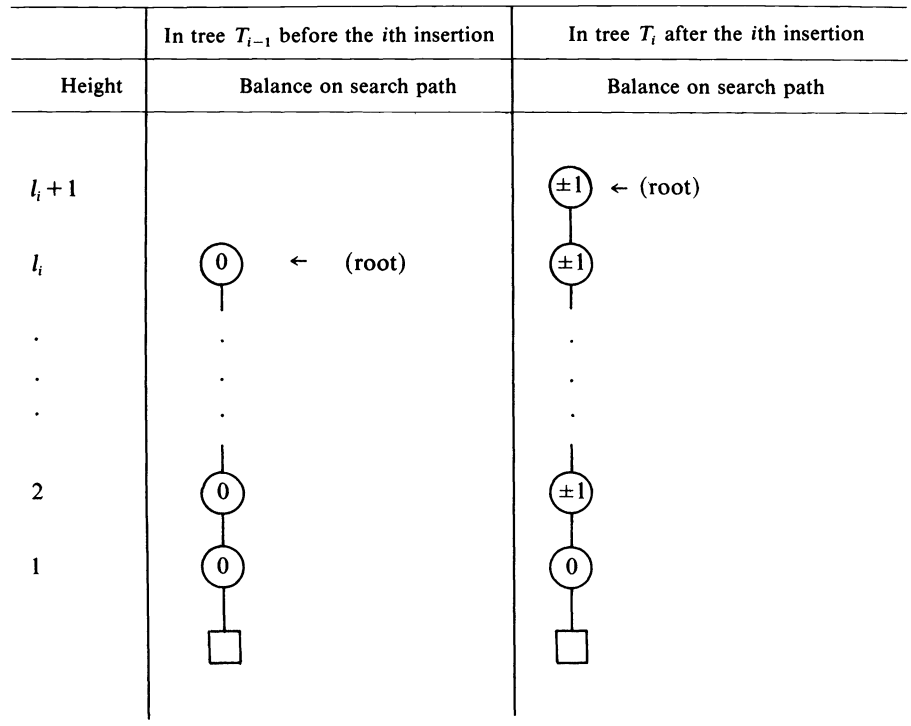

In tree T_{i-1} before the ith insertion
In tree T_i after the ith insertion
Height
Balance on search path
Balance on search path
l_i+1
l_i
2
1
0
±1
← (root)

FIG. 3

the definitions of x_i $(1 \leqq i \leqq 5)$:

$$\text{Val}(T_n) = \text{Val}(T_0) + \sum_{i=1}^{n} l_i - x_2 - 2(x_3 + x_4).$$

But $\sum_{i=1}^{n} l_i = x_1$ and $\text{Val}(T_0) = 0$ and thus

$$x_1 = \text{Val}(T_n) + x_2 + 2(x_3 + x_4).$$

Using $x_2 + x_3 + x_4 + x_5 = n$ we obtain

$$x_1 = \text{Val}(T_n) + n + (x_3 + x_4 - x_5). \qquad \square$$

Since $\text{Val}(T_n) \leqq n$ and $x_3 + x_4 \leqq n$ we infer $x_1 \leqq 3n$ from Lemma 1, i.e. the amortized length of the critical path is at most 3. In Lemma 2 below we recall a better upper bound on $\text{Val}(T_n)$ and so improve the upper bound on x_1.

LEMMA 2 (Knuth). *Let T_n be an* AVL*-tree with n leaves. Then* $\text{Val}(T_n) \leqq (\phi - 1)(n - 1)$ *where* $\phi = (1 + \sqrt{5})/2 \simeq 1.618$.

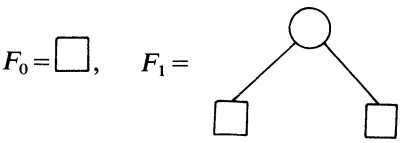
Proof. cf. [6, *exercise* 6.2.3.3]. $\square$

THEOREM 1. *The total number x_1 of balance changes in step* A6 *of the insertion algorithm in a sequence of n arbitrary insertions into an initially empty tree satisfies*

$$x_1 \leqq 2.618n.$$

Proof. Immediate from Lemma 1 and $x_3 + x_4 \leqq n$, $x_5 \geqq 0$ and $\text{Val}(T_n) \leqq 0.618n$ (Lemma 2). $\square$

How good is the bound given in Theorem 1? Note first that the bound given in Lemma 2 is sharp: Fibonacci trees have exactly that number of unbalanced nodes. Fibonacci trees $F_0, F_1, F_2, \cdots$ are defined as follows:



and F_{h+2} consists of a root and a copy of F_h and F_{h+1} each as left and right subtree respectively. This specifies the Fibonacci trees up to the left-right symmetry. Also F_h has exactly $\text{Fib}(h+1)$ leaves where $\text{Fib}(0) = \text{Fib}(1) = 1$ and $\text{Fib}(h+2) = \text{Fib}(h+1) + \text{Fib}(h)$ for $h \geqq 0$.

Besides the upper bound on $\text{Val}(T_n)$ given by Lemma 2 we use two more inequalities in the proof of Theorem 1: $x_3 + x_4 \leqq n$ and $x_5 \geqq 0$. We will next show that these bounds cannot be improved upon in general.

LEMMA 3. *There is a sequence of* $\text{Fib}(h)$ *insertions into F_h such that*:

1) *Every insertion (but the last) is terminated by a rotation or double rotation.*
2) *The last insertion leads to a height increase of the entire tree.*
3) *The final tree is F_{h+1}.*

Proof. The claim is obviously true for $h = 0, 1$ and 2, namely we can go from F_0 to F_1 (F_1 to F_2) with one insertion which leads to an increase in height and from F_2 to F_3 with two insertions one of which leads to a rotation and one of which leads to a height increase as shown in Fig. 4.

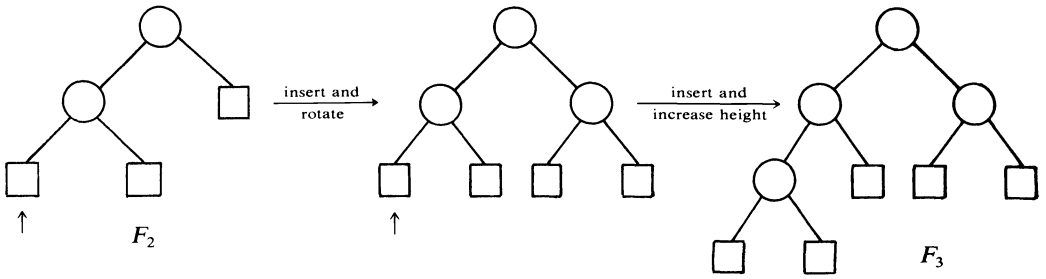
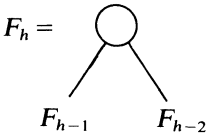


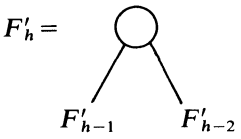
FIG. 4

For $h \geqq 3$ we proceed by induction. So consider

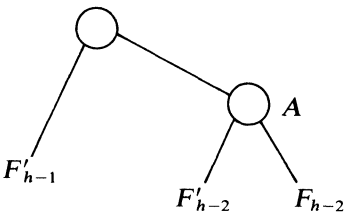


We will first use Fib $(h-1)$ insertions in order to change the left (w.l.o.g.) subtree F_{h-1} into an F_h (which we denote F'_h to distinguish it from our starting tree). Note that by induction hypothesis all but the last insertion leads to a rotation/double rotation *inside* the left subtree. The last insertion will turn the left subtree into an F'_h and increase the height of the left subtree to h, thus moving the root out of balance. Next we will rotate or double rotate at the root.

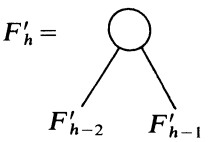
Case 1. Rotation, i.e.



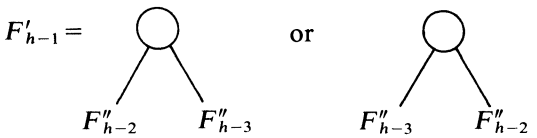
Rotation yields



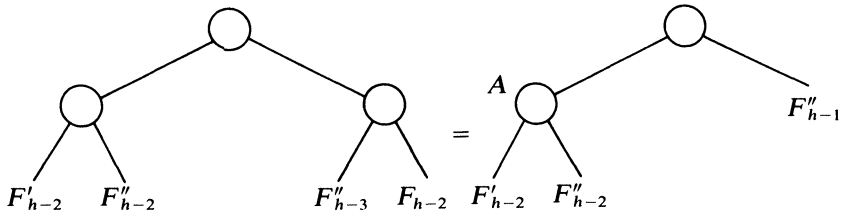
Case 2. Double rotation, i.e.



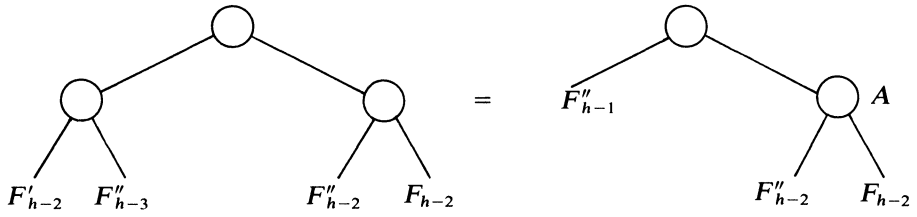
We have to distinguish two cases: whether F'_{h-1} is tilted to the left or right, i.e.



In the first case double rotation yields



In the second case double rotation yields



In either case we obtain up to left right symmetry the same tree: a balanced root whose subtrees are a F_{h-1} and a balanced node A with two copies of F_{h-2}.

Next we perform Fib $(h-2)$ insertions into F_{h-2}. All but the last insertion will lead to rotations/double rotations within that tree without increasing the height. The last insertion creates a F_{h-1} and moves node A and the root out of balance thus increasing the total height of the tree. Altogether we created an F_{h+1} out of F_h by

$$\text{Fib}\,(h-1)+\text{Fib}\,(h-2)=\text{Fib}\,(h)$$

insertions; all insertions (but the last) lead to rotations or double rotations and the last insertion increased the height. □

THEOREM 2. *There are infinitely many n and sequences of n insertions such that the total number of balance changes in step* A6 *of the insertion algorithm satisfies*

$$x_1 \geqq 2.618n - O(\log n).$$

Proof. Let $n=\text{Fib}\,(h+1)$ for some h. Start with the empty tree and build $F_0, F_1, \cdots, F_h$ as described in Lemma 3. Then $x_2=0$ since absorption never occurs and hence $x_3+x_4+x_5=n$. Also $x_5=h=O(\log n)$ and $\text{Val}\,(T_n)=(\phi-1)(n-1)$. □

Theorems 1 and 2 together give complete information about the amortized cost of step A_6 of the insertion algorithm in the worst case. The amortized length of the critical path is 2.618 in the worst case and thus very small. We want to stress again that this bound holds for *arbitrary* sequences of insertions. Experimental data (Foster [3], Knuth [6], Karlton et al. [5]) is only available for random sequences of insertions. It suggests that the expected length of the critical path under random insertions is about 1.8 and hence only slightly less than amortized length.

4. The total number of balance changes under *n* random insertions.

Randomness assumption. Consider a tree T with $n-1$ nodes in it and hence with n leaves. These $n-1$ keys divide all possible key values into n intervals. The insertion of a new key k into T is said to be a random insertion if k has equal probabilities for being in any one of the n intervals defined above. In other words, each leaf has equal probability of being split into two.

Mehlhorn [7] has analysed the fringe of an AVL-tree under n random insertions refining work of Brown [2]. The fringe consists of the 3 types of subtrees shown in Fig. 5.

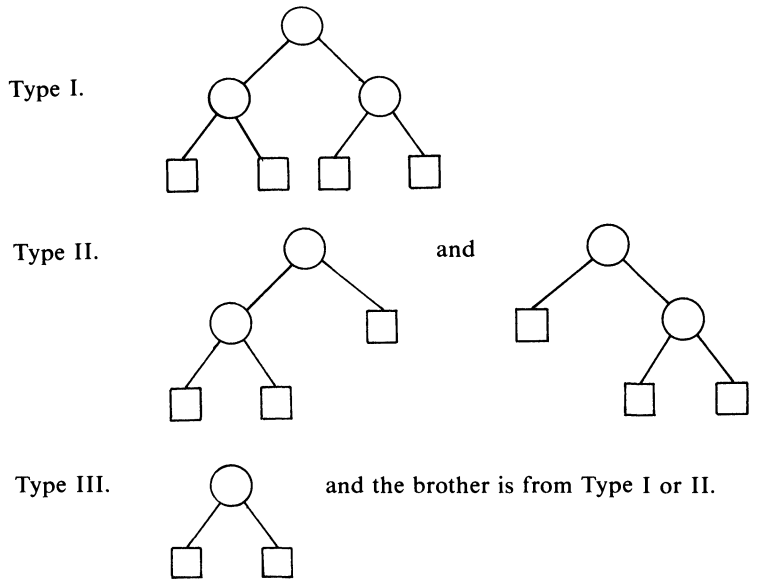


FIG. 5

We quote two results from [7].

THEOREM 3 [7]. *Let $a_i(n)$, for $1 \leqq i \leqq 3$, be the respective number of the above subtree of type i in a random* AVL*-tree T with n leaves. Then*

$$\begin{pmatrix} a_1(n) \\ a_2(n) \\ a_3(n) \end{pmatrix} = \frac{1}{35-28p(n)} \begin{pmatrix} 3 \\ 5-4p(n) \\ 4-8p(n) \end{pmatrix} n + \begin{pmatrix} o(n) \\ o(n) \\ o(n) \end{pmatrix}$$

with $0 \leqq p(n) \leqq \frac{4}{11}$ and $p(n)$ is the probability that the brother of a type I *subtree is of type* III *in a random* AVL*-tree with n leaves.*

THEOREM 4 [7]. *Let $\overline{\mathrm{Val}}(n)$ be the expected number of unbalanced nodes in a random* AVL*-tree with n leaves. Then*

$$(\tfrac{17}{91})n - o(n) \leqq \overline{\mathrm{Val}}(n) \leqq (\tfrac{17}{35})n + o(n).$$

Proof. In [7] it is shown that for the average number $\bar{B}(n)$ of balanced nodes in a random AVL-tree with n leaves the following holds:

$$(\tfrac{18}{35})n + o(n) \leqq \bar{B}(n) \leqq (\tfrac{74}{91})n + o(n).$$

Since $\overline{\mathrm{Val}}(n) + \bar{B}(n) = n$ we get the bounds of $\overline{\mathrm{Val}}(n)$: □

THEOREM 5. *The expected number $\bar{x}_1$ of balance changes in step* A6 *of the insertion algorithm in a sequence of n random insertions into an initially empty* AVL*-tree satisfies*

$$1.47n - o(n) \leqq \bar{x}_i \leqq 2.26n + o(n).$$

Proof. Let $p_j(i)$ denote the probability of the execution of operation Op_j (see § 2) during the $(i+1)$th insertion into a random AVL-tree with i leaves $(2 \leqq j \leqq 5)$. Then

according to Theorem 3 we have

$$p_2(i) \geqq \text{Probability of the absorptions on the fringe}$$
$$= \frac{a_2(i)+2a_3(i)}{i} \geqq \min_{0\leqq p\leqq 4/11}\left\{\frac{5-4p+2(4-8p)}{35-28p}\right\} - o(1) = 0.230 - o(1).$$

Also

$$p_3(i)+p_4(i) \geqq \text{probability of single and double rotation on the fringe}$$
$$\geqq (2a_2(i)+4a_1(i)\cdot p(i))/i - o(1)$$
$$\geqq \min_{0\leqq p\leqq 4/11}\left\{\frac{2(5-4p)+4\cdot 3\cdot p)}{35-28p}\right\} - o(1) = 0.285 - o(1).$$

This can be seen as follows: a rotation will always happen when the insertion is in one of the two deep leaves of a Type II tree (probability $2 \cdot a_2(i)/i$) and if insertion is in any leaf of a type I tree whose brother is a Type III tree (probability $4a_1(i) \cdot p(i)/i$).

Furthermore, $p_2(i)+p_3(i)+p_4(i)+p_5(i) = 1$ and hence

$$0.285 - o(1) \leqq p_3(i)+p_4(i) \leqq 0.770 - p_5(i) + o(1),$$
$$= 0.770 + o(1),$$

since $p_5(i) = o(1)$.

For the $(i+1)$*th* insertion we define the random variable y_i as follows $(0 \leqq i \leqq n-1)$

$$y_1 = \begin{cases} 1 & \text{if the } (i+1)\text{th random insertion into a random AVL-tree with} \\ & i \text{ leaves causes a single or double rotation,} \\ 0 & \text{otherwise.} \end{cases}$$

The expectation $E(y_i)$ of random variables y_i is equal to $p_3(i)+p_4(i)$ and hence

$$0.285 - o(1) \leqq E(y_i) \leqq 0.770 + o(1).$$

Let $\bar{x}_{3,4}$ denote the expected number of single or double rotations in a sequence of n random insertions, and let $\bar{x}_5$ be the expected number of height increases. Then

$$\bar{x}_{3,4} = \sum_{i=0}^{n-1} E(y_i)$$

and hence

$$0.285n - o(n) \leqq \bar{x}_{3,4} \leqq 0.770n + o(n).$$

Also $\log n \leqq \bar{x}_5 \leqq 1.44 \log n$ since an AVL-tree with n leaves has height between $\log n$ and $1.44 \log n$.

Substituting into $\bar{x}_1 = \overline{\text{Val}}\,(n) + n + \bar{x}_{3,4} - \bar{x}_5$ and using Theorem 4 we get

$$1.47n - o(n) \leqq \bar{x}_1 \leqq 2.26n + o(n). \qquad \square$$

As mentioned above, there is considerable experimental evidence for the belief that $\bar{x}_1$ should be about $1.78n$ a figure which is near the average of the lower and upper bound given in Theorem 3.

5. On the distribution of the balance changes on the levels of the tree. In this section we study the distribution of the balance changes $0 \to \pm 1$ to the different levels of an AVL-tree. We need to refine the definition of Val (T) in 3 with respect to the heights.

DEFINITION. $\mathrm{val}_t(T)$ is the number of nodes with balance ± 1 and height t in the AVL-tree T.

Example. In the example in Fig. 6

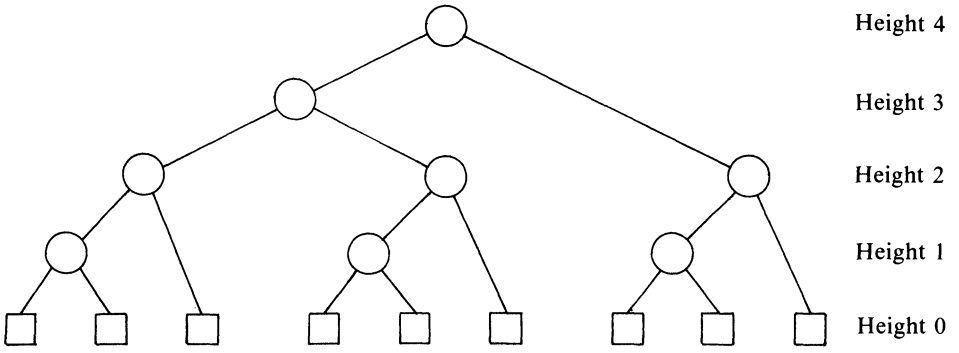


FIG. 6

we have $\mathrm{val}_1(T)=0$, $\mathrm{val}_2(T)=3$, $\mathrm{val}_3(T)=0$ and $\mathrm{val}_4(T)=1$. Note that nodes of height 1 are always balanced and hence $\mathrm{val}_1(T)=0$ for all trees T.

Next we need to refine the analysis of § 3. There we studied the effect of an insertion on $\mathrm{Val}(T)$; in this section we investigate this effect separately for each height t. As in § 3 we consider a sequence of n insertions into the initially empty tree T_0; let T_i be the tree after the ith insertion.

LEMMA 4. *Let h be the height of the critical node for the insertion into T_{i-1} if such a node exists and let h be the height of T_{i-1} otherwise. Then*

$$1)\qquad \mathrm{val}_t(T_i)=\mathrm{val}_t(T_{i-1})+\begin{cases}0 & \text{if } t=1 \text{ or } t>h,\\ 1 & \text{if } 2\leqq t\leqq h-1,\\ -1 & \text{if } t=h,\end{cases}$$

if a critical node exists;

$$2)\qquad \mathrm{val}_t(T_i)=\mathrm{val}_t(T_{i-1})+\begin{cases}0 & \text{if } t=1,\\ 1 & \text{if } 2\leqq t\leqq h+1\end{cases}$$

if no critical node exists.

Proof. By inspection of the figures in the proof of Lemma 1. □

It is interesting to observe that absorption, rotation or double rotation have exactly the same effect on $\mathrm{val}_t(T)$; this fact makes our analysis possible. Lemma 4 gives the effect of a single insertion on $\mathrm{val}_t(T)$, $t\geqq 1$. In Lemma 5 below we study the cumulative effect of all n insertions.

DEFINITION. Let C_t, $t\geqq 2$, be the number of insertions (among our sequence of n insertions) such that the critical node exists and has height $\geqq t$ and let I_t, $t\geqq 2$, be the number of insertions such that the critical node does not exist and the tree inserted into has height $\geqq t-2$.

We have the following Lemmas 5 and 6. In Lemma 6 we derive an upper bound on $\mathrm{val}_t(T_n)$ and in Lemma 5 we derive bounds on C_t and I_t in terms of $\mathrm{val}_i(T_n)$, $i\leqq t$. We then use Lemmas 5 and 6 to prove the main result of this section, Theorem 6.

LEMMA 5. a) $C_{t+1}=\sum_{i=2}^{t}(\mathrm{val}_i(T_n)-I_{i+1})/2^{t+1-i}+C_2/2^{t-1}$ *for* $t\geqq 2$,

b) $I_t=\mathrm{height}(T_n)+2-t$ *for* $2\leqq t\leqq \mathrm{height}(T_n)+1$,

c) $C_2+I_2=n$.

Proof. a) The proof of part a) is based on the following claim.

CLAIM. $\text{val}_t(T_n) = 2C_{t+1} + I_{t+1} - C_t$ *for* $t \geqq 2$.

Proof. Certainly $\text{val}_t(T_0) = 0$ for $t \geqq 2$. Next note that val_t is increased by one for every insertion such that either the critical node exists and has height $\geqq t+1$ or the critical node does not exist and the tree inserted into has height $\geqq t-1$ (there are exactly $C_{t+1} + I_{t+1}$ insertions of this type) and that val_t is decreased by one for every insertion whose critical node exists and has height t (there are exactly $C_t - C_{t+1}$ insertions of this type).

Thus

$$\text{val}_t(T_n) = \text{val}_t(T_0) + (C_{t+1} + I_{t+1}) - (C_t - C_{t+1}) \quad \text{for } t \geqq 2. \qquad \square$$

The claim above can be rewritten as

$$C_{t+1} = (\text{val}_t(T_n) - I_{t+1})/2 + C_t/2.$$

Part a) is now easily shown by induction.

b) For l, $0 \leqq l \leqq \text{height}(T_n) - 1$, there is exactly one insertion which increases the height from l to $l+1$. Thus $I_t = \text{height}(T_n) + 2 - t$.

c) The critical node always has height at least 2 and the tree inserted into has height at least 0. $\square$

LEMMA 6. $\text{val}_t(T_n) \leqq n/(1.618)^t$ *for* $t \geqq 1$.

Proof. We use the following notations: $m_1(t)$ is the number of unbalanced nodes of height t in T_n, and $m_2(t)$ is the number of balanced nodes of height t in T_n. Then $m_1(1) = 0$ and $m_1(t) + m_2(t)$ is the number of nodes of height t, $t \geqq 1$. Also $\text{val}_t(T_n) = m_1(t)$.

CLAIM 1. $2m_2(t) + m_1(t+1) + m_1(t) = m_1(t-1) + m_2(t-1)$ *for* $2 \leqq t \leqq \text{height}(T_n)$.

Proof. The right-hand side is the number of nodes of height $t-1$, the left-hand side counts the number of edges terminating at height $t-1$. There are two such edges for every balanced node of height t and one such edge for every unbalanced nodes of height t or $t+1$. $\square$

If we define $m_1(0) = 0$, $m_2(0) = n$ then the claim above is also true for $t = 1$.

Claim 1 yields

$$m_1(t+1) + Cm_1(t) + Cm_2(t) \leqq \frac{1}{C}[m_1(t) + Cm_1(t-1) + Cm_2(t-1)]$$

for the constant $C \simeq 1.618 < 2$ such that $C - 1/C = 1$. Hence

$$Cm_1(t) \leqq \frac{1}{C^t}[m_1(1) + Cm_1(0) + Cm_2(0)] = \frac{n}{C^{t-1}}$$

and

$$\text{val}_t(T_n) = m_1(t) \leqq \frac{n}{C^t} = \frac{n}{(1.618)^t}. \qquad \square$$

THEOREM 6. *Consider a sequence of n arbitrary insertions into an initially empty* AVL-*tree. Then*

$$C_t \leqq 6.85 \frac{n}{(1.618)^t}$$

and

$$I_t \leqq 1.44 \log n + 2 - t \quad \text{for } t \geqq 2.$$

Proof. The bound on I_t is immediate from Lemma 5b and the fact that an AVL-tree with n leaves has height at most $1.44 \log n$. The bound on C_t follows from Lemma

5a, c and Lemma 6, namely from Lemma 5a we get

$$C_t = \sum_{i=2}^{t-1} (\mathrm{val}_i(T_n) - I_{t+1})/2^{t-i} + C_2/2^{t-2}$$

$$\leqq n/2^{t-2} + \sum_{i=2}^{t-1} \mathrm{val}_i(T_n)/2^{t-i} \quad \text{since } C_2 \leqq n \text{ by Lemma 5c and } I_{t+1} \geqq 0$$

$$\leqq n/2^{t-2} + \sum_{i=2}^{t-1} (n2^i)/2^t(1.618)^i \quad \text{because of Lemma 6}$$

$$\leqq 6.85n/(1.618)^t \quad \text{by straightforward estimation.} \qquad \square$$

Theorems 1 and 6 give us good information on the amortized length of the critical path:

1) The amortized length is bounded by 2.618.

2) The number of insertions which have a critical path of length exceeding t decreases exponentially in t.

6. Conclusions. We have shown that the total number of balance changes required to process a sequence of n arbitrary insertions into an initially empty AVL-tree is at most $2.618n$. Moreover, the number of insertions in such a sequence which have a critical path of length exceeding t decreases exponentially in t. Finally, for sequences of n random insertions the expected total number of balance changes lies between $1.47n$ and $2.26n$.

Recently, Tsakalidis [9] has shown that the total number of rebalancing operations (=balance and structural changes) in processing a sequence of n deletions from an AVL-tree with n leaves is bounded by $1.618n$. Experiments by Karlton et al. [5] suggest that the expected number of rebalancing operations is $1.126n$ for a sequence of n random deletions.

For mixed sequences of insertions and deletions the amortized rebalancing cost is *not* constant. The reader should have no difficulty in finding a sequence of n insertions and deletions which produces $O(n \log n)$ balance changes. However, for *random* sequences of insertions and deletions the expected number of rebalancing operations might be $O(n)$ as experiments (cf. [5]) suggest. We leave the proof of this experimental fact as a major challenge to the reader.

REFERENCES

[1] G. M. ADEL'SON-VEL'SKII AND E. M. LANDIS, *An algorithm for the organisation of information*, Dokl. Akad. Nauk SSSR, 146 (1962), pp. 263-266 (in Russian); English Translation in Soviet. Math., 3, pp. 1259-1262.

[2] M. R. BROWN, *A partial analysis of random height-balanced trees*, this Journal, 8 (1979), pp. 33-41.

[3] C. C. FOSTER, *Information storage and retrieval using* AVL-*trees*, ACM 20th National Conference, 1965, pp. 192-205.

[4] S. HUDDLESTON AND K. MEHLHORN, *A new data structure for representing sorted lists*, Acta Informatica, 17 (1982), pp. 157-184.

[5] P. L. KARLTON, S. H. FULLER, R. E. SCROGGS AND E. B. KAEHLER, *Performance of height-balanced trees*, Comm. ACM, 19 (1976), pp. 23-28.

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

[7] K. MEHLHORN, *A partial analysis of height-balanced trees*, Technical Report A 79/13, Univ. des Saarlandes, 1979.

[8] ———, *A partial analysis of height-balanced trees under random insertions and deletions*, this Journal, 11 (1982), pp. 748-760.

[9] A. K. TSAKALIDIS, *Rebalancing operations for deletions in* AVL-*trees*, RAIRO Inform. théorique, in press.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
003

THE SIGNATURE OF A PLANE CURVE*

JOSEPH O'ROURKE†

Abstract. The *signature* of a plane curve Γ associated with every point p of Γ the length of Γ to the left of or on the line tangent to Γ at p. The signature has properties that make it a useful tool for pattern recognition: it discards the location, orientation, and scale, and "slant" in special cases, but preserves symmetries. Its integral is a measure of convexity. This paper explores the theoretical properties of this concept. It is shown that in the special case of closed rectilinear curves, the signature retains enough information to permit exact reconstruction of the curve. Computing the signature and reconstructing curves from their signatures are interesting computational problems; time complexity bounds on these problems are presented. Several challenging open questions are posed.

Key words. computational geometry, convexity, handwriting analysis, λ-matrix, multidimensional sorting, pattern recognition, plane curves, polygons, rectilinear curves, shape, signature

1. Introduction. Goodman and Pollack have defined a process called "multidimensional sorting" that captures the interrelationships among the points in a finite set in multidimensional Euclidean space [GP]. The sort is encoded in a "λ-matrix," which in two dimensions records the number of points to the left of each of the $O(n^2)$ directed lines determined by pairs of points in an n point set. Their idea suggests the following specialization to directed polygonal paths: associate with each edge the number of vertices of the path that are to the left of the line including the edge. This association requires only $O(n)$ space, which makes it a viable candidate for practical applications. However, some of the useful information encoded in the λ-matrix is of course lost when only $O(n)$ of its entries are retained. This paper is the result of attempting to discover how much information is lost and how much retained. Rather than studying the λ-matrix, however, the *signature* is investigated: this is a specialization of the λ-matrix in sense above, but a generalization in that it is concerned with continuous curves. The signature has interesting properties, raises interesting questions in both geometry and algorithms, and is a useful tool for pattern recognition applications [OW]. A precise definition follows.

Let Γ be a continuous, unit-length, directed curve in the Euclidean plane; the path formed by Γ may be open or closed, and it may cross itself. Consider Γ to be parametrized by its arc length s, so that it is a mapping from $[0, 1]$ to points in the plane. Let $T(s) = d\Gamma(s)/ds$ be a tangent vector to the curve at s when it exists. Then the *signature* Σ_Γ of Γ is a function that associates with each point of Γ (identified by its arc length parameter s) the length of Γ that is on or to the left of the line determined by $T(s)$. Thus $\Sigma_\Gamma(s)$ is a function mapping $[0, 1]$ into $[0, 1]$; often the subscript Γ will be dropped when clear from the context. The signature is undefined at points of Γ that have no tangent. Attention will be restricted in this paper to curves that have only a finite number of such "kinks."

Figure 1 shows a simple polygon and its signature. Along the interior of each edge, Σ is constant because the tangent is unchanging; thus the signature is a step function for polygonal curves. It is undefined at every vertex, but vertical lines are drawn connecting the steps in the figure for visual continuity. Note that the signature

* Received by the editors May 25, 1983, and in revised form January 8, 1984. This research was supported in part by the National Science Foundation under grants MCS 81-04780 and MCS 83-51468.

† Department of Electrical Engineering and Computer Science, Johns Hopkins University, Baltimore Maryland 21218.

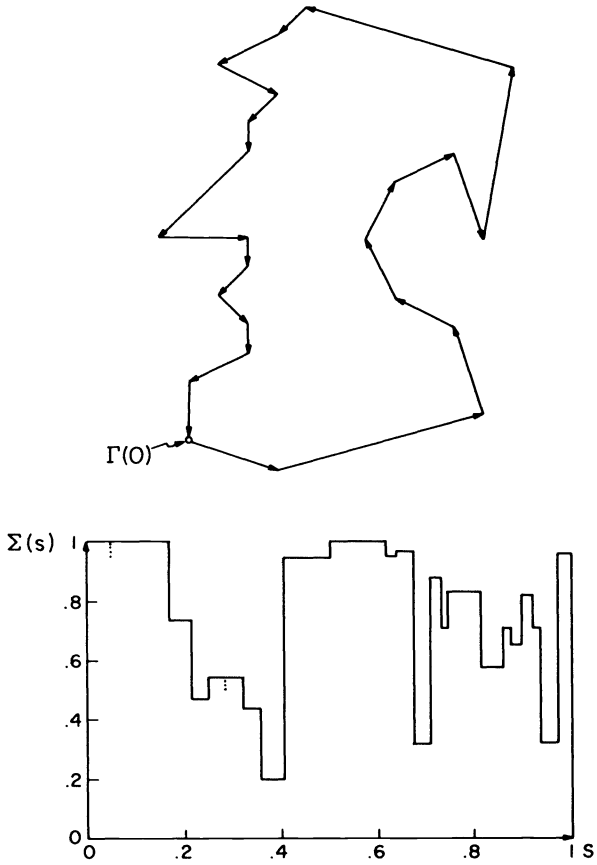


FIG. 1. *A simple polygon and its signature.*

is 1 for every edge on the convex hull of the figure, since the curve is of unit length and is directed counterclockwise.

Figures 2a and 2b show a more complex polygonal path and its signature. Again it is a step function, but the large number of edges in the path make it approach a continuous curve. It should be apparent that when a continuous curve is uniformly and densely digitized, the length of a section of the curve is approximately proportional to the number of vertices in that section. This intuition is supported by Fig. 2c, which shows the *discrete signature* S_Γ for the curve Γ in Fig. 2a: S_Γ is defined only for polygonal paths, and maps the arc length $s \in [0, 1]$ of Γ to the number of vertices on or to the left of the tangent at $\Gamma(s)$.

In general the signature of a curve does not uniquely identify the curve. For example, all smooth convex figures (of unit length) have the same signature: a constant at 1. But there is an important special class of curves for which the signature is a unique identifier: closed rectilinear curves (with certain degenerate self-intersections excluded). A proof of this fact is contained in § 3 after basic properties of the signature are developed in § 2. Computational issues are discussed in § 4, and several open problems are posed in the final section.

2. Basic properties. This section will informally present the basic properties of signatures, mostly without proof.

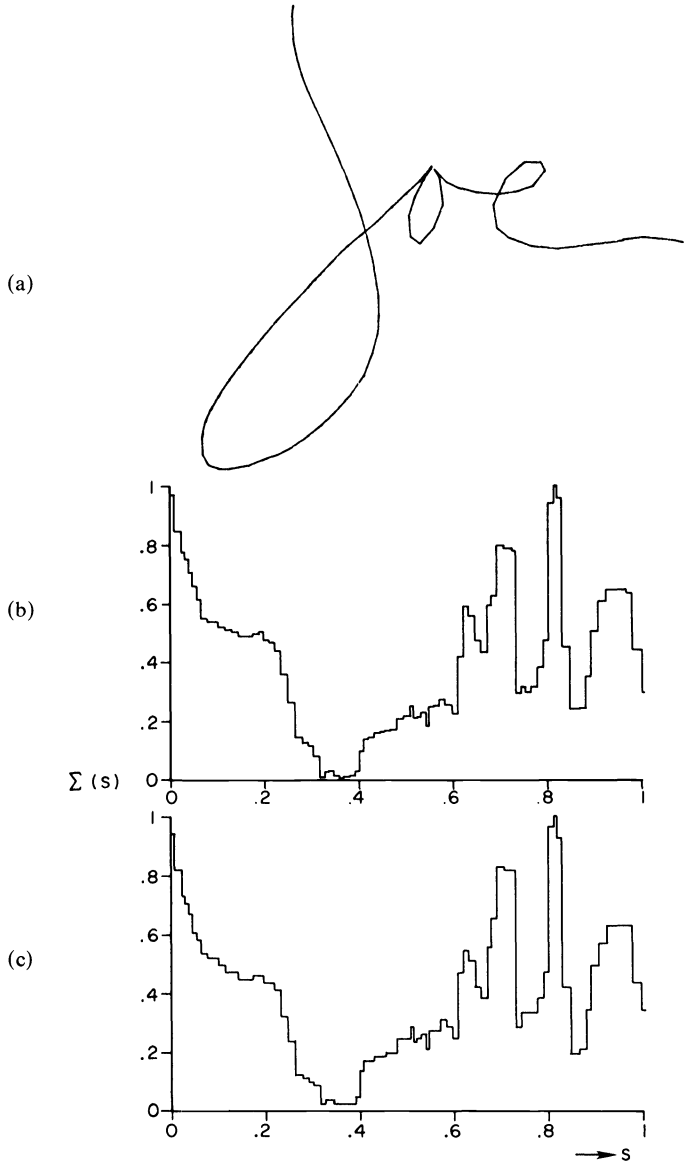


FIG. 2. *A polygonal path of* 89 *edges* (a), *and its continuous* (b) *and discrete* (c) *signatures.*

The signature of a smooth curve without flat sections (nonzero length portions of curvature zero) is a continuous function, since the tangent will not meet or disengage from a finite section upon infinitesimal turning. Also, the signature responds continuously to a deformation of such curves. The signature is obviously invariant with respect to translation and rotation of Γ, and to scale changes if the curve is normalized to unit length. The signature preserves symmetries in the sense that if Γ is isomorphic to its mirror image, then Σ_Γ is also equal to its mirror image.[1] These properties make the signature attractive as an encoding of the "shape" of a curve, and serve as the basis for its applications to pattern recognition.

[1] Σ_Γ symmetric does not imply that Γ is symmetric, which suggests a notion of "pseudosymmetry."

The signature has especially nice properties for *rectilinear curves*: those composed of a finite number of straight segments that are aligned with the axes of an orthogonal coordinate system. The signature of a rectilinear curve is invariant under "slanting transformations": those that rotate the orthogonal coordinate axes without changing their scale, changing to oblique parallel coordinates. See Fig. 3a for an example. This invariance does not hold for arbitrary curves, as Fig. 3b demonstrates, but it does suggest that it will be tolerant of slant when applied to handwriting samples (and hence the name *signature*). This tolerance has been verified in [OW], where several slanted and unslanted handwriting samples were compared. The discrete signature of arbitrary curves, on the other hand, is invariant under arbitrary affine transformations.

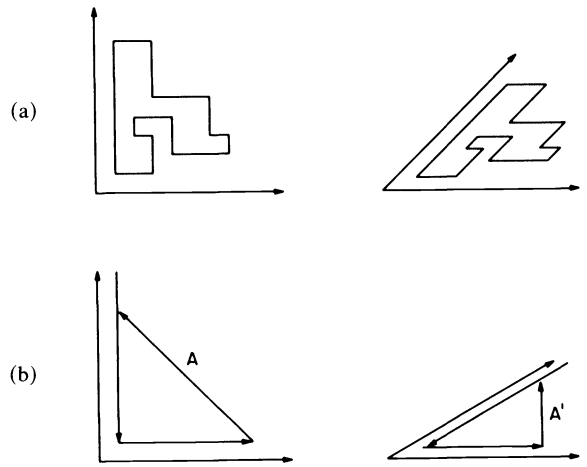


FIG. 3. *Slanting a rectilinear curve leaves the signature invariant* (a), *but this is not always true for nonrectilinear curves*: *in* (b) *edge A is* 38% *of the total curve length, but A', its slanted counterpart, is only* 18% *of the total. A slant of* ϑ *maps x and y into* $x' = x + y \cos \vartheta$ *and* $y' = y \sin \vartheta$.

The integral of the signature is a measure of convexity in the sense that, among simple closed curves, it is 1 if and only if the curve is convex and traversed counterclockwise, and is smaller for highly nonconvex curves. It appears difficult, however, to construct curves that have very small integrals. This is easily made quantitiative in the case of highly constrained curves such as rectilinear curves, as the following theorem demonstrates.

THEOREM 1. *Define* I_Γ *by*

$$I_\Gamma = \int_{s=0}^{s=1} \Sigma_\Gamma(s)\, ds. \tag{1}$$

If Γ *is a rectilinear simple polygon directed counterclockwise, then* $\frac{1}{2} < I_\Gamma \leqq 1$, *and if directed clockwise, then* $0 \leqq I_\Gamma < \frac{1}{2}$.

Proof. Let s_1 be the arc length parameter for an arbitrary noncorner point on Γ, and let s_2 be the arc length for the closest point of Γ on the ray orthogonal to the unique edge on which $\Gamma(s_1)$ lies, directed towards the interior of the polygon. See Fig. 4. It should be clear that, if s_2 is also a noncorner point, then (a) $\Gamma(s_1)$ and $\Gamma(s_2)$ lie on edges of the same orientation (horizontal or vertical), (b) the edges have opposite signs, and (c) the signs are such that the left sides of the edges face one another. It follows that $\Sigma(s_1) + \Sigma(s_2) > 1$ since together the two tangents have the entire plane to their left, and in addition this sum counts the portion of the curve between them twice.

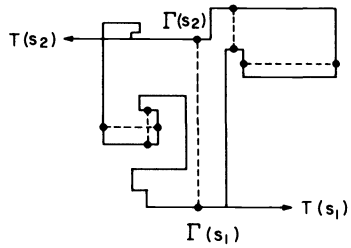


FIG. 4. *Noncorner points visible to one another along vertical or horizontal lines are called "antipodal."*

If we ignore the finitely-many corner points, the entire length of Γ can be partitioned into such point pairs, which we will call antipodal pairs. Since there is a one-to-one correspondence between the elements of an antipodal pair, the integral of the signature may be computed as follows, where we let $\pi(s_1) = s_2$ be a function relating arc lengths of pair members:

$$\int_{s=0}^{s=1} \Sigma(s)\, ds = \tfrac{1}{2} \int_{s=0}^{s=1} \Sigma(s) + \Sigma(\pi(s))\, ds.$$

The inequality derived previously then implies that the integral is $>\frac{1}{2}$.

The argument for the clockwise case proceeds in the same manner, except that the relevant inequality is $\Sigma(s_1)+\Sigma(s_2)<1$. □

This theorem can be extended to curves drawn on hexagonal grids, or in fact on any grid sufficiently regular to permit partitioning into antipodal pairs.

Several simple closed curves have been found whose integrals are $\frac{1}{4}+\varepsilon$ [GU]. Figure 5 shows an example due to Ungar. It is an open problem to prove or disprove that $\frac{1}{4}$ is a lower bound.

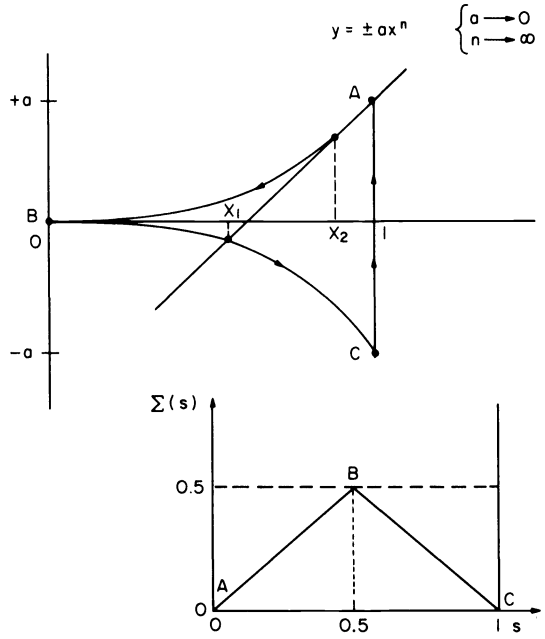


FIG. 5. *Ungar's "needle", shown here with* $n = 3$. *As* $a \to 0$, *the CA contribution becomes negligible. As* $n \to \infty$, *the signature value at A approaches* 0, *and at B it approaches* 1/2. *Also the difference* $x_2 - x_1$ *approaches* 0, *so the signature changes linearly from A to B as illustrated.*

3. Uniqueness of signature for rectilinear curves. In order to investigate the amount of "shape" information that is encoded in the signature, this section studies the special case of rectilinear curves in some detail. It is shown, somewhat surprisingly, that both the signature and the discrete signature for rectilinear curves retain enough information to permit exact reconstruction of the curve from the signature. How to actually compute the reconstruction will be discussed in the succeeding section. First some additional notation is introduced.

The signature of a rectilinear curve is punctuated by undefined values associated with each corner, which partition the signature in correspondence to the individual straight edges of the curve. Between adjacent undefined points, Σ is a constant. Label the edges of the curve $e_1, e_2, \cdots, e_n$, and write $\Sigma(e_i)$ for the constant value of Σ for s varying over edge e_i of the curve. The length of edge e_i, call it $\lambda(e_i)$, is easily derived from the signature (as long as the locations of the undefined points are known), as can be seen in the example shown in Fig. 6. The signature of a rectilinear curve will be represented by its sequence of constant values and lengths, $(\Sigma(e_i), \lambda(e_i))$, $0 \leqq i \leqq n$.

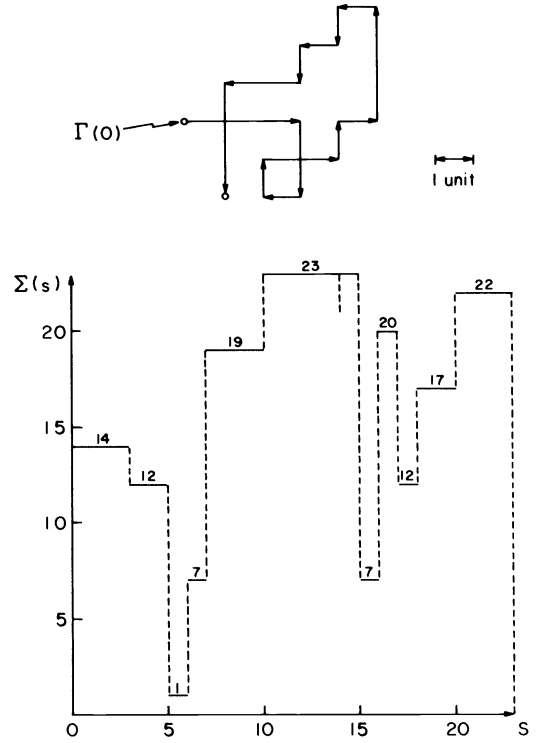


FIG. 6. *A rectilinear curve and its signature. In this example, the curve is not normalized to unit length.*

One additional piece of information permits the curve to be reconstructed from the signature. Define the sign of an edge $\sigma(e_i)$ to be $+1$ if e_i aims in either the positive X or Y direction, and -1 if it aims in the negative X or Y direction (consider an edge directed towards increasing values of s, the arc length parameter). A rectilinear curve can easily be reconstructed if the sign of its edges are known; in fact $\Sigma(e_i)$ is not needed. Arbitrarily set the first edge to be horizontal and draw from $(0, 0)$ to $(\sigma(e_1) \cdot \lambda(e_1), 0)$, then to $(\sigma(e_1) \cdot \lambda(e_1), \sigma(e_2) \cdot \lambda(e_2))$, then to $(\sigma(e_1) \cdot \lambda(e_1) + \sigma(e_3) \cdot \lambda(e_3), \sigma(e_2) \cdot \lambda(e_2))$, and so on. The curve is reconstructed up to a 90° rotation, dependent upon the initial arbitrary choice.

It is not immediately clear how to determine $\sigma(e_i)$ from the signature. The main theoretical result of this paper shows that of the 2^n possible choices for the sign of the n edges, only 2 are compatible with the signature values for closed rectilinear curves, and these two are inversions of one another.

Assign to each edge e of a rectilinear curve Γ an *orientation* of either *horizontal* or *vertical*. Denote by $\bar{\Gamma}$ the curve composed of the same edges as Γ, in the same sequential order, each with the same orientation, but with the sign of each edge e flipped to $-\sigma(e)$.

LEMMA 1. *For any rectilinear curve* Γ, *the signature of* $\bar{\Gamma}$ *is identical to the signature of* Γ: $\Sigma_{\bar{\Gamma}} = \Sigma_{\Gamma}$.

Proof. It is easy to see that $\bar{\Gamma}$ is just an upside-down version of Γ. Thus, even though each edge has flipped in direction, the amount of the curve to the left or on each edge remains the same. □

The main theorem of this section may now be stated.

THEOREM 2. *Let* Γ *be a rectilinear curve that is*

(1) *closed*;

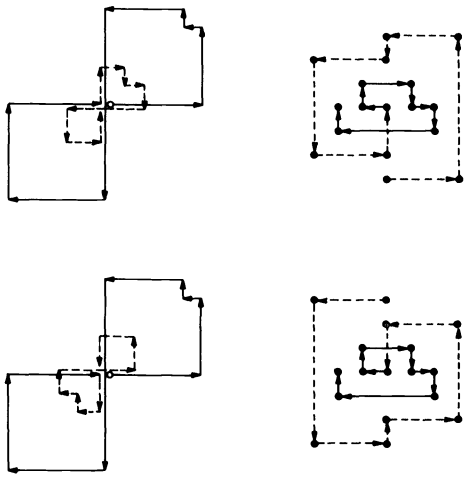
(2) *nondegenerate in the sense that*

(a) *no two distinct vertices coincide, and*

(b) *no two distinct edges collinearly overlap.*

Then both the signature and the discrete signature of Γ *are unique within this class of curves: there is no other closed, nondegenerate, rectilinear curve* Γ', *different from both* Γ *and* $\bar{\Gamma}$, *such that* $\Sigma_{\Gamma'} = \Sigma_{\Gamma}$ or $S_{\Gamma'} = S_{\Gamma}$.

Before commencing the proof, some discussion of conditions (1) and (2) of the theorem statement is in order. That the curve be closed is a necessary condition in the case of discrete signatures, as there exist two nonisomorphic open rectilinear curves with the same discrete signature: see Fig. 7b. I have been unable to construct two open curves with the same continuous signature, but the proof to be presented only applies to closed curves. The nondegeneracy conditions are at least jointly necessary, for both



(a) (b)

FIG. 7. (a) *Two nonisomorphic closed rectilinear curves that have the same continuous signature. In these curves, the parallel and nearly collinear edges are meant to be truly collinear, and are separated in the drawing so that distinct edges can be distinguished.* (b) *Two nonisomorphic open rectilinear curves that have the same discrete signature.*

the signature and the discrete signature, as the example in Fig. 7a demonstrates. Whether (2a) and (2b) are individually necessary remains unsettled by the proof in this paper.

The proof will be phrased in terms of the continuous signature; remarks on the changes necessary to prove the theorem for discrete signatures will be made at the end of this section. The proof is by contradiction. Assume that such a Γ' does exist. It must have the same number of edges as Γ, and the edges must have the same length and occur in the same order. We can rotate Γ' by 90° if necessary to ensure that corresponding edges in Γ and Γ' have the same orientation. All that can differ between Γ and Γ' is the sign of the edges.

After some preliminary definitions, a sketch of the proof will be presented followed by a series of 9 lemmas necessary for the proof. All lemmas will assume the existence of Γ' as described above.

If two corresponding edges e and e' in Γ and Γ' have the same sign, $\sigma(e) = \sigma(e')$, then they will be called *fixed edges*; if $\sigma(e) = -\sigma(e')$, then they will be called *flipping edges.*[2] Any contiguous sequence of fixed (flipping) edges will be called a *fixed (flipping) chain.* For example, the edges drawn solid in Fig. 7 are part of a fixed chain of Γ, and the dashed edges form flipping chains. The vector from the tail of the first edge to the head of the last edge in a chain will be called the *displacement* $\Delta = (\Delta x, \Delta y)$ of the chain.

A sketch of the proof is as follows. The signature value associated with an edge indicates its distance from a boundary measured in terms of the amount of the curve between it and the boundary. The signature values associated with two edges of the same orientation are therefore related in accordance with which is closer to a border, and determine their sorted order. Two parallel edges in Γ that do not flip in the transition to Γ' must maintain their original sorted order. It is this basic fact that is used to constrain the possible Γ' candidates. Avoiding an interchange of fixed edges forces each chain to have a longer displacement than its neighbor. Since the curve is closed, this eventually forces a chain to be longer than itself, a contradiction. This contradiction is only avoided when every chain has the exact same displacement. This case can be proven impossible by showing that the signature values for two corresponding edges in Γ and Γ' must be different.

LEMMA 2. *The displacement is not equal to* $(0, 0)$ *for any chain of* Γ *that is a proper subset of* Γ.

Proof. If $\Delta = (0, 0)$, then two vertices coincide. These vertices are distinct because the chain is not all of Γ. This contradicts the degeneracy condition (2a) of the theorem. □

LEMMA 3. *If C and C' are corresponding flipping chains in Γ and Γ' with displacements Δ and Δ', then $\Delta' = -\Delta$.*

Proof. Negating each horizontal and vertical displacement in a chain negates the chain's total displacement. See Fig. 8. □

The next lemma shows that fixed edges must maintain their sorted order, which greatly constrains the structure of Γ'.

LEMMA 4. *Let e_1 and e_2 be fixed edges in Γ with the same orientation; assume wlog that this orientation is vertical. Classify their relationship to one another as left if e_2 is strictly to the left of e_1, equal if they are collinear, and right otherwise. Then the relationship between the corresponding edges e_1' and e_2' in Γ' cannot be reversed: if the relationship is left in Γ, then it cannot be right in Γ', and if right in Γ, it cannot be left in Γ'.*

[2] Note that which edges are fixed and which flipping depends on the particular Γ' candidate under consideration. If attention is switched to $\bar{\Gamma}'$, for example, then the sense of fixed and flipping is interchanged.

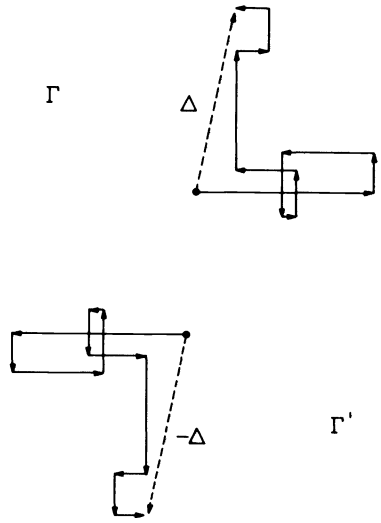


FIG. 8. *Flipping a chain turns it upside-down, negating its displacement.*

Proof. Let e_1 be to the right of e_2, and let $L_i = \Sigma(e_i)$ and $R_i = 1 - L_i$ for $i = 1, 2$. If e_1 and e_2 have the same sign, the situation is as illustrated in Fig. 9a: $L_2 < L_1$ in Γ, but if the edges change their sorted order in Γ', then $L_1 < L_2$, a contradiction. If e_1 and e_2 have opposite signs, as shown in Fig. 9b, then $R_2 < L_1$ in Γ, but order interchange in Γ' would lead to $L_1 < R_2$, again a contradiction. □

Although two fixed edges cannot interchange position, they can move from collinearity to noncollinearity. The next lemma delimits the possibilities.

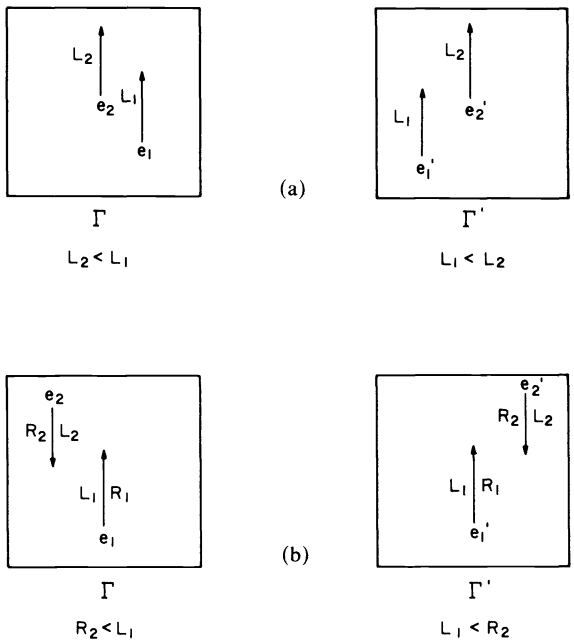


FIG. 9. *Fixed edges cannot reverse their order in the two curves.*

LEMMA 5. *Let e_1 and e_2 be as in the previous lemma. Then the relationship between them can change from left to equal or vice versa only if $\sigma(e_1) = +1$ and $\sigma(e_2) = -1$, and from right to equal or vice versa only if $\sigma(e_1) = -1$ and $\sigma(e_2) = +1$.*

Proof. Fig. 10a illustrates the impossibility of the *left/equal* transition when $\sigma(e_1) = -1$ using the same notation and conventions as in the previous lemma: $L_2 < R_1$ in Γ but $R_1 < L_2$ in Γ'. Fig. 10b shows that the transition is indeed possible when $\sigma(e_1) = +1$, as long as $R_2 < L_2$ and $L_2 > R_1$. The *right/equal* transition results can be obtained by inverting Fig. 10. □

LEMMA 6. *There must be at least 4 distinct chains.*

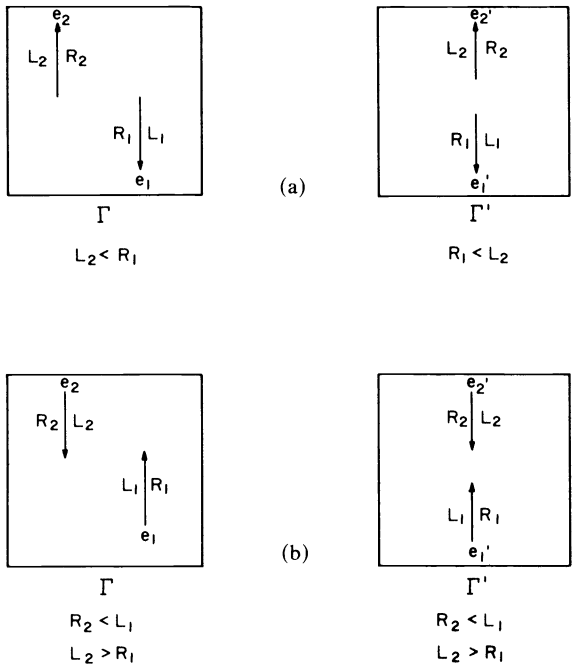


FIG. 10. *Fixed edges can only move to collinearity when they point towards one another.*

Proof. There can only be one chain if $\Gamma' = \Gamma$ or $\Gamma' = \bar{\Gamma}$, and these cases are explicitly excluded in the statement of the theorem. Assume there are precisely two chains, one fixed and one flipping. Call these chains C_1 and C_2 respectively in Γ. Since the curve is closed by clause (1) of the theorem, the head of one chain meshes with the tail of the other; so it must be the case that $\Delta_2 = -\Delta_1$ in Γ and $\Delta_2' = -\Delta_1'$ in Γ'. But, by Lemma 3, $\Delta_2' = -\Delta_2$ because C_2 is flipping, whereas $\Delta_1' = \Delta_1$ since C_1 is fixed. These equations are only satisfied when $\Delta_1 = \Delta_2 = 0$, which contradicts Lemma 2. Finally, since the chains alternate between fixed and flipping by definition, and since Γ is closed, there must be an even number of chains if there are more than one. Thus there cannot be precisely 3 chains. □

The next three lemmas establish some simple facts about chains.

LEMMA 7. *Each chain must consist of at least two edges.*

Proof. Assume that there exists a chain that consists of a single edge e. Assume wlog that e is horizontal. Also assume that the chain is flipping. This can be assumed wlog because any chain that is fixed in Γ and Γ' is flipping in Γ and $\bar{\Gamma}'$ and vice versa, and Lemma 1 shows that $\bar{\Gamma}'$ is just as good a candidate for contradiction as Γ'. The

edge e is bounded by two fixed vertical edges. By Lemma 3, the displacement represented by e, which is nonzero by Lemma 2, is negated from Γ to Γ'. This interchanges the relative positions of the bounding fixed edges, contradicting Lemma 4. □

LEMMA 8. *Each chain must begin and end with edges of different orientation.*

Proof. Assume wlog that a chain C both begins and ends with horizontal edges, and that it is a flipping chain.

Case 1. C has a nonzero horizontal displacement, $|\Delta x| > 0$.

This case is effectively equivalent to the conditions of Lemma 7: the relative positions of the two bounding fixed vertical edges are interchanged, contradicting Lemma 4.

Case 2. C has a pure vertical displacement, $\Delta x = 0$.

Since the two bounding fixed chains each must contain at least two edges by Lemma 7, they each must include a fixed horizontal edge. Label the vertical coordinates of the endpoints of the two bounding vertical edges as in Fig. 11, assuming temporarily that the displacement of C in Γ is >0. By this assumption, $y_2 < y_3$, and to avoid the degenerate overlap forbidden by clause (2b) of the theorem, we must also have $y_1 < y_4$. By Lemma 3, C's displacement is negated in Γ', implying that $y_3' < y_2'$. Thus at least one point of e_2' is below e_1'. But again because the edges cannot overlap, the entirety of edge e_2' must lie below e_1', which implies that $y_4' < y_1'$. But this necessarily interchanges the relative positions of the fixed horizontal edges e_0 and e_3, contradicting Lemma 4. Starting with the assumption that C's displacement is negative is equivalent to interchanging the roles of Γ and Γ' in the above argument, and leads to the same contradiction. □

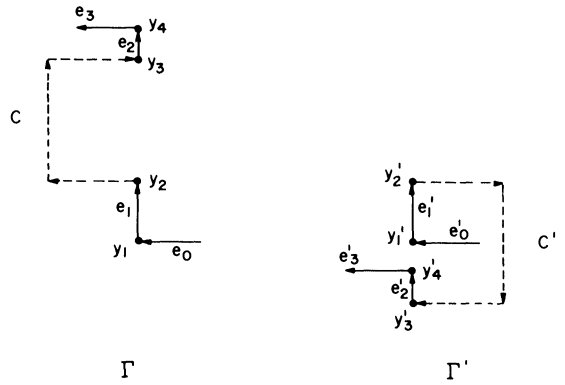


FIG. 11. *A chain that has zero horizontal displacement, and both begin and end with horizontal edges.*

The next lemma establishes the crucial constraint on the displacement of adjacent chains.

LEMMA 9. *Suppose there exist three consecutive chains C_0, C_1, C_2, with C_0 and C_2 fixed and C_1 flipping, and such that C_0 ends with a vertical edge e_0 (see Fig. 12). Then the horizontal displacement of C_2 is at least as large as that of C_1: $|\Delta x_2| \geqq |\Delta x_1|$.*

Proof. If $\Delta x_1 = 0$, the theorem is trivially true. So assume that $|\Delta x_1| > 0$. Let Q be the vertex adjacent to both C_1 and C_2, as shown in Fig. 12. Then every point of C_2 that falls within the vertical strip of width $2|\Delta x_1|$ centered on Q (call this the Q-strip) is to the left of e_0 in Γ (if $\Delta x_1 < 0$ as illustrated), but to the right of e_0 in Γ'. If $\Delta x_1 > 0$, the same holds true in reverse. Now since C_2 begins with a horizontal edge e_1, it must end with a vertical edge e_2, by Lemma 8. But this vertical edge cannot lie within the interior of the Q-strip in Γ, else it would change its relative position with e_0, contradict-

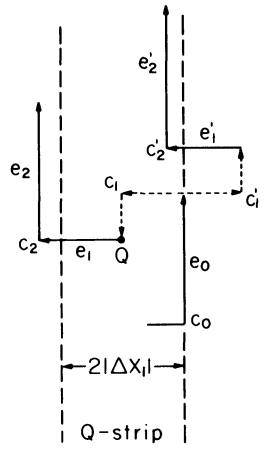


FIG. 12. *C_0 is shown fixed with the same absolute position in Γ and Γ'. C_1 and C_2 are shown in both their Γ and Γ' positions.*

ing Lemma 4. Therefore the head of C_2 must lie on the boundary or outside of the Q-strip; since its tail is at Q, $|\Delta x_2| \geqq |\Delta x_1|$. □

LEMMA 10. *The preconditions of Lemma 9 must hold for at least one group of three chains C_0, C_1, C_2.*

Proof. Let C_1 be any flipping chain. If C_1 begins with a vertical edge, rotate Γ and Γ' by 90° so that C_1 begins with a horizontal edge. By Lemma 6, there must be at least 4 chains, and therefore the two fixed chains C_0 and C_2 adjacent to C_1 must be distinct. Since C_1 begins with a horizontal edge, C_0 must end with a vertical edge. Thus the conditions of Lemma 9 are established. □

Proof of Theorem 2. Let $C_0, C_1, C_2, \cdots, C_k$ be the distinct chains of Γ and Γ', with C_0, C_1, C_2 satisfying the preconditions of Lemma 9, as guaranteed by Lemma 10. Then Lemma 9 yields $|\Delta x_1| \leqq |\Delta x_2|$. Now observe that C_1, C_2, C_3 satisfy the conditions of Lemma 9 except for the reversal of *fixed* and *flipped*: C_1 ends with a vertical edge. If we now switch our attention to $\bar{\Gamma}'$, then every fixed chain becomes flipped and vice versa, and Lemma 9 holds. This establishes $|\Delta x_2| \leqq |\Delta x_3|$. Continuing in this fashion we obtain the series of inequalities

$$|\Delta x_1| \leqq |\Delta x_2| \leqq |\Delta x_3| \leqq \cdots \leqq |\Delta x_k| \leqq |\Delta x_0| \leqq |\Delta x_1|. \tag{2}$$

Note the final inequality, $|\Delta x_0| \leqq |\Delta x_1|$, arises from clause (a) of the theorem statement, requiring Γ to be closed and C_k, C_0, C_1 to be consecutive.

The argument of Lemma 9 can be turned on its side, so to speak: if C_0, C_1, C_2 are consecutive with C_1 flipping and C_2 beginning with a horizontal edge (e_1 in Fig. 12), then $|\Delta y_0| \geqq |\Delta y_1|$. Note that the same three chains guaranteed by Lemma 10 can serve to start this argument. Proceeding as above, we can conclude

$$|\Delta y_1| \geqq |\Delta y_2| \geqq |\Delta y_3| \geqq \cdots \geqq |\Delta y_k| \geqq |\Delta y_0| \geqq |\Delta y_1|. \tag{3}$$

Now it is clear that the above inequalities lead to a contradiction if any term is strictly greater or less than its neighbor. Therefore it must be the case that

$$|\Delta x_i| = |\Delta x_j| = |\Delta x|, \qquad |\Delta y_i| = |\Delta y_j| = |\Delta y| \tag{4}$$

for all i and j. We will now show that these rather strict restrictions on the chains lead to a contradiction.

Assume wlog that e_0, the end vertical edge of C_0, is directed upwards, $\sigma(e_0) = +1$. The above equations require that e_2, the final vertical edge of C_2, must lie on the boundary of the Q-strip. Lemma 5 requires that $\sigma(e_2) = -1$, and that only a left/equal (or vice versa) interchange between e_2 and e_0 occur. Thus, independent of whether $\Delta x_1 = +\Delta x$ or $-\Delta x$, e_2 must lie on the left boundary of the Q-strip, and so $\Delta x_2 = -\Delta x$. Now since $\sigma(e_2) = -1$, the same argument applied to C_2, C_3, C_4 yields $\Delta x_4 = +\Delta x$.

Returning to C_0, C_1, C_2, there are 8 distinct possibilities generated by $\pm\Delta x_1$, $\pm\Delta y_1$, $\pm\Delta y_2$ (recall that Δx_2 must be $-\Delta x$). If $\Delta_1 = -\Delta_2$ or $\Delta_1 = \Delta_2$, then the heads of e_0 and e_2 touch in either Γ or Γ' respectively. This eliminates the four possibilities

Δ_1	Δ_2
$(+\Delta x, +\Delta y)$	$(-\Delta x, -\Delta y)$
$(+\Delta x, -\Delta y)$	$(-\Delta x, +\Delta y)$
$(-\Delta x, +\Delta y)$	$(-\Delta x, +\Delta y)$
$(-\Delta x, -\Delta y)$	$(-\Delta x, -\Delta y)$

The two possibilities

Δ_1	Δ_2
$(+\Delta x, -\Delta y)$	$(-\Delta x, -\Delta y)$
$(-\Delta x, +\Delta y)$	$(-\Delta x, -\Delta y)$

are illustrated in Fig. 13. Both either lead to a degenerate overlap violating clause (2b) of the theorem, or an interchange of fixed edges, contradicting Lemma 4. It therefore must be that $\Delta y_2 = +\Delta y$, but both $\Delta_1 = (+\Delta x, +\Delta y)$ and $\Delta_1 = (-\Delta x, -\Delta y)$ are viable.

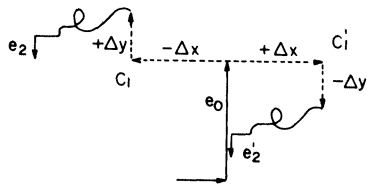


FIG. 13. *When* $\Delta_2 = (-\Delta x, -\Delta y)$, *then if* $\Delta_1 = (-\Delta x, +\Delta y)$, *either* e_2 *will be brought into degenerate overlap with* e_0, *or the horizontal edges preceding* e_0 *and* e_2 *will interchange. If* $\Delta_1 = (+\Delta x, -\Delta y)$, *then the same drawing obtains with the roles of* Γ *and* Γ' *reversed.*

Repeating the argument for C_2, C_3, C_4 again yields just two possibilities. These relationships are summarized in Table 1.[3]

TABLE 1

C_1 flipping	C_2 fixed	C_3 flipping	C_4 fixed
$(-\Delta x, -\Delta y)$	$(-\Delta x, +\Delta y)$	$(+\Delta x, +\Delta y)$	$(+\Delta x, -\Delta y)$
$(-\Delta x, -\Delta y)$	$(-\Delta x, +\Delta y)$	$(-\Delta x, -\Delta y)$	$(+\Delta x, -\Delta y)$
$(+\Delta x, +\Delta y)$	$(-\Delta x, +\Delta y)$	$(+\Delta x, +\Delta y)$	$(+\Delta x, -\Delta y)$
$(+\Delta x, +\Delta y)$	$(-\Delta x, +\Delta y)$	$(-\Delta x, -\Delta y)$	$(+\Delta x, -\Delta y)$

[3] The reason that there are two choices for the flipping chains and only one for the fixed chains is that we assumed that $\sigma(e_0) = +1$, removing one possibility by this convention.

If the x components of Δ_1 and Δ_3 are the same, as they are in the 2nd and 3rd rows of this table, then e_4, the last vertical edge of C_4, interchanges order with e_0 in the transition from Γ to Γ', violating Lemma 4. Thus only rows 1 and 4 of the table remain viable. Summing up the net displacement in either of these rows reveals that after 4 chains, the total displacement is zero. Thus, to avoid a degenerate self-intersection, there must be precisely 4 chains, and $C_4 \equiv C_0$. We will now show by explicit calculation that in these two highly constrained situations, the signature values on certain edges are not consistent in Γ and Γ'.

Rows 1 and 4 of the table are clearly just flipped versions of one another, and we can concentrate on the transition in one direction. Assume then that we have the arrangement shown in Fig. 14a for Γ. As usual, we assume wlog that e_0, the last edge of C_0, is vertical with $\sigma(e_0) = +1$, so that e_2, the last edge of C_2, is vertical with $\sigma(e_2) = -1$. Edge e_2 is to the left of e_0 in Γ, and they become collinear in Γ', a transition permitted by Lemma 5, as shown in Fig. 14b. The intuitive feeling that the signature values associated with e_0 and e_2 must be larger in Γ than in Γ' is borne out by the following calculation. Let $L_0 = \Sigma(e_0)$ and $L_2 = \Sigma(e_2)$ in Γ, and L_0' and L_2' similarly in Γ'. We will show that $L_0 + L_2 > L_0' + L_2'$.

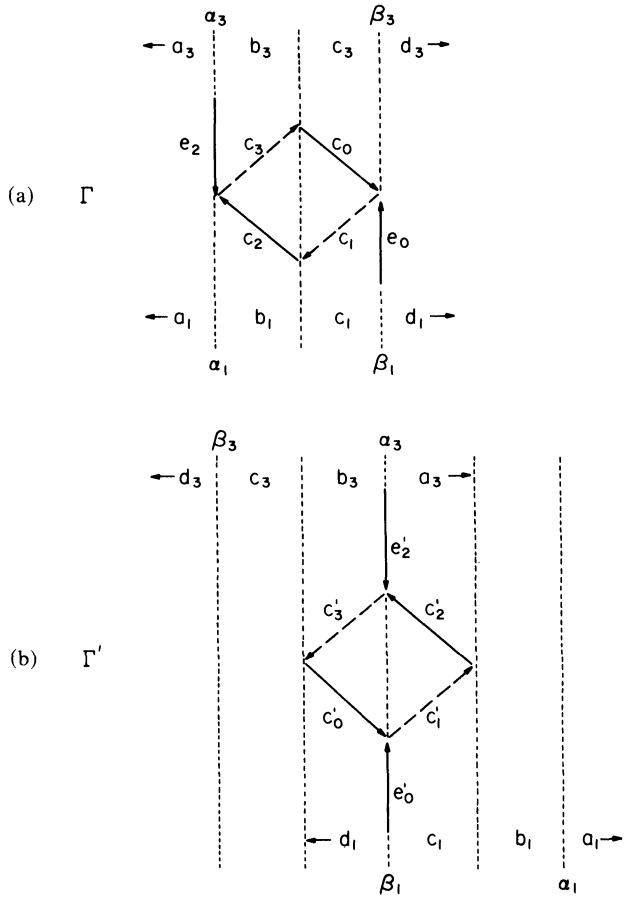


FIG. 14. *The restricted 4-chain curves cannot have the same signature, as can be proven by partitioning the contribution of each chain into vertical strips.*

At first we will ignore the contributions to the signature values made by the fixed chains, lumping them into terms A_0, A_2, A_0', A_2'. Partition the flipping chains C_1 and C_3 by vertical lines and label the amount of each chain within the strips a_1, b_1, c_1, d_1 for C_1 and a_3, b_3, c_3, d_3 for C_3 as illustrated (some of these terms may be zero). Distinguish the amount of these chains on the critical partition lines collinear with e_0 and e_2 as α_1, β_1 for C_1 and α_3, β_3 for C_3 as illustrated; the amount falling on the other partition lines can be assigned to the strip on either side. Then Fig. 14a shows that

$$\begin{aligned} L_0 &= a_1+b_1+c_1+a_3+b_3+c_3+\alpha_1+\beta_1+\alpha_3+\beta_3+A_0, \\ L_2 &= b_1+c_1+d_1+b_3+c_3+d_3+\alpha_1+\beta_1+\alpha_3+\beta_3+A_2, \end{aligned} \tag{5}$$

where A_0 is the total amount of the fixed chains to the left or on e_0, and A_2 is a similar term for e_2. Fig. 14b shows that

$$\begin{aligned} L_0' &= d_1+b_3+c_3+d_3+\beta_1+\alpha_3+\beta_3+A_0', \\ L_2' &= a_1+b_1+c_1+\alpha_3+\alpha_1+\beta_1+\alpha_3+A_2', \end{aligned} \tag{6}$$

where A_0' and A_2' are the contributions of the fixed chains.

Now since Γ and Γ' have the same signature, $L_0'=L_0$ and $L_2'=L_2$, and therefore $L_0+L_2-(L_0'+L_2')=0$. This yields

$$b_1+c_1+b_3+c_3+\alpha_1+\beta_3+(A_0+A_2)-(A_0'+A_2')=0. \tag{7}$$

Note that $A_0 \geqq A_0'$ and $A_2 \geqq A_2'$, since e_0 and e_2 move towards one another anchored to their fixed chains. As all other terms are $\geqq 0$, each term must be identically zero. But it is not possible for both b_1 and c_1 to be zero, since the endpoint of chain C_1 lies between the b_1 and c_1 strips, and some portion of C_1 must lie in at least one strip to reach its endpoint.

End proof of Theorem 2.

The preceding proof, with all the lemmas, also holds for the discrete signature. The only changes required are that the contradiction illustrated in Fig. 9b is $R_2 < L_1$ in Γ versus $L_1 \leqq R_2$ in Γ'; and $A_0 > A_0'$ in the proof above because at least one vertex of a fixed chain moves to the right of e_0, namely the first vertex of C_2.

4. Computational issues. Two challenging computational problems are posed by signatures: fast construction of the signature from a given polygonal curve and fast reconstruction of a rectilinear curve from its signature.

4.1. Computing the signature. The signature requires $\Theta(n)$ storage, and this provides a trivial lower bound on the time complexity of any algorithm that computes it. The obvious brute-force algorithm (for each edge, traverse the entire path and note how much length or how many vertices fall to one side) yields an $O(n^2)$ upper bound for both the continuous and discrete signature computation. An $\Omega(n \log n)$ lower bound for the computation of the unnormalized discrete signature can be obtained by the following sorting reduction due to Edelsbrunner [E]. Given real numbers $y_1, \cdots, y_n$, construct a "histogram" curve Γ with n vertical bars of heights $y_1, \cdots, y_n$ as illustrated in Fig. 15. It should be clear that for each horizontal bar top e, $S_\Gamma(e)$ will assume one of the values $\{2n+2, 2n+4, \cdots, 2n+2k, \cdots, 4n\}$. These values can be sorted in linear time with a radix sort [Kn]. Association of the sorted order with the original y_k's establishes that sorting is linear time reducible to the computation of the discrete signature. No such lower bound is evident for the continuous signature.

Computing the discrete signature is a special case of the much-studied "half-plane query counting" problem [EW], [F1], [F2], [W]. This problem requires the construction

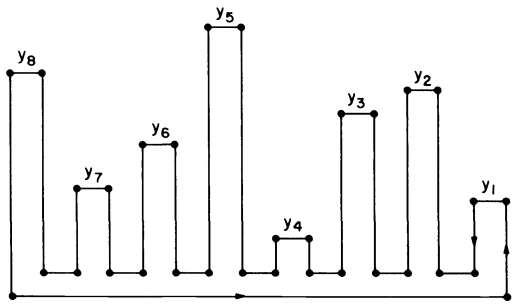


FIG. 15. *Construction reducing sorting to computation of the discrete signature.*

of a data structure to store n two-dimensional points such that queries of the form, "How many points are inside this (arbitrary) half-plane?", are swiftly answered. The discrete signature presents a special case in that the query half-planes are not arbitrary: there are precisely n relevant half-planes (determined by the edges of the polygonal path), and each will be used exactly once in a query. The following argument of Seidel [S] achieves $O(n^{3/2} \log n)$ time and $O(n)$ space bounds for the computation of the discrete signature. Partition the n vertices into $\sqrt{n}$ groups each containing $\sqrt{n}$ points. Dualize each group of points to a "ranked" arrangement of lines as in [EOW]; this requires $O(n)$ time and space per group. Now locate the dual point of each of the n polygon edges within each arrangement at a cost of $O(\sqrt{n} \log n)$ [K] per polygon edge. The number of points to the left of the edge is available from the ranked arrangement as the number of lines above or below the dual point [EOS]. Organizing the location queries to exhaust one arrangement before moving to another permits the space requirements to be kept to $O(n)$.

Both the continuous and discrete signatures are easily computed in $O(n \log n)$ time for rectilinear curves by a simple plane-sweep algorithm that will not be detailed here. The status of the general case is summarized in Table 2.

TABLE 2

computation	lower bound	upper bound
discrete signature	$\Omega(n \log n)$	$O(n^{3/2} \log n)$
continuous signature	$\Omega(n)$	$O(n^2)$

4.2. Reconstruction computation. Since Theorem 2 guarantees unique signatures for closed nondegenerate rectilinear curves, such curves can be reconstructed from their signatures by an exhaustive search of the 2^n possible reconstructions. This brute-force algorithm has been implemented. The algorithm arbitrarily fixes the orientation and direction of one edge, then proceeds to explore both possible directions for each succeeding edge. A search path is aborted as soon as a contradiction between signature values is detected. The contradictions are found by actually recording the path in an array, and examining the projections of the path's length on the X and Y axes. As each edge of the path is added, both projections must be checked for consistency, a calculation that could cost as much as $O(n)$, where n is the total number of edges in the curve. Since there are potentially 2^n different paths, the worst-case complexity of the algorithm is $O(n2^n)$.

The algorithm was tested on 35 randomly generated open rectilinear curves. Each curve consisted from 1 to 20 edges. In every case, the signature was found to be unique: only two curves were found to be consistent with the signature, one a completely flipped version of the other. Figure 16 shows the total number of edge placements tried by the algorithm as a function of n. In the worst case, this function could be $2^1+2^2+\cdots+2^n=2(2^n-1)$, but the graph shows that its growth in the average case is linear ($\leqq 5n$). Coupled with the $O(n)$ consistency check, the algorithm runs in $O(n^2)$. This rather remarkable performance may indicate the existence of an algorithm that does not use brute-force search.

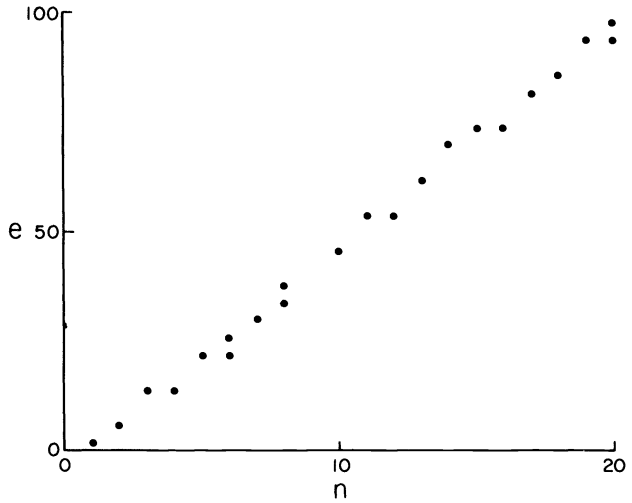


FIG. 16. *Performance of the brute-force searching algorithm on* 35 *random examples. The abscissa is the number of edges in the polygon, and the ordinate is the number of edge placements attempted during the search.*

5. Open questions. Several interesting questions remain to be answered:

(1) Which functions on [0, 1] are signatures of some curve? Similar characterization questions can be asked for polygonal curves, rectilinear curves, closed curves, etc. It is easily established by a counting argument that not every step function is the signature of some rectilinear curve, and it is possible to construct a step function that is not the signature of any connected curve, but otherwise the characterization problem remains open.

(2) Is 1/4 a lower bound on the signature integral of simple closed counterclockwise curves? This would give a partial characterization of such curves, as well as representing a surprisingly general property of plane curves.

(3) Can the signature (continuous or discrete) of a n-segment polygonal path be computed in $O(n \log n)$ time? What if the path is restricted to be a simple polygon?

(4) Can a rectilinear curve be reconstructed in polynomial time from its signature (continuous or discrete), or is this problem NP-complete?

It is possible to modify the definition of signature for simple closed curves to use the normal rather than the tangent, or to measure the area to the left rather than the length. It is also possible to extend both this and the original definition to three dimensions: the length signature can be defined for space curves using a plane orthogonal to the "osculating plane" [St], and a volume signature for closed surfaces can be defined using the tangent plane. These notions remain to be explored.

Acknowledgments. I thank Richard Washington for implementing the signature computation and reconstruction algorithms, and for producing several of the figures. I thank Herbert Edelsbrunner, Jacob Goodman, Rao Kosaraju, Richard Pollack, Raimund Seidel, Peter Ungar, and Emmerich Welzl for enlightening discussions. Finally I thank the two referees, whose remarkably detailed comments made a significant contribution to the paper.

REFERENCES

[E] H. EDELSBRUNNER, personal communication, 1983.

[EOS] H. EDELSBRUNNER, J. O'ROURKE AND R. SEIDEL, *Constructing arrangements of lines and hyperplanes with applications*, Proc. IEEE 24th Symposium on Foundations of Computer Science, 1983, pp. 83-91, this Journal, 15 (1986), to appear.

[EW] H. EDELSBRUNNER AND E. WELZL, *Halfplanar range search in linear space and $O(n^{0.695})$ query time*, Technische Universität Graz Bericht F 111, 1983.

[F1] M. L. FREDMAN, *Lower bounds on the complexity of some optimal data structures*, this Journal, 10 (1981), pp. 1-10.

[F2] ———, *A lower bound on the complexity of orthogonal range queries*, J. Assoc. Comput. Mach., 28 (1981), pp. 696-705.

[GP] J. E. GOODMAN AND R. POLLACK, *Multidimensional sorting*, this Journal, 12 (1983), pp. 484-507.

[GU] J. E. GOODMAN AND P. UNGAR, personal communication, 1982.

[K] D. G. KIRKPATRICK, *Optimal search in planar subdivisions*, this Journal, 12 (1983), pp. 28-35.

[Kn] D. E. KNUTH, *The Art of Computer Programming* III: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[OW] J. O'ROURKE AND R. WASHINGTON, *The signature of a curve: a new tool for pattern recognition*, Computational Geometry, G. Toussaint, ed., North-Holland, Amsterdam, 1985.

[S] R. SEIDEL, personal communication, 1983.

[St] J. J. STOKER, *Differential Geometry*, John Wiley, New York, 1969.

[W] D. E. WILLARD, *Polygon retrieval*, this Journal, 11 (1982), pp. 149-165.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
004

SELF-ADJUSTING HEAPS*

DANIEL DOMINIC SLEATOR† AND ROBERT ENDRE TARJAN†

Abstract. In this paper we explore two themes in data structure design: *amortized computational complexity* and *self-adjustment.* We are motivated by the following observations. In most applications of data structures, we wish to perform not just a single operation but a sequence of operations, possibly having correlated behavior. By averaging the running time per operation over a worst-case sequence of operations, we can sometimes obtain an overall time bound much smaller than the worst-case time per operation multiplied by the number of operations. We call this kind of averaging *amortization.*

Standard kinds of data structures, such as the many varieties of balanced trees, are specifically designed so that the worst-case time per operation is small. Such efficiency is achieved by imposing an explicit structural constraint that must be maintained during updates, at a cost of both running time and storage space. However, if amortized running time is the complexity measure of interest, we can guarantee efficiency without maintaining a structural constraint. Instead, during each access or update operation we adjust the data structure in a simple, uniform way. We call such a data structure *self-adjusting.*

In this paper we develop the *skew heap,* a self-adjusting form of heap related to the leftist heaps of Crane and Knuth. (What we mean by a heap has also been called a "priority queue" or a "mergeable heap".) Skew heaps use less space than leftist heaps and similar worst-case-efficient data structures and are competitive in running time, both in theory and in practice, with worst-case structures. They are also easier to implement. We derive an information-theoretic lower bound showing that skew heaps have minimum possible amortized running time, to within a constant factor, on any sequence of certain heap operations.

Key words. Self-organizing data structure, amortized complexity, heap, priority queue

1. Introduction.

Many kinds of data structures have been designed with the aim of making the worst-case running time per operation as small as possible. However, in typical applications of data structures, it is not a single operation that is performed but rather a sequence of operations, and the relevant complexity measure is not the time taken by one operation but the total time of a sequence. If we average the time per operation over a worst-case sequence, we may be able to obtain a time per operation much smaller than the worst-case time. We shall call this kind of averaging over time *amortization.* A classical example of amortized efficiency is the compressed tree data structure for disjoint set union [15], which has a worst-case time per operation of $O(\log n)$ but an amortized time of $O(\alpha(m, n))$ [13], where n is the number of elements in the sets, m is the number of operations, and α is an inverse of Ackerman's function, which grows very slowly.

Data structures efficient in the worst case typically obtain their efficiency from an explicit structural constraint, such as the balance condition found in each of the many kinds of balanced trees. Maintaining such a structural constraint consumes both running time and storage space, and tends to produce complicated updating algorithms with many cases. Implementing such data structures can be tedious.

If we are content with a data structure that is efficient in only an amortized sense, there is another way to obtain efficiency. Instead of imposing any explicit structural constraint, we allow the data structure to be in an arbitrary state, but we design the access and update algorithms to adjust the structure in a simple, uniform way, so that the efficiency of future operations is improved. We call such a data structure *self-adjusting.*

* Received by the editors October 12, 1983, and in revised form September 15, 1984.
† AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

Self-adjusting data structures have the following possible advantages over explicitly balanced structures:

(i) They need less space, since no balance information is kept.

(ii) Their access and update algorithms are easy to understand and to implement.

(iii) In an amortized sense, ignoring constant factors, they can be at least as efficient as balanced structures.

Self adjusting structures have two possible disadvantages:

(i) More local adjustments take place than in the corresponding balanced structures, especially during accesses. (In a balanced structure, adjustments usually take place only during updates, not during accesses.) This can cause inefficiency if local adjustments are expensive.

(ii) Individual operations within a sequence can be very expensive. Although expensive operations are likely to be rare, this can be a drawback in real-time applications.

In this paper we develop and analyze the *skew heap*, a self-adjusting form of heap (priority queue) analogous to leftist heaps [4], [7]. The fundamental operation on skew heaps is *melding*, which combines two disjoint heaps into one. In § 2 we present the basic form of skew heaps, which use top-down melding. In § 3 we discuss skew heaps with bottom-up melding, which are more efficient for insertion and melding. In § 4 we study various less common heap operations. In § 5 we show that in an amortized sense skew heaps are optimal to within a constant factor on any sequence of certain operations. Section 6 compares skew heaps to other heap implementations, and contains additional remarks and open problems. The appendix contains our tree terminology.

This paper represents only a part of our work on amortized complexity and self-adjusting data structures; companion papers discuss self-adjusting lists [12] and self-adjusting search trees [13]. Some of our results have previously appeared in preliminary form [11].

2. Skew heaps. A *heap* (sometimes called a *priority queue* [8] or *mergeable heap* [1]) is an abstract data structure consisting of a set of items selected from a totally ordered universe, on which the following operations are possible:

function *make heap*(h): Create a new, empty heap, named h.

function *find min*(h): Return the minimum item in heap h. If h is empty, return the special item "null".

procedure *insert*(x, h): Insert item x in heap h, not previously containing it.

function *delete min*(h): Delete the minimum item from heap h and return it. If the heap is initially empty, return null.

function *meld*(h_1, h_2): Return the heap formed by taking the union of disjoint heaps h_1 and h_2. This operation destroys h_1 and h_2.

There are several ways to implement heaps in a self-adjusting fashion. The one we shall discuss is an analogue of the leftist heaps proposed by Crane [4] and refined by Knuth [8]. To represent a heap, we use a *heap-ordered binary tree*, by which we mean a binary tree whose nodes are the items, arranged in heap order: if $p(x)$ is the parent of x, then $p(x) < x$. (For simplicity we shall assume that each item is in at most one heap; this restriction is easily lifted by regarding the tree nodes and heap items as distinct, with a pointer in each tree node pointing to the corresponding heap item.) To represent such a tree we store with each item x two pointers, *left*(x) and *right*(x), to its left child and right child respectively. If x has no left child we define *left*(x) = null; if x has no right child we define *right*(x) = null. Access to the tree is by a pointer to

its root; we represent an empty tree by a pointer to null. We shall sometimes denote an entire tree or subtree by its root, with the context resolving the resulting ambiguity.

With a representation of heaps as heap-ordered binary trees, we can carry out the various heap operations as follows. We perform *make heap*(h) in $O(1)$ time by initializing h to null. Since heap order implies that the root is the minimum item in a tree, we can carry out *find min* in $O(1)$ time by returning the root. We perform *insert* and *delete min* using *meld*. To carry out *insert*(x, h), we make x into a one-node heap and meld it with h. To carry out *delete min*(h), we replace h by the meld of its left and right subtrees and return the original root.

To perform *meld*(h_1, h_2), we form a single tree by traversing the right paths of h_1 and h_2, merging them into a single right path with items in increasing order. The left subtrees of nodes along the merge path do not change. (See Fig. 1.) The time for the meld is bounded by a constant times the length of the merge path. To make melding efficient, we must keep right paths short. In leftist heaps this is done by maintaining the invariant that, for any node x, the right path descending from x is a shortest path down to a null node. Maintaining this invariant requires storing at every node the length of a shortest path down to a missing node; after a meld we walk back up the merge path, updating shortest path lengths and swapping left and right children as necessary to maintain the leftist property. The length of the right path in a leftist tree of n nodes is at most $\lfloor \log n \rfloor$[1], implying an $O(\log n)$ worst-case time bound for each of the heap operations, where n is the number of nodes in the heap or heaps involved.

In our self-adjusting version of this data structure, we meld by merging the right paths of the two trees and then swapping the left and right children of *every* node on the merge path except the lowest. (See Fig. 1.) This makes the potentially long right path formed by the merge into a left path. We call the resulting data structure a *skew heap*.

In our analysis of skew heaps, we shall use the following general approach. We associate with each possible collection S of skew heaps a real number $\Phi(S)$ called the *potential* of S. For any sequence of m operations with running times $t_1, t_2, \cdots, t_m$, we define the *amortized time* a_i of operation i to be $a_i = t_i + \Phi_i - \Phi_{i-1}$, where Φ_i, for $i = 1, 2, \cdots, m$, is the potential of the skew heaps after operation i and Φ_0 is the potential of the skew heaps before the first operation. The total running time of the sequence of operations is then

$$\sum_{i=1}^{m} t_i = \sum_{i=1}^{m} (a_i - \Phi_i + \Phi_{i-1}) = \Phi_0 - \Phi_m + \sum_{i=1}^{m} a_i.$$

That is, the total running time equals the total amortized time plus the decrease in potential from the initial to the final collection of heaps. In most of our analyses the potential will be zero initially and will remain nonnegative. If this is the case then the total amortized time is an upper bound on the actual running time.

This definition is purely formal; its utility depends on the ability to choose a potential function that results in small amortized times for the operations. Whenever we use this technique we shall define the potential of a single heap; the potential of a collection is the sum of the potentials of its members. Intuitively, a heap with high potential is one subject to unusually time-consuming operations; the extra time spent corresponds to a drop in potential.

We shall prove an $O(\log n)$ bound on the amortized time of skew heap operations. To do this we define the potential function using an idea of Sleator and Tarjan [9],

[1] Throughout this paper we use base-two logarithms.

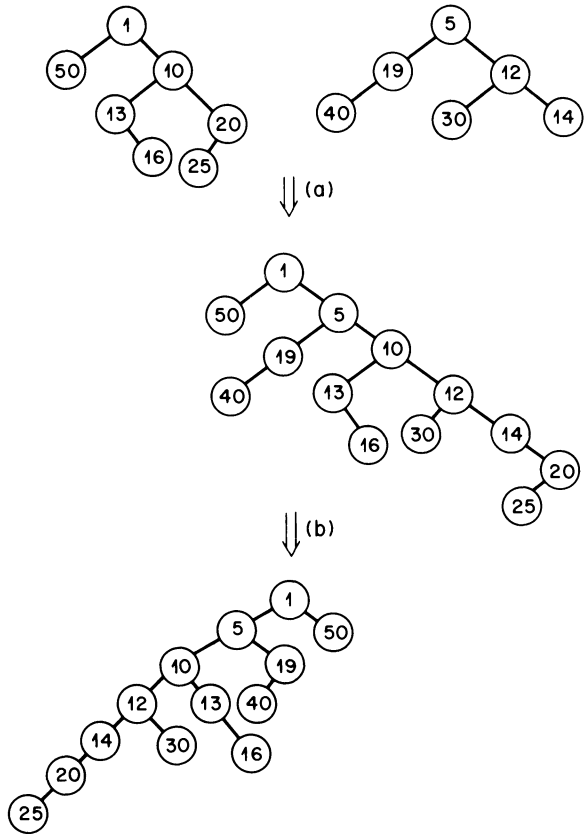


FIG. 1. *A meld of two skew heaps.* (a) *Merge of the right paths.* (b) *Swapping of children along the path formed by the merge.*

[10]. For any node x in a binary tree, we define the *weight* $wt(x)$ of x as the number of descendants of x, including x itself. We use the weights to partition the nonroot nodes into two classes: a nonroot node x is *heavy* if $wt(x) > wt(p(x))/2$ and *light* otherwise. We shall regard a root as being neither heavy nor light. The following lemmas are immediate from this definition.

LEMMA 1. *Of the children of any node, at most one is heavy.*

Proof. A heavy child has more than half the weight of its parent. This can be true of only one child. □

LEMMA 2. *On any path from a node x down to a descendent y, there are at most $\lfloor \log(wt(x)/wt(y)) \rfloor$ light nodes, not counting x. In particular, any path in an n-node tree contains at most $\lfloor \log n \rfloor$ light nodes.*

Proof. A light child has at most half the weight of its parent. Thus if there are k light nodes not including x along the path from x to y, $wt(y) \leqq wt(x)/2^k$, which implies $k \leqq \log(wt(x)/wt(y))$. □

We define the potential of a skew heap to be the total number of right heavy nodes it contains. (A nonroot node is *right* if it is a right child and *left* otherwise.) The intuition justifying this choice of potential is as follows. By Lemma 2, any path in a skew heap, and in particular any path traversed during melding, contains only $O(\log n)$ light nodes. Any heavy node on such a path is converted from right to left by the meld, causing a drop of one in the potential. As we shall prove rigorously below,

this implies that any melding time in excess of $O(\log n)$ is covered by a drop in the potential, giving an amortized melding time of $O(\log n)$.

Suppose we begin with no heaps and carry out an arbitrary sequence of skew heap operations. The initial potential is zero and the final potential is nonnegative, so the total running time is bounded by the sum of the amortized times of the operations. Furthermore, since the potential of a skew heap of n items is at most $n-1$, if we begin with any collection of skew heaps and carry out an arbitrary sequence of operations, the total time is bounded by the total amortized time plus $O(n)$, where n is the total size of the initial heaps.

The amortized time of a *find min* operation is $O(1)$, since the potential does not change. The amortized times of the other operations depend on the amortized time for *meld*. Consider a meld of two heaps h_1 and h_2, containing n_1 and n_2 items, respectively. Let $n = n_1 + n_2$ be the total number of items in the two heaps. As a measure of the melding time we shall charge one per node on the merge path. Thus the amortized time of the meld is the number of nodes on the merge path plus the change in potential. By Lemma 1, the number of light nodes on the right paths of h_1 and h_2 is at most $\lfloor \log n_1 \rfloor$ and $\lfloor \log n_2 \rfloor$, respectively. Thus the total number of light nodes on the two paths is at most $2\lfloor \log n \rfloor - 1$. (See Fig. 2.)

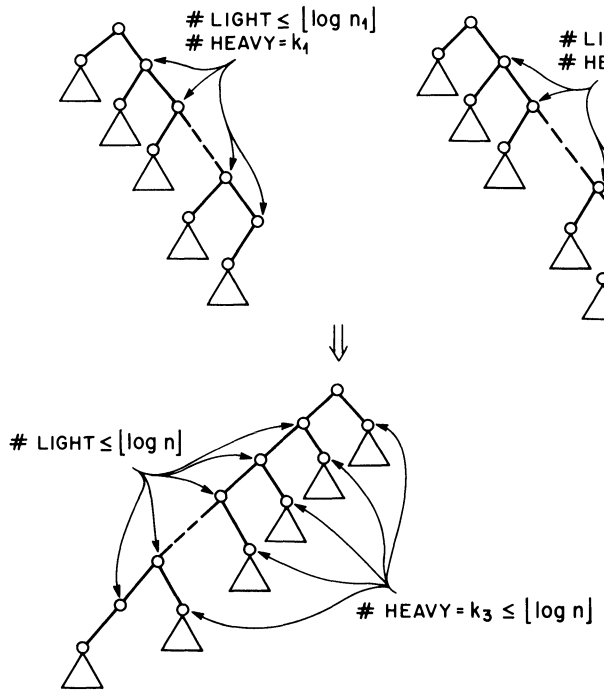


FIG. 2. *Analysis of right heavy nodes in meld.*

Let k_1 and k_2 be the number of heavy nodes on the right paths of h_1 and h_2, respectively, and let k_3 be the number of nodes that become right heavy children of nodes on the merge path. Every node counted by k_3 corresponds to a light node on the merge path. Thus Lemma 2 implies that $k_3 \leqq \lfloor \log n \rfloor$.

The number of nodes on the merge path is at most $2+\lfloor\log n_1\rfloor+k_1+\lfloor\log n_2\rfloor+k_2\leqq 1+2\lfloor\log n\rfloor+k_1+k_2$. (The "2" counts the roots of h_1 and h_2.) The increase in potential caused by the meld is $k_3-k_1-k_2\leqq\lfloor\log n\rfloor-k_1-k_2$. Thus the amortized time of the meld is at most $3\lfloor\log n\rfloor+1$.

THEOREM 1. *The amortized time of an insert, delete min, or meld skew heap operation is $O(\log n)$, where n is the number of items in the heap or heaps involved in the operation. The amortized time of a make heap or find min operation is $O(1)$.*

Proof. The analysis above gives the bound for *find min* and *meld*; the bound for *insert* and *delete min* follows immediately from that of *meld*. □

One may ask whether amortization is really necessary in the analysis of skew heaps, or whether skew heaps are efficient in a worst-case sense. Indeed they are not: we can construct sequences of operations in which some operations take $O(n)$ time. For example, suppose we insert n, $n+1$, $n-1$, $n+2$, $n-2$, $n+3,\cdots,1$, $2n$ into an initially empty heap and then perform *delete min*. The tree resulting from the insertions has a right path of n nodes, and the *delete min* takes $\Omega(n)$ time. (See Fig. 3.) There are similar examples for the other versions of skew heaps we shall consider.

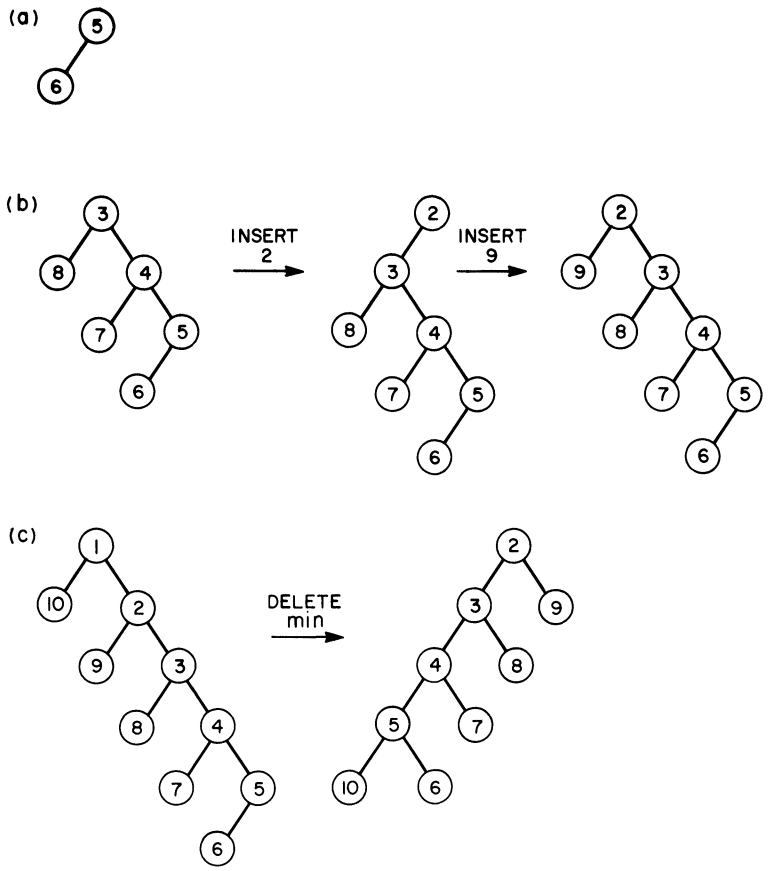


FIG. 3. *Insertion of* 5, 6, 4, 7, 3, 8, 2, 9, 1, 10 *into an initially empty heap, followed by delete min.* (a) *The heap after two insertions.* (b) *Insertion of* 2 *and* 9. (c) *The delete min operation.*

The following programs, written in an extension of Dijkstra's guarded command language [5], implement the various skew heap operations. A **val** parameter to a function or procedure is called by value; a **var** parameter is called by value and result. The double arrow "↔" denotes swapping. Parallel assignments all take place simultaneously.

```
function make heap;
  return null
end make heap;

function find min(val h);
  return h
end find min;

procedure insert(val x, var h);
  left(x) := right(x) := null;
  h := meld(x, h)
end insert;

function delete min(var h);
  var x;
  x := h; h := meld(left(h), right(h)); return x
end delete min;
```

Our implementation of *meld* differs slightly from the informal description. The program traverses the two right paths from the top down, merging them and simultaneously swapping left and right children. When the bottom of one of the paths is reached, the remainder of the other path is simply attached to the bottom of the merge path, and the process terminates. Only the nodes visited have their children exchanged; the last node whose children are exchanged is the lowest node on the one of the two paths that is completely traversed. (See Fig. 4.) In the informal description, all nodes on both right paths are visited. Theorem 1 holds for the actual implementation; the same proof applies if k_1 and k_2 are redefined to be the number of heavy nodes on the right paths of h_1 and h_2 actually traversed during the meld.

We shall give two versions of *meld*: a recursive version, *rmeld*, and an iterative version, *imeld*. The recursive version uses an auxiliary function *xmeld* to do the actual melding, in order to avoid redundant tests for null.

```
function rmeld(val h1, h2);
  return if h2 = null → h1 [] h2 ≠ null → xmeld(h1, h2) fi
end rmeld;

function xmeld(val h1, h2);
  [h2 ≠ null]
  if h1 = null → return h2 fi;
  if h1 > h2 → h1 ↔ h2 fi;
  left(h1), right(h1) := xmeld(right(h1), h2), left(h1);
  return h1
end xmeld;
```

The iterative version of *meld* uses four variables, x, y, h_1, and h_2, and maintains the following loop invariant: if $left(y)$ is replaced by null, then x, h_1, and h_2 are the roots of three disjoint heaps containing all the nodes; y is the bottommost node of the left path down from x (the merge path) and is such that $y < \min\{h_1, h_2\}$.

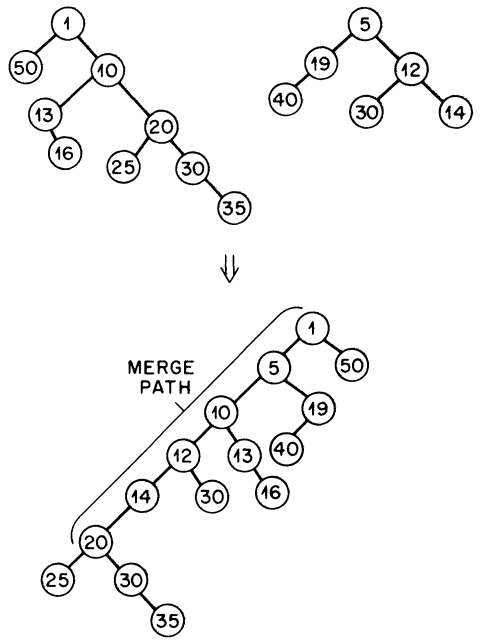


FIG. 4. *Implemented version of top-down melding.*

```
function imeld(val h1, h2);
  var x, y;
  if h1 = null → return h2 ▯ h2 = null → return h1 fi;
  if h1 > h2 → h1↔h2 fi;
  x, y, h1, right(h1) := h1, h1, right(h1), left(h1);
  do h1 ≠ null →
    if h1 > h2 → h1↔h2 fi;
    y, left(y), h1, right(h1) := h1, h1, right(h1), left(h1)
  od;
  left(y) := h2;
  return x
end imeld;
```

Note. The swapping of h_1 and h_2 in the loop can be avoided by writing different pieces of code for the cases $h_1 > h_2$ and $h_1 \leqq h_2$. The four-way parallel assignment can be written as the following four sequential assignments: $left(y) := h_1$; $y := h_1$; $h_1 := right(y)$; $right(y) := left(y)$. The assignment "$y := h_1$" can be deleted by unrolling the loop. With these changes a meld takes $O(1)$ time plus three assignments and two comparisons per node on the merge path.

One possible drawback of skew heaps is the number of pointer assignments needed. We can reduce the pointer updating by storing in each node a bit indicating that the pointers to the left and right children have been reversed. Children can then be swapped merely by changing a bit. This idea trades bit assignments for pointer assignments but takes extra space and complicates the implementation.

3. Bottom-up skew heaps. In some applications of heaps, such as in the computation of minimum spanning trees [3], [16], it is important that melding be as efficient as possible. By melding skew heaps bottom-up instead of top-down, we can reduce

the amortized time of *insert* and *meld* to $O(1)$ without affecting the time bounds of the other operations. If h_1 and h_2 are the heaps to be melded, we walk up the right paths of h_1 and h_2, merging them and exchanging the children of all nodes on the merge path except the lowest. When reaching the top of one of the heaps, say h_1, we attach the root of h_1 (the top node on the merge path) as the right child of the lowest node remaining on the right path of h_2. The root of h_1 is the last node to have its children swapped, unless h_1 is the *only* node on the merge path, in which case no swapping takes place. (See Fig. 5.)

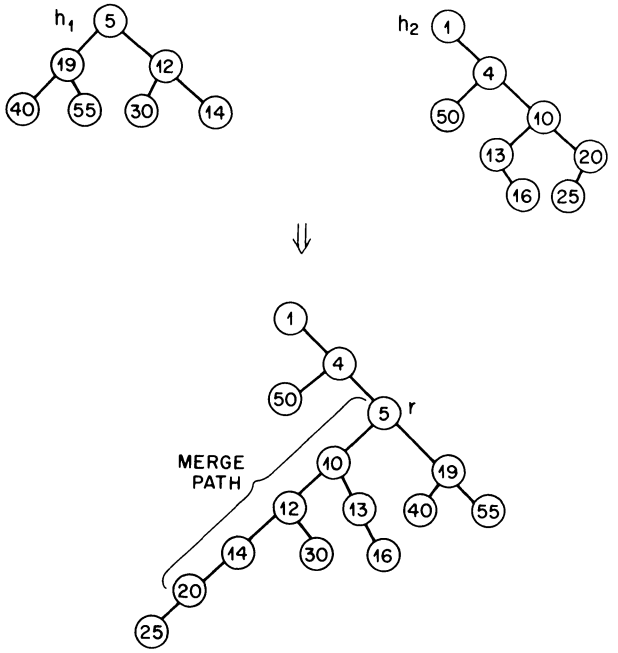


FIG. 5. *Bottom-up melding.*

We can implement this method by storing with each node x an extra pointer $up(x)$, defined to be the parent of x if x is a right child, or the lowest node on the right path descending from x if x is a left child or a root. Thus right paths are circularly linked, bottom-up. (See Fig. 6.) We call this the *ring representation.* We shall consider alternative representations at the end of the section.

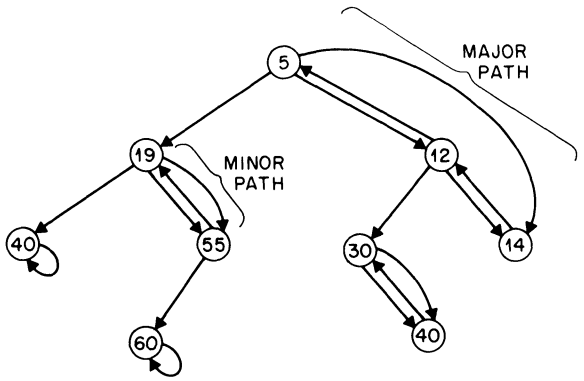


FIG. 6. *Ring representation of a skew heap.*

Obtaining good amortized time bounds for the various operations on bottom-up skew heaps requires a more complicated potential function than the one used in § 2. To define the potential, we need two subsidiary definitions. If T is any binary tree, we define the *major path* of T to be the right path descending from the root, and the *minor path* to be the right path descending from the left child of the root. (See Fig. 6.) We define node weights and light and heavy nodes as in § 2. We define the potential of a bottom-up skew heap to be the number of right heavy nodes in the tree plus twice the number of right light nodes on the major and minor paths.

Consider a bottom-up meld of two skew heaps h_1 and h_2. We shall show that the amortized time of the meld is $O(1)$, if we count one unit of running time per node on the merge path. Let h_3 be the melded heap, and suppose, without loss of generality, that h_1 is the heap whose top is reached during the meld. Let r be the root of h_1. (See Fig. 5.) The merge path is the top part of the left path descending from r in h_3. It contains all nodes on the major path of h_1 and possibly some of the nodes on the major path of h_2. The major path of h_3 consists of the nodes on the major path of h_2 not on the merge path, node r, and, if the merge path contains two or more nodes, the minor path of h_1. The minor path of h_3 is the minor path of h_2. The only nodes whose weights change during the merge are those on the major paths of h_1 and h_2; the weights of these nodes can increase but not decrease.

Consider the change in potential caused by the meld. Any node on the major path of h_2 not on the merge path can gain in weight, becoming a right heavy instead of a right light node. Each such change decreases the potential by one. No such node can change from heavy to light, because the weights of both it and its parent increase by the same amount. Node r, the root of h_1, becomes a node on the major path of h_3, increasing the potential by one if r becomes heavy or two if r becomes light. The top node on the minor path of h_1 also can become a node on the major path of h_3, increasing the potential by at most two. The remaining nodes on the minor path of h_1 and the nodes on the minor path of h_2 are associated with no change in the potential.

It remains for us to consider the nodes other than r on the merge path. Let x be such a node. If x is originally heavy, it remains heavy but becomes left, causing a decrease of one in potential (the new right sibling of x is light). If x is originally light, its new right sibling may be heavy, but there is still a decrease of at least one in potential (from two units for x as a light node on a major path to at most one unit for the new right sibling of x, which is not on a major or minor path).

Combining these estimates, we see that the amortized meld time, a, defined to be the number of nodes on the merge path, t, plus the change in potential, $\Delta\Phi$, satisfies $a = t + \Delta\Phi \leqq 5$: the potential decreases by at least one for each node on the merge path except r, and increases by at most two for r and two for the top node on the minor path of h_1.

THEOREM 2. *The amortized time of a make heap, find min, insert, or meld operation on bottom-up skew heaps is* $O(1)$. *The amortized time for a delete min operation on an* n*-node heap is* $O(\log n)$.

Proof. Both the worst-case and amortized times of *make heap* and *find min* are $O(1)$, since neither causes a change in potential. The $O(1)$ amortized time bound for *insert* and *meld* follows from the analysis above. Consider a *delete min* operation on an n-node skew heap h. Deleting the root of h takes $O(1)$ time and produces two skew heaps h_1 and h_2, whose roots are the left and right children of the root of h, respectively. This increases the potential by at most $4 \log n$: there is an increase of two units for each light right node on the minor paths of h_1 and h_2. (The major path of h, minus the root, becomes the minor path of h_2; the minor path of h becomes the major path

of h_1.) The meld that completes the deletion takes $O(1)$ amortized time, giving a total amortized deletion time of $O(\log n)$. □

Since any n-node bottom-up skew heap has a potential of $O(n)$, if we begin with any collection of such heaps and carry out any sequence of operations, the total time is bounded by the total amortized time plus $O(n)$, where n is the total size of the initial heaps. If we begin with no heaps, the total amortized time bounds the total running time.

The ring representation of bottom-up skew heaps is not entirely satisfactory, since it needs three pointers per node instead of two. The extra pointer is costly in both storage space and running time, because it must be updated each time children are swapped. There are several ways to reduce the number of pointers to two per node. We shall discuss two in this section and a third in § 4.

One possible change is to streamline the ring representation by not storing right child pointers. An even more appealing possibility is to store with each node an *up* pointer to its parent and a *down* pointer to the lowest node on the right path of its left subtree. The *up* pointer of the root, which has no parent, points to the lowest node on the major path. The *down* pointer of a node with empty left subtree points to the node itself. We shall call this the *bottom-up representation* of a skew heap.

Both of these representations will support all the heap operations in the time bounds of Theorem 2. We shall discuss the bottom-up representation; we favor it because swapping children, which is the local update operation on skew heaps, is especially easy.

The implementations of *make heap* and *find min* are exactly as presented in § 2. We shall give two implementations of *insert*. The first merely invokes *meld*. The second includes an in-line customized version of *meld* for greater efficiency.

```
procedure insert (val x, var h);
  up(x) := down(x) := x;
  h := meld(x, h)
end insert;
```

The more efficient version of *insert* tests for three special cases: $h = \text{null}$, $x < h$ (x becomes the root of the new tree), and $x > up(h)$ (x becomes the new lowest node on the major path). In the general case, x is inserted somewhere in the middle of the major path by a loop that uses two local variables, y, and z; y is the highest node on the major path so far known to satisfy $y > x$, and z is the lowest node on the major path (after the swapping of children that has taken place so far).

```
procedure insert (val x; var h);
  var y, z;
  if h = null → h := up(x) := down(x) := x; return fi;
  if x < h → down(x) := up(h); h := up(x) := up(h) := x; return fi;
  if x > up(h) → up(x) := up(h); down(x) := up(h) := x; return fi;
  y := z := up(h);
  do x < up(y) → y := up(y); z ↔ down(y) od;
  up(x), down(x) := up(y), z;
  up(y) := up(h) := x
end insert;
```

The following program implements *meld*. The program uses two variables, h_3 and x, to hold the output heap and the next item to be added to the output heap, respectively.

```
function meld(val h1, h2);
  var h3, x;
  if h1 = null → return h2 ▯ h2 = null → return h1 fi;
  if up(h1) < up(h2) → h1 ↔ h2 fi;
  [initialize h3 to hold the bottom right node of h1]
  h3 := up(h1); up(h1) := up(h3); up(h3) := h3;
  do h1 ≠ h3 →
    if up(h1) < up(h2) → h1 ↔ h2 fi;
    [remove from h1 its bottom right node, x]
    x := up(h1); up(h1) := up(x);
    [add x to the top of h3 and swap its children]
    up(x) := down(x); down(x) := up(h3); h3 := up(h3) := x
  od;
  [attach h3 to the bottom right of h2]
  up(h2) ↔ up(h3);
  return h2
end meld;
```

The only cleverness in this code is in the termination condition of the loop. Just before the last node is removed from heap h_1, $up(h_1) = h_1$, which means that at the beginning of the next iteration $h_1 = h_3$, and the loop will terminate. The code contains an annoyingly large number of assignments. Some of these can be removed, and the efficiency of the program improved, by storing $up(h_1)$, $up(h_2)$, and $up(h_3)$ in local variables and unrolling the **do** loop to store state information in the flow of control. This obscures the algorithm, however.

Implementing *delete min* poses a problem: there is no way to directly access the children of the root (the node to be deleted) to update their *up* pointers. We can overcome this problem by performing the deletion as follows: we merge the minor and major paths of the tree, swapping children all along the merge path. We stop the merge *only* when the root is reached along *both* paths. The following program implements this method. The program uses four local variables: h_3 is the output heap, y_1 and y_2 are the current nodes on the major and minor paths, and x is the next node to be added to the output heap. The correctness of the program depends on the assumption that an item appears only once in a heap. As in the case of *meld*, we can improve the efficiency somewhat by storing $up(h_3)$ in a local variable and unrolling the loop.

```
function delete min(var h);
  var x, y1, y2, h3;
  if h = null → return null fi;
  y1, y2 := up(h), down(h);
  if y1 < y2 → y1 ↔ y2 fi;
  if y1 = h → h = null; return y1 fi;
  [initialize h3 to hold y1]
  h3 := y1; y1 := up(y1); up(h3) := h3;
  do true →
    if y1 < y2 ↔ y2 fi;
    if y1 = h → h := h3; return y1 fi;
    [remove x = y1 from its path]
    x := y1; y1 := up(y1);
    [add x to the top of h3 and swap its children]
    up(x) := down(x); down(x) := up(h3); h3 := up(h3) := x
  od
end delete min;
```

The same potential function and the same argument used at the beginning of this section prove Theorem 2 for the bottom-up representation of skew heaps; namely, *make heap* and *find min* take $O(1)$ time (both amortized and worst case), *insert* and *meld* take $O(1)$ amortized time, and *delete min* on an n-node heap takes $O(\log n)$ amortized time.

4. Other operations on skew heaps. There are a variety of additional heap operations that are sometimes useful. In this section we shall consider the following four:

function *make heap*(s): Return a new heap whose items are the elements in set s.
function *find all*(x, h): Return the set of all items in h less than or equal to x.
procedure *delete*(x, h): Delete item x from heap h.
procedure *purge*(h): Assuming that each item in heap h is marked "good" or "bad", delete enough bad items from h so that the minimum item left in the heap is good.

The operation *make heap*(s) is an extension of *make heap* as defined in § 2 and allows us to initialize a heap of arbitrary size. The operation *find all* is a form of range query. The *delete* operation allows deletion from a heap of any item, not just the minimum one. An alternative way to delete arbitrary items is with the *purge* operation, using *lazy deletion*: When inserting an item in a heap, we mark it "good". When deleting an item we do not alter the structure of the heap but merely mark the item "bad". Before any *find min* or *delete min* operation, we purge the heap. Lazy deletion is especially useful when items can be marked for deletion implicitly, as in the computation of minimum spanning trees using heaps [3], [16]. Deleting many bad items simultaneously reduces the time per deletion. The only drawback of lazy deletion is that a deleted item cannot be reinserted until it has been purged. We can overcome this by copying the item, at a cost of extra storage space.

We shall begin by implementing the four new operations on the top-down skew heaps of § 2. The operations *make heap*(s) and *purge* are best treated as special cases of the following more general operation:

function *heapify*(s): Return a heap formed by melding all the heaps in the set s. This operation assumes that the heaps in s are disjoint and destroys them in the process of melding them.

As discussed by Tarjan [16], we can carry out *heapify*(s) by repeated pairwise melding. We perform a number of passes through the set s. During each pass, we meld the heaps in s in pairs; if the number of heaps is odd, one of them is not melded until a subsequent pass. We repeat such passes until there is only one heap left, which we return.

This method is efficient for any heap representation that allows two heaps of total size n to be melded in $O(\log n)$ time. To analyze its running time, consider a single pass. Let k be the number of heaps in s before the pass and let n be their total size. After the pass, s contains only $\lceil k/2 \rceil \leqq 2k/3$ heaps. The time for the pass is $O(k + \sum_{i=1}^{\lceil k/2 \rceil} \log n_i)$, where n_i is the number of items in the ith heap remaining after the pass. The sizes n_i satisfy $1 \leqq n_i \leqq n$ and

$$\sum_{i=1}^{\lceil k/2 \rceil} n_i \leqq n.$$

The convexity of the log function implies that the time bound is maximum when all

the n_i are equal, which gives a time bound for the pass of $O(k+\log(n/k))$. Summing over all passes, the time for the entire *heapify* is

$$O\left(\sum_{i=0}^{\lfloor \log 3/2k \rfloor} \left(\frac{2}{3}\right)^i \left(k+k\log\left(\left(\frac{3}{2}\right)^i \frac{n}{k}\right)\right)\right) = O\left(k+k\log\left(\frac{n}{k}\right)\right),$$

where k is the original number of heaps and n is the total number of items they contain. For skew heaps, this is an amortized time bound. For worst-case data structures such as leftist or binomial heaps [2], [16], this is a worst-case bound.

We can perform *make heap*(s) by making each item into a one-item heap and applying *heapify* to the set of these heaps. Since $k = n$ in this case, the time for *make heap* is $O(n)$, both amortized and worst-case.

We can perform *purge*(h) by traversing the tree representing h in preorder, deleting every bad node encountered and saving every subtree rooted at a good node encountered. (When visiting a good node, we immediately retreat, without visiting its proper descendants.) We complete the purge by heapifying the set of subtrees rooted at visited good nodes. If k nodes are purged from a heap of size n, the time for the purge is $O(k+k\log(n/k))$, since there are at most $k+1$ subtrees to be heapified. For skew heaps, this is an amortized bound.

We can carry out *find all* on any kind of heap-ordered binary tree in time proportional to the size of the set returned. We traverse the tree in preorder starting from the root, listing every node visited that does not exceed x. When we encounter a node greater than x, we immediately retreat, without visiting its proper descendants. Note that since heap order is not a total order, *find all* cannot return the selected items in sorted order without performing additional comparisons.

If *find all* is used in combination with lazy deletion and it is not to return bad items, it must purge the tree as it traverses it. The idea is simple enough: When traversing the tree looking for items not exceeding x, we discard every bad item encountered. This breaks the tree into a number of subtrees, which we heapify. It is not hard to show that a *find all* with purging that returns j good items and purges k bad items from an n-item heap takes $O(j+k+k\log(n/k))$ amortized time. We leave the proof of this as an exercise.

The fourth new operation is deletion. We can delete an arbitrary item x from a top-down skew heap (or, indeed, from any kind of heap-ordered binary tree) by replacing the subtree rooted at x by the meld of its left and right subtrees. This requires maintaining parent pointers for all the nodes, since deleting x changes one of the children of $p(x)$. The amortized time to delete an item from an n-node heap is $O(\log n)$. In addition to the $O(\log n)$ time for what is essentially a *delete min* on the subtree rooted at x, there is a possible additional $O(\log n)$ gain in potential, since deleting x reduces the weights of proper ancestors of x, causing at most $\log n$ of them to become light: their siblings, which may be right, may become heavy.

By changing the tree representation we can carry out all the heap operations, including arbitrary deletion, top-down using only two pointers per node. Each node points to its leftmost child (or to null if it has no children) and to its right sibling, or to its parent if it has no right sibling. (The root, having neither a right sibling nor a parent, points to null.) Knuth calls this the "binary tree representation of a tree, with right threads"; we shall call it the *threaded representation.* (See Fig. 7.) With the threaded representation, there is no way to tell whether an only child is left or right, but for our purposes this is irrelevant; we can regard every only child as being left. From any node we can access its parent, left child, or right child by following at most two pointers, which suffices for implementing all the heap operations.

Now let us consider the four new operations on bottom-up skew heaps. Assume for the moment that we use the ring representation. (See Fig. 6.) The operation *heapify*(s) is easy to carry out efficiently: We merely meld the heaps in s in any order. This takes $O(k)$ amortized time if there are k heaps in s. In particular, we can initialize a heap of n items in $O(n)$ time (amortized and worst-case) by performing n successive insertions into an initially empty heap. We can purge k items from a heap of n items in $O(k+k(\log n/k))$ amortized time by traversing the tree in preorder and melding the subtrees rooted at visited good nodes in arbitrary order. The main contribution to the time bound is not the melds, which take $O(k)$ amortized time, but the increase in potential caused by breaking the tree into subtrees: a subtree of size n_i gains in potential by at most $4\log n_i$ because of the light right nodes on its major and minor paths. (See the definition of potential used in § 3.) The total potential increase is maximum when all the subtrees are of equal size and is $O(k+k\log(n/k))$.

The operation *find all* is exactly the same on a bottom-up skew heap as on a top-down skew heap, and takes time proportional to the number of items returned. A *find all* with purging on a bottom-up skew heap is similar to the same operation on a top-down skew heap, except that we can meld the subtrees remaining after bad items are deleted in any order. The amortized time bound is the same, namely $O(j+k+k\log(n/k))$ for a *find all* with purging that returns j good items and purges k bad items from an n-item heap.

Arbitrary deletion on bottom-up skew heaps requires changing the tree representation, since the ring representation provides no access path from a left child to its parent. We shall describe a representation that supports all the heap operations, including bottom-up melding, and needs only two pointers per node. The idea is to use the threaded representation proposed above for top-down heaps, but to add a pointer to the lowest node on the right path. (See Fig. 7.) We shall call this extra pointer the *down pointer.*

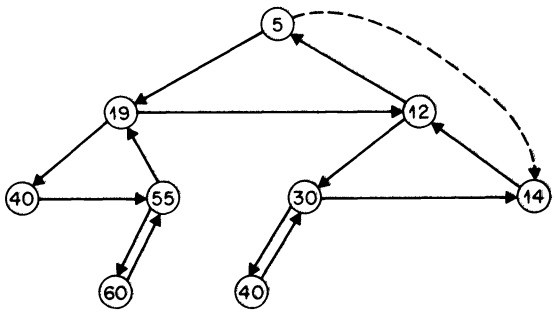


FIG. 7. *Threaded representation of a skew heap. All only children are regarded as being left. The dashed pointer is the down pointer, used only in the bottom-up version.*

When performing a heap operation, we reestablish the down pointer when necessary by running down the appropriate right path. For example, suppose we meld two heaps h_1 and h_2 bottom-up, and that heap h_1 is exhausted first, so that the root of h_1, say r, becomes the top node on the merge path. We establish the down pointer in the melded heap by descending the new right path from r; this is the minor path in the original heap h_1 unless the merge path contains only r, in which case r has a null right child. (See Fig. 5.)

To perform a *delete min* operation, we descend the minor path of the heap to establish a down pointer for the left subtree, and then we meld the left and right

subtrees; the down pointer for the right subtree is the same as the down pointer for the entire tree. We perform *purge* as described above for bottom-up skew heaps, establishing a down pointer for every subtree to be melded by traversing its right path.

Arbitrary deletion is the hardest operation to implement, because if the deleted node is on the right path the down pointer may become invalid, and discovering this requires walking up toward the root, which can be expensive. One way to delete an arbitrary node x is as follows. First, we replace the subtree rooted at x by the meld of its left and right subtrees, using either top-down or bottom-up melding. Next, we walk up from the root of the melded subtree until reaching either a left child or the root of the entire tree. During this walk we swap the children of every node on the path except the lowest. If the node reached is the root, we reestablish the left pointer by descending the major path (which was originally the minor path).

By using an appropriate potential function, we can derive good amortized time bounds for this representation. We must approximately double the potential used in § 3, to account for the extra time spent descending right paths to establish down pointers. We define the potential of a tree to be twice the number of heavy right nodes not on the major path, plus the number of heavy right nodes on the major path, plus four times the number of light right nodes on the minor path, plus three times the number of light right nodes on the major path. Notice that a right node on the minor path has one more credit than it would have if it were on the major path. The extra credits on the minor path pay for a traversal of it when necessary to establish the down pointer. A straightforward extension of the analysis in § 3 proves the following theorem.

THEOREM 3. *On bottom-up skew heaps represented in threaded fashion with down pointers, the heap operations have the following amortized running times:* $O(1)$ *for make heap, find min, insert, and meld;* $O(\log n)$ *for delete min or delete on an n-node heap;* $O(n)$ *for make heap(s) on a set of size n;* $O(k+k\log(n/k))$ *for purge on an n-node heap if k items are purged;* $O(j+k+k\log(n/k))$ *for find all with purging on an n-node heap if j items are returned and k items are purged.*

5. A lower bound. The notion of amortized complexity affords us the opportunity to derive lower bounds, as well as upper bounds, on the efficiency of data structures. For example, the compressed tree data structure for disjoint set union is optimal to within a constant factor in an amortized sense among a wide class of pointer manipulation algorithms [15]. We shall derive a similar but much simpler result for bottom-up skew heaps: in an amortized sense, skew heaps are optimal to within a constant factor on any sequence of certain heap operations.

To simplify matters, we shall allow only the operations *meld* and *delete min.* We assume that there is an arbitrary initial collection of single-item heaps and that an arbitrary but fixed sequence of *meld* and *delete min* operations is to be carried out. An algorithm for carrying out this sequence must return the correct answers to the *delete min* operations whatever the ordering of the items; this ordering is initially completely unspecified. The only restriction we make on the algorithm is that it make binary rather than multiway decisions; thus binary comparisons are allowed but not radix sorting, for example.

Suppose there are a total of m operations, and that the *ith delete min* operation is on a heap of size n_i. If we carry out the sequence using bottom-up skew heaps, the total running time is $O(m+\sum_i \log n_i)$. We shall prove that any correct algorithm must make at least $\sum_i \log n_i$ binary decisions; thus, if we assume that any operation takes $\Omega(1)$ time, bottom-up skew heaps are optimum to within a constant factor in an amortized sense.

The proof is a simple application of information theory. The various possible orderings of the items produce different correct outcomes for the *delete min* instructions. For an algorithm making binary decisions, the binary logarithm of the total number of possible outcomes is a lower bound on the number of decisions in the worst case. The ith *delete min* operation has n_i possible outcomes, regardless of the outcomes of the previous *delete min* operations. (Any item in the heap can be the minimum.) Thus the total number of possible outcomes of the entire sequence is $\prod_i n_i$, and the number of binary decisions needed in the worst case is $\sum_i \log n_i$.

This lower bound is more general than it may at first appear. For example, it allows insertions, which can be simulated by melds. However, the bound does not apply to situations in which some of the operations constrain the outcome of later ones, as for instance when we perform a *delete min* on a heap, reinsert the deleted item, and perform another *delete min.*

6. Remarks. The top-down skew heaps we have introduced in § 2 are simpler than leftist heaps and as efficient, to within a constant factor, on all the heap operations. By changing the data structure to allow bottom-up melding, we have reduced the amortized time of *insert* and *meld* to $O(1)$, thereby obtaining a data structure with optimal efficiency on any sequence of *meld* and *delete min* operations. Table 1 summarizes our complexity results.

TABLE 1
Amortized running times of skew heap operations.

	top-down skew heaps	bottom-up skew heaps
make heap	$O(1)$	$O(1)$
find min	$O(1)$	$O(1)$
insert	$O(\log n)$	$O(1)$
meld	$O(\log n)$	$O(1)$
delete min	$O(\log n)$	$O(\log n)$
delete	$O(\log n)$	$O(\log n)$

Several interesting open problems remain. On the practical side, there is the question of exactly what pointer structure and what implementation of the heap operations will give the best empirical behavior. On the theoretical side, there is the problem of extending the lower bound in § 5 to allow other combinations of operations, and of determining whether skew heaps or any other form of heaps are optimal in a more general setting. Two very recent results bear on this question. Fredman (private communication) has shown that the amortized bound of $O(k+k\log(n/k))$ we derived for k deletions followed by a *find min* is optimum for comparison-based algorithms. Fredman and Tarjan [6] have proposed a new kind of heap, called the *Fibonacci heap*, that has an amortized time bound of $O(\log n)$ for arbitrary deletion and $O(1)$ for *find min*, *insert*, *meld*, and the following operation, which we have not considered in this paper:

decrease(x, y, h): Replace item x in heap h by item y, known to be no greater than x.

The importance of Fibonacci heaps is that *decrease* is the dominant operation in many network optimization algorithms, and the use of Fibonacci heaps leads to improved time bounds for such algorithms [6]. The Fibonacci heap cannot properly be called a self-adjusting structure, because explicit balance information is stored in the nodes. This leads to the open problem of devising a self-adjusting heap implementa-

tion with the same amortized time bounds as Fibonacci heaps. Skew heaps do not solve this problem, because *decrease* (implemented as a deletion followed by an insertion or in any other obvious way) takes $\Omega(\log n)$ amortized time.

More generally, our results only scratch the surface of what is possible using the approach of studying the amortized complexity of self-adjusting data structures. We have also analyzed the amortized complexity of self-adjusting lists, and in particular the move-to-front heuristic, under various cost measures [12], and we have devised a form of self-adjusting search tree, the *splay tree*, which has a number of remarkable properties and applications [13]. The field is ripe for additional work.

Appendix.

Tree terminology. We consider binary trees as defined by Knuth [6]: every tree node has two children, a *left child* and a *right child*, either or both of which can be the special node null. If node y is a child of node x, then x is the *parent* of y, denoted by $p(y)$. The *root* of the tree is the unique node with no parent. If $x = p^i(y)$ for some $i \geq 0$, x is an *ancestor* of y and y is a *descendant* of x; if $i > 0$, x is the *proper ancestor* of y and y a *proper descendant* of x. The *right path* descending from a node x is the path obtained by starting at x and repeatedly proceeding to the right child of the current node until reaching a node with null right child; we define the *left path* descending from x similarly. The *right path* of a tree is the right path descending from its root; we define the *left path* similarly. A *path* from a node x to a *missing node* is a path from x to null, such that each succeeding node is a child of the previous one. The direction from parent to child is *downward* in the tree; from child to parent, *upward.*

REFERENCES

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

[2] M. R. BROWN, *Implementation and analysis of binomial queue algorithms*, this Journal, 7 (1978), pp. 298–319.

[3] D. CHERITON AND R. E. TARJAN, *Finding minimum spanning trees*, this Journal, 5 (1976), pp. 724–742.

[4] C. A. CRANE, *Linear lists and priority queues as balanced binary trees*, Technical Report STAN-Cs-72-259, Computer Science Dept, Stanford Univ., Stanford, CA, 1972.

[5] E. W. DIJKSTRA, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[6] M. L. FREDMAN AND R. E. TARJAN, *Fibonacci heaps and their uses in network optimization algorithms*, Proc. 25th Symposium on Foundations of Computer Science, 1984, pp. 338–346.

[7] D. E. KNUTH, *The Art of Computer Programming, Volume* 1: *Fundamental Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1973.

[8] ———, *The Art of Computer Programming, Volume* 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[9] D. D. SLEATOR, *An $O(nm \log n)$ algorithm for maximum network flow*, Technical Report STAN-CS-80-831, Computer Science Dept, Stanford Univ., Stanford, CA, 1980.

[10] D. D. SLEATOR AND R. E. TARJAN, *A data structure for dynamic trees*, J. Comp. System Sci., 26 (1983), pp. 362–391; also Proc. Thirteenth Annual ACM Symposium on Theory of Computing, 1981, pp. 114–122.

[11] ———, *Self-adjusting binary trees*, Proc. Fifteenth Annual ACM Symposium on Theory of Computing, 1983, pp. 235–246.

[12] ———, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[13] ———, *Self-adjusting binary search trees*, J. Assoc. Comput. Mach., to appear.

[14] R. E. TARJAN, *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comput. Mach., 22 (1975), pp. 215–225.

[15] ———, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comp. System Sci., 18 (1979), pp. 110–127.

[16] ———, *Data Structures and Network Algorithms*, CBMS Regional Conference Series in Applied Mathematics 44, Society for Industrial and Applied Mathematics, Philadelphia, 1983.

[17] J. VUILLEMIN, *A data structure for manipulating priority queues*, Comm. ACM, 21 (1978), pp. 309–314.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
005

THE COMPLEXITY OF LANGUAGES GENERATED BY ATTRIBUTE GRAMMARS*

JOOST ENGELFRIET†

Abstract. A string-valued attribute grammar (SAG) has a semantic domain of strings over some alphabet, with concatenation as basic operation. It is shown that the output language (i.e., the range of the translation) of a SAG is log-space reducible to a context-free language.

Key words. theory, compilers, attribute grammars, alternation, complexity, classes of languages defined by grammars, by resource-bounded automata

CR categories. D3.4, F1.2, F2, F4.3

Introduction. Attribute grammars (AG) are a mechanism for translating the derivation trees of a context-free grammar into values of some semantic domain [26]. A particular, very restricted, domain of interest is the set of strings over some alphabet with concatenation as basic operation. Such "string-valued" attribute grammars (SAG), see [10], are of importance for two reasons. The first reason is that every attribute grammar can be viewed as a SAG. In fact, viewing the right-hand sides of semantic rules of an AG as strings rather than meaningful expressions, a SAG is obtained whose attribute evaluation corresponds to the symbolic evaluation of the attributes of the AG: the SAG translates the derivation trees of the context-free grammar into strings (usually expressions) which eventually may be interpreted again as the values computed by the AG. The second reason that string-valued attribute grammars are important is that they can be used as a language generating mechanism: the range of the translation of a SAG G, denoted OUT (G), is a formal language. If G describes a compiler that translates high-level programs into assembler programs, then OUT (G) is the "assembler dialect spoken by the compiler," i.e., the set of all assembler programs that are the translation of some high-level program. For the more restricted formalism of top-down tree transducers, such "tree transformation languages" were first studied in [32].

In this paper we prove that for every SAG G, OUT (G) is in LOG (CF): the class of languages log-space reducible to context-free languages. Hence these output languages are recognizable in polynomial time or log square space (on a deterministic Turing machine). The proof of this result was obtained as a generalization of the proof of IO $\subseteq$ LOG (CF) in [2], where IO is the class of inside-out macro languages [19]. In fact, IO is equal to the class of all OUT (G), where G is a SAG with one synthesized attribute (and an arbitrary number of inherited attributes), see [10]. To show that a language L is in LOG (CF), one can use the technique introduced in [36] (and used in [2]) to accept L by a nondeterministic multihead pushdown automaton (MPDA) in polynomial time: the equivalence of polynomial time MPDA and LOG (CF) is shown in [36]. To show that OUT (G) is in LOG (CF) we will use a variation of this technique that concerns alternating multihead finite automata (AMFA) rather than MPDA. It is well-known that AMFA and MPDA accept the same class of languages (viz. PTIME, the class of deterministic polynomial time Turing machine languages,

* Received by the editors December 27, 1982, and in final revised form April 15, 1984.

† Department of Computer Science, Twente University of Technology, 7500 AE Enschede, the Netherlands. Present address, Department of Mathematics and Computer Science, University of Leiden, Leiden, the Netherlands.

see [7], [5]). It is shown in [35] that time of the MPDA is polynomially related to tree-size of the AMFA (where tree-size is the number of nodes of the accepting computation tree of the AMFA). Thus LOG (CF) is the class of languages accepted by AMFA in polynomial tree-size.

The simulation of some grammar-like formalism by a polynomial tree-size AMFA can be obtained by simulating any derivation tree of the grammar by an AMFA in such a way that its tree-size is polynomial in the size of the derivation tree and the length of the produced string, and then showing that every string produced by the grammar can actually be produced by a derivation tree of polynomial size (in the length of the string); see, e.g., [37], [2], [17] where this is done for polynomial time MPDA, simulating a left-most derivation of the grammar. In our case, we simulate the attribute evaluation of G on an arbitrary derivation tree by an AMFA with polynomial tree-size, and show that every string v in OUT (G) can be obtained from a derivation tree of size linear in the length of v. Since the computation tree of the AMFA branches in the same way as the derivation tree it is simulating, we obtain as a particular case the fact that if G is linear (i.e., the underlying context-free grammar is linear), then OUT (G) can be recognized by a nondeterministic multihead finite automaton, and so OUT $(G) \in$ NSPACE (log n), the class of nondeterministic log-space Turing machine languages. Since EDTOL is contained in this class of output languages, this generalizes the result from [25] that EDTOL $\subseteq$ NSPACE (log n); see also [38], [21].

The paper is organized as follows. In § 1 we establish some rather precise terminology for discussing attribute grammars, in particular SAG. In § 2 we show how, for an arbitrary AG G, OUT (G) can be recognized by a nondeterministic recursive program, and that for SAG this program can be implemented on an AMFA. This section already contains the above-mentioned result on linear SAG. In § 3 we prove that for a restricted class of SAG a linear bound can be put on the size of the derivation tree (linear in the length of the string into which the SAG translates the tree). Finally, in § 4, we show that every SAG is output-equivalent to such a restricted SAG. In § 5 we give the main theorem and discuss some of its consequences.

1. Preliminaries. In this section we provide the reader with the unavoidable heap of terminology needed to discuss attribute grammars. For an example of an AG see § 1.5 and Fig. 1. The last subsection contains terminology on complexity classes.

The empty string is denoted λ; $|w|$ is the length of string w. We do not distinguish between languages L and $L-\{\lambda\}$.

1.1. Context-free grammars. The class of context-free languages is denoted CF. A context-free grammar (CFG) $G=(N, T, P, Z)$ consists of nonterminals, terminals, productions, and the initial nonterminal, respectively. A derivation tree t has leaves labeled by elements of $T \cup \{\lambda\}$; if the root of t is labeled Z, then t is a *complete* derivation tree; if some leaves of t are labeled by a nonterminal, then t is a *partial* derivation tree. A node of t is a *nonterminal node* if it is labeled by a nonterminal. The *nonterminal size* of t is the number of its nonterminal nodes. The root of t is denoted by root (t), and its yield by yield (t).

A production is usually written as $X_0 \to w_1 X_1 w_2 \cdots w_k X_k w_{k+1}$ with $X_j \in N$, $w_j \in T^*$, $k \geqq 0$. A production is *final* if $k=0$ (i.e., it is $X \to w$ with $X \in N$ and $w \in T^*$), and it is *linear* if $k=1$ (i.e., of the form $X \to w_1 Y w_2$ with X, $Y \in N$ and $w_1 w_2 \in T^*$). G is *linear* if all its productions are linear or final.

Let t be a (partial) derivation tree. An *occurrence of a production* $p: X_0 \to w_1 X_1 \cdots w_k X_k w_{k+1}$ in t consists of a node x_0 of t, labeled X_0, and all its sons, such that the labels of the sons, from left to right, form the string $w_1 X_1 \cdots w_k X_k w_{k+1}$. We

say that p is applied at x_0. We will always denote by $x_1, \cdots, x_k$ the sons of x_0 labeled $X_1, \cdots, X_k$ (in their order from left to right); thus $x_0, x_1, \cdots, x_k$ are the nonterminal nodes of the occurrence of p. We always assume that G is *reduced* in the usual sense that every production occurs in some complete derivation tree.

1.2. Attribute grammars. An attribute grammar (AG) G is described in the following 4 points, see [26].

(1) G has a *semantic domain* (V, Φ) where V is a set (of values) and Φ is a collection of functions of type $V^m \to V$, $m \geqq 0$.

(2) G has a *context-free grammar* (N, T, P, Z) such that Z does not occur in the right-hand side of any production.

(3) Each nonterminal X has a finite set $\mathrm{ATT}(X)$ of *attributes*. $\mathrm{ATT}(X)$ is the union of two disjoint sets $\mathrm{IN\text{-}ATT}(X)$ and $\mathrm{SYN\text{-}ATT}(X)$ of inherited and synthesized attributes, respectively. For the initial nonterminal Z, $\mathrm{IN\text{-}ATT}(Z) = \varnothing$ and $\mathrm{SYN\text{-}ATT}(Z) = \{d\}$; d is called the *designated attribute* (the value of d will be the meaning of the derivation tree).

(4) For a production $p: X_0 \to w_1 X_1 \cdots w_k X_k w_{k+1}$, a pair $\langle a, j\rangle$ such that $a \in \mathrm{ATT}(X_j)$, $0 \leqq j \leqq k$, is called an *attribute of production p*. The set of all attributes of p is denoted $\mathrm{ATT}(p)$. Each production p has an associated set $\mathrm{RULES}(p)$ of *semantic rules*. A semantic rule of p is specified by a function $f \in \Phi$, say of type $V^m \to V$ $(m \geqq 0)$, and a sequence of $m+1$ (not necessarily distinct) attributes of $p: \langle\langle a_0, j_0\rangle, \langle a_1, j_1\rangle, \cdots, \langle a_m, j_m\rangle\rangle$. Such a semantic rule is denoted $\langle a_0, j_0\rangle := f(\langle a_1, j_1\rangle, \cdots, \langle a_m, j_m\rangle)$ and we say that it defines $\langle a_0, j_0\rangle$ using $\langle a_i, j_i\rangle$, $1 \leqq i \leqq m$. $\mathrm{RULES}(p)$ contains one semantic rule defining each attribute $\langle a, 0\rangle$ with $a \in \mathrm{SYN\text{-}ATT}(X_0)$, and one semantic rule defining each attribute $\langle a, j\rangle$ with $j \geqq 1$ and $a \in \mathrm{IN\text{-}ATT}(X_j)$, and no other semantic rules. Moreover we assume, without loss of generality, that G is in Bochmann Normal Form, i.e., that no attribute defined by a semantic rule is also used in a semantic rule.

This ends the definition of attribute grammar. An attribute grammar is *linear* (linAG) if its context-free grammar is linear.

1.3. Dependency graphs. We define dependency graphs as in [26], except that, for the dependency graph of a derivation tree, we label each node with a (modified) right-hand side of a semantic rule. Let G be an AG as described in 1.2.

The *dependency graph of a production p* is the directed graph of which the nodes are the attributes of p and which has an edge from $\langle a, i\rangle$ to $\langle b, j\rangle$ iff there is a semantic rule in $\mathrm{RULES}(p)$ which defines $\langle b, j\rangle$ using $\langle a, i\rangle$.

Let t be a (partial) derivation tree of G. Let x be a nonterminal node of t labeled X. An *attribute of node x* is a pair $\langle a, x\rangle$ with $a \in \mathrm{ATT}(X)$. The set $\mathrm{ATT}(t)$ of *attributes of t* is $\{\langle a, x\rangle | x$ is a nonterminal node of t, $\langle a, x\rangle$ is an attribute of $x\}$.

The dependency graph of t is obtained by combining all dependency graphs of all (occurrences of) productions in t. The *dependency graph* $D(t)$ *of a (partial) derivation tree t* is the labeled directed graph whose nodes are the attributes of t (i.e., $\mathrm{ATT}(t)$) and whose edges and labels of nodes are determined as follows. Consider an occurrence of a production $p: X_0 \to w_1 X_1 \cdots w_k X_k w_{k+1}$ in t with nonterminal nodes $x_0, x_1, \cdots, x_k$. For every semantic rule r: $\langle a_0, j_0\rangle := f(\langle a_1, j_1\rangle, \cdots, \langle a_m, j_m\rangle)$ in $\mathrm{RULES}(p)$ there is an edge from $\langle a_i, x_{j_i}\rangle$ to $\langle a_0, x_{j_0}\rangle$ for every i $(1 \leqq i \leqq m)$, and $\langle a_0, x_{j_0}\rangle$ is labeled with $f(\langle a_1, x_{j_1}\rangle, \cdots, \langle a_m, x_{j_m}\rangle)$, i.e., with the function f and the sequence $\langle\langle a_1, x_{j_1}\rangle, \cdots, \langle a_m, x_{j_m}\rangle\rangle$. We also say that semantic rule r *defines* attribute $\langle a_0, x_{j_0}\rangle$ of t. It is easy to see that each attribute of t is defined by at most one semantic rule, and thus, if t is complete, each node gets a unique label (if t is not complete, some nodes do not have

a label). Note that each labeled node n of $D(t)$ is labeled by $f(n_1, \cdots, n_m)$ with $f \in \Phi$ and $n_1, \cdots, n_m$ are all nodes of $D(t)$ from which at least one edge runs to n.

An AG is *noncircular* if no dependency graph of a complete derivation tree contains an oriented cycle. We assume throughout this paper that AG are noncircular, see [26].

Note that the dependency graph $D(t)$ of some complete derivation tree t contains all information needed to compute the value of the designated attribute of the root of t. An AG is *reduced* if for every complete derivation tree t and every node $\langle a, x\rangle$ of $D(t)$ there is an oriented path from $\langle a, x\rangle$ to $\langle d, \text{root}\,(t)\rangle$. This means that every attribute of t is needed to compute the meaning of t. It is shown in [18] that for every AG there is an equivalent reduced AG.

Let t be a derivation tree with root labeled X. The *i/s-graph* $is(t)$ of t is the directed graph with the attributes of X as nodes and with an edge from a to b if $a \in \text{IN-ATT}\,(X)$, $b \in \text{SYN-ATT}\,(X)$, and there is an oriented path from $\langle a, \text{root}\,(t)\rangle$ to $\langle b, \text{root}\,(t)\rangle$ in $D(t)$. The set of all i/s-graphs for nonterminal X is defined by $is(X) = \{is(t) |\ t$ is a derivation tree with root labeled $X\}$. It is shown in [26] that these sets are computable.

1.4. Assignments and output set. We define assignments of values to attributes for nonterminals, productions, and derivation trees. This determines the semantics of attribute grammars. Let G again be an AG as described in § 1.2 (with semantic domain (V, Φ)).

An *assignment for nonterminal* X is a mapping $\text{ATT}\,(X) \to V$.

An *assignment for production* $p: X_0 \to w_1 X_1 \cdots w_k X_k w_{k+1}$ is a mapping $\text{ATT}\,(p) \to V$. If $\beta_j : \text{ATT}\,(X_j) \to V$ is an assignment for X_j, $0 \leqq j \leqq k$, then we denote by $\langle \beta_0, \beta_1, \cdots, \beta_k\rangle$ the assignment $\alpha : \text{ATT}\,(p) \to V$ for p such that $\alpha(\langle a, j\rangle) = \beta_j(a)$. Clearly every assignment for p is of the form $\langle \beta_0, \beta_1, \cdots, \beta_k\rangle$ for some β_j, $0 \leqq j \leqq k$. An assignment α for p is *correct* if, for every semantic rule $\langle a_0, j_0\rangle := f(\langle a_1, j_1\rangle, \cdots, \langle a_m, j_m\rangle)$ in RULES (p), $\alpha(\langle a_0, j_0\rangle) = f(\alpha(\langle a_1, j_1\rangle), \cdots, \alpha(\langle a_m, j_m\rangle))$.

An *assignment for derivation tree* t is a mapping $\alpha : \text{ATT}\,(t) \to V$. If c is an occurrence in t of a production $p: X_0 \to w_1 X_1 \cdots w_k X_k w_{k+1}$ with nonterminal nodes $x_0, x_1, \cdots, x_k$, then the assignment α_c for p is defined by $\alpha_c(\langle a, j\rangle) = \alpha(\langle a, x_j\rangle)$. An assignment α for t is *correct* if α_c is correct for p, for every occurrence c of p in t (for all $p \in P$). Thus, a correct assignment for t is an assignment of values to the attributes of t such that the semantic rules are satisfied. It is easy to prove that a correct assignment α for t is a mapping from the nodes of $D(t)$ to V such that if node n of $D(t)$ is labeled $f(n_1, \cdots, n_m)$, then $\alpha(n) = f(\alpha(n_1), \cdots, \alpha(n_m))$. It is now not difficult to see that for noncircular AG every complete derivation tree t has exactly one correct assignment α; for each attribute $\langle a, x\rangle$ of t, $\alpha(\langle a, x\rangle)$ is called the *value* of $\langle a, x\rangle$; the *meaning of* t is the value of $\langle d, \text{root}\,(t)\rangle$. Thus a noncircular AG translates every complete derivation tree t into its meaning $\alpha(\langle d, \text{root}\,(t)\rangle)$. In this paper we are interested in the range of this translation.

For an AG G, the *output set* of G, denoted by $\text{OUT}\,(G)$, is the subset of V defined by $\text{OUT}\,(G) = \{\alpha(\langle d, \text{root}\,(t)\rangle) | t$ is complete derivation tree with correct assignment $\alpha\}$. Two AG G_1 and G_2, are *output-equivalent* if $\text{OUT}\,(G_1) = \text{OUT}\,(G_2)$.

1.5. String-valued attribute grammars. An AG is string-valued if the values of its attributes are strings and the only operation on strings is concatenation, see [10].

A *string-valued attribute grammar* (SAG) is an AG with semantic domain (Σ^*, Φ) for some alphabet Σ (called the *output alphabet*), where Φ consists of all derived functions of the free monoid Σ^* generated by the elements of Σ. Formally, this means that for every string $v \in (\Sigma \cup \{y_1, \cdots, y_m\})^*$, $m \geqq 0$, where $y_1, \cdots, y_m$ are new symbols,

the function f_v is in Φ, where $f_v(w_1, \cdots, w_m)$ is the result of substituting w_i for y_i throughout v ($1 \leqq i \leqq m$). Informally, this means that we specify our semantic rules by $\langle a_0, j_0\rangle := u_1\langle a_1, j_1\rangle u_2\langle a_2, j_2\rangle \cdots u_m\langle a_m, j_m\rangle u_{m+1}$ with $u_i \in \Sigma^*$, $1 \leqq i \leqq m+1$. Thus the right-hand side of a semantic rule in RULES (p) is any element of $(\Sigma \cup \mathrm{ATT}(p))^*$. A correct assignment $\alpha : \mathrm{ATT}(p) \to \Sigma^*$ for production p should satisfy the following: if $\alpha(\langle a_i, j_i\rangle) = w_i$ for $1 \leqq i \leqq m$, then $\alpha(\langle a_0, j_0\rangle) = u_1 w_1 u_2 w_2 \cdots u_m w_m u_{m+1}$. The label of $\langle a_0, x_{j_0}\rangle$ in $D(t)$, defined by this semantic rule, is $u_1\langle a_1, x_{j_1}\rangle u_2 \cdots u_m\langle a_m, x_{j_m}\rangle u_{m+1}$. The output set OUT ($G$) of a SAG G is a language over Σ: the *output language* of G. We define the classes of output languages OUT (SAG) = {OUT (G)|G is a SAG} and OUT (linSAG) = {OUT (G)|G is a linear SAG}.

We mention the following property of SAG. Let t be a derivation tree of a SAG G. If there is an oriented path from $\langle a_1, x_1\rangle$ to $\langle a_2, x_2\rangle$ in $D(t)$ and α is a correct assignment for t, then $\alpha(\langle a_1, x_1\rangle)$ is a substring of $\alpha(\langle a_2, x_2\rangle)$. This can easily be proved by induction on the length of the path, using the special form of the semantic rules (see also in § 1.4 what it means for $D(t)$ that α is correct). Thus the value of each attribute needed to compute the meaning v of t occurs as a substring in v. In particular, for a reduced SAG *every* attribute value is a substring of v.

Example of a SAG. Consider the following SAG G with output alphabet $\Sigma = \{a, b, \$\}$ and with CFG (N, T, P, Z) where $N = \{Z, A\}$, $T = \{a, b\}$, ATT (Z) = $\{d\}$, IN-ATT (A) = $\{i\}$ and SYN-ATT (A) = $\{s\}$. The productions of P with their semantic rules are as follows:

p	RULES (p)
$Z \to aA$	$\langle d, 0\rangle := \langle s, 1\rangle, \langle i, 1\rangle := a$;
$Z \to bA$	$\langle d, 0\rangle := \langle s, 1\rangle, \langle i, 1\rangle := b$;
$A \to aA$	$\langle s, 0\rangle := \langle s, 1\rangle\langle s, 1\rangle, \langle i, 1\rangle := \langle i, 0\rangle a$;
$A \to bA$	$\langle s, 0\rangle := \langle s, 1\rangle\langle s, 1\rangle, \langle i, 1\rangle := \langle i, 0\rangle b$;
$A \to \lambda$	$\langle s, 0\rangle := \langle i, 0\rangle\$\langle i, 0\rangle\$$.

The meaning of the derivation tree t in Fig. 1 is $ab\$ab\$ab\$ab\$$, as can easily be seen from its (labeled) dependency graph. It is easy to see that OUT (G) = $\{(w\$)^n | n = 2^{|w|}, w \in \{a, b\}^*, w \neq \lambda\}$. Note that G is linear and reduced.

1.6. Complexity classes. As usual (see [22] and [5]) we denote by DSPACE ($f(n)$), NSPACE ($f(n)$), and ASPACE ($f(n)$) the classes of languages accepted by deterministic, nondeterministic, and alternating (respectively) Turing machines in space $f(n)$. The definitions for TIME are analogous; also PTIME = $\cup$ {DTIME (n^c)|$c \geqq 1$} and NPTIME = $\cup$ {NTIME (n^c)|$c \geqq 1$}. LOG (CF) is the class of languages reducible to context-free languages in log-space, see [36].

A multihead finite automaton (MFA) is, as usual, a nondeterministic automaton with a two-way read-only input tape on which the input string is written, surrounded by endmarkers. The automaton has a finite number of read-heads, each of which can move in both directions on the input tape; the automaton is able to see whether two heads are on the same square of the input tape. Let MFA also denote the class of languages accepted by these automata; it is well-known that MFA = NSPACE (log n).

An alternating multihead finite automaton (AMFA), see [5], is like an MFA but its states are partitioned into existential and universal states. A computation tree (see [29], [35]) of an AMFA M is a finite tree of which the nodes are labeled by configurations of M such that the descendants of any internal node labeled by an existential (universal) configuration consist of one (all, respectively) successor(s) of that configur-

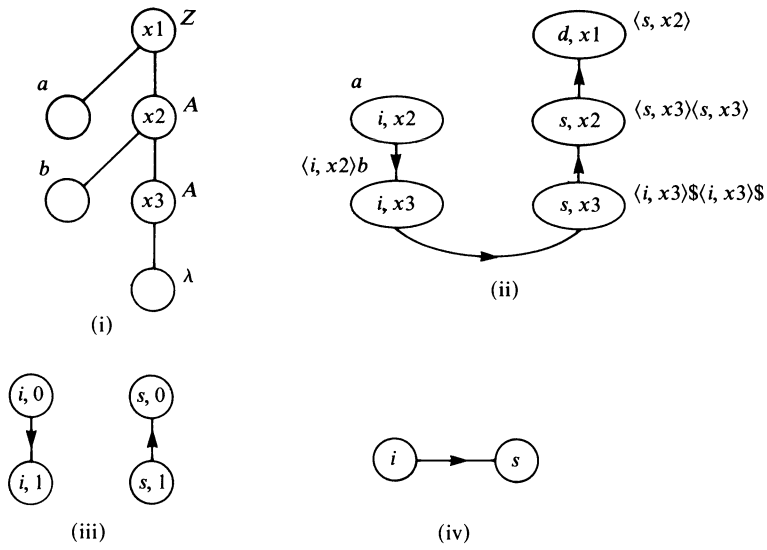


FIG. 1. (i) *Complete derivation tree t; its nonterminal nodes are* $x1$, $x2$, *and* $x3$. (ii) *Dependency graph* $D(t)$ *of t, labeled with strings over* $\Sigma \cup \mathrm{ATT}(t)$. (iii) *Dependency graph of production* $A \to aA$ (*and* $A \to bA$). (iv) i/s*-graph is*(t_1), *where* t_1 *is any of the subtrees of t with root A.*

ation. A computation tree is accepting if all its leaves are labeled by accepting configurations. M accepts a string v if there is an accepting computation tree with the initial configuration for v at the root. If the size (i.e., the number of nodes) of the computation tree is k, then v is said to be accepted within *tree-size* bound k [35]. Let AMFA also denote the class of languages accepted by such devices, and let P-AMFA denote the class of languages accepted by AMFA in polynomial tree-size. Note that an MFA is an AMFA with no universal states; the tree-size is the length of the computation, i.e., time.

It is well known that $\mathrm{AMFA} = \mathrm{ASPACE}(\log n) = \mathrm{PTIME}$ (see [5], [7]) and $\mathrm{P\text{-}AMFA} = \mathrm{LOG}(\mathrm{CF})$, see [35], [36], cf. the introduction. Since MFA clearly work in polynomial time, $\mathrm{MFA} \subseteq \mathrm{P\text{-}AMFA} \subseteq \mathrm{AMFA}$, i.e., $\mathrm{NSPACE}(\log n) \subseteq \mathrm{LOG}(\mathrm{CF}) \subseteq \mathrm{PTIME}$. We finally note that, since $\mathrm{CF} \subseteq \mathrm{DSPACE}(\log^2 n)$, see [30], also $\mathrm{LOG}(\mathrm{CF}) \subseteq \mathrm{DSPACE}(\log^2 n)$. In [35] it is even shown that $\mathrm{LOG}(\mathrm{CF}) \subseteq \mathrm{ATIME}(\log^2 n)$; note that $\mathrm{ATIME}(\log^2 n) \subseteq \mathrm{DSPACE}(\log^2 n)$, see [5].

2. Recognizing the output set. Let G be an AG with semantic domain (V, Φ) and CFG (N, T, P, Z). Let us consider the problem of finding out whether a given value $v \in V$ belongs to the output set $\mathrm{OUT}(G)$ of G. Clearly, it suffices to find a complete derivation tree t and a correct assignment α (of values to the attributes of t) such that $\alpha(\langle d, \mathrm{root}(t)\rangle) = v$, where d is the designated attribute. To do this we can use recursion to construct nondeterministically a derivation tree of the CFG, guessing simultaneously the values of the attributes of the nodes constructed so far (except of course the value of $\langle d, \mathrm{root}(t)\rangle$ which is taken to be v) and checking that this partial assignment is correct (in the sense that the assignments for each occurrence of a production p in the partial derivation tree are all correct for p, cf. § 1.4). If the computation is successful, i.e., the guessed assignment of the complete derivation tree t is correct, then $v \in \mathrm{OUT}(G)$, because t has only one correct assignment.

Note that this would also work for circular AG, if for such an AG we put $\alpha(\langle d, \mathrm{root}(t)\rangle)$ in $\mathrm{OUT}(G)$ for *every* correct assignment α of every complete derivation

tree t. For the reader familiar with affix grammars ([28], [39]) and their correspondence to attribute grammars, we note that in fact the above recursive algorithm simulates a derivation of the affix grammar corresponding to the AG. A similar notion of derivation can also be defined for (extended) attribute grammars, see [31].

We now write down the algorithm in some more detail in the form of an Algol-60-like program consisting of a call of a recursive boolean procedure "test", which has two (value) parameters: a nonterminal X and an assignment α for X. In the body of the procedure a local variable β is used of type "array of assignments for nonterminals" (of length maxrhs, i.e., the maximal number of occurrences of nonterminals in the right-hand side of a production in P). For a given $v \in V$, we denote by α_v the assignment for Z such that $\alpha_v(d) = v$.

The program that tests whether $v \in \mathrm{OUT}(G)$ is as follows.

```
begin
  boolean proc test (X, α); nonterminal X; assignment α;
    begin assignment array β [1: maxrhs];
          boolean result; integer i;
      guess a production p: X → w1 X1 ··· wk Xk wk+1 in P;
1.    for i := 1 to k do β[i] := any assignment for Xi;
2.    if ⟨α, β[1], ··· , β[k]⟩ is correct for p
3.      then result := test (X1, β[1]) and ··· and test (Xk, β[k])
        else result := false;
      test := result
    end;
  {main program}
  if test (Z, αv) then accept
end.
```

Note that $\langle \alpha, \beta[1], \cdots, \beta[k] \rangle$ is an assignment for p, see § 1.4. Line 3 should of course be replaced by: result := true; **for** $i := 1$ **to** k **do** result := result **and** test $(X_i, \beta[i])$. Note that, in case $k = 0$, test just checks whether $\langle \alpha \rangle$ is correct for p.

It is not difficult to see that this program indeed recognizes $\mathrm{OUT}(G)$. In fact, a call test (X, α) has a computation that returns the value true if and only if there exist a derivation tree t with root labeled X and a correct assignment γ for t, such that $\gamma(\langle a, \mathrm{root}(t) \rangle) = \alpha(a)$ for every $a \in \mathrm{ATT}(X)$. This can be shown by induction on the recursion depth of the procedure and the height of the derivation tree.

We now show that for a reduced string-valued AG the above program can be implemented on an alternating multihead finite automaton (AMFA), and we estimate the tree-size used by the automaton (see § 1.6). Let G be a reduced (see § 1.3) and string-valued (see § 1.5) AG with output alphabet Σ. Then the program above tests whether $v \in \Sigma^*$ is an element of the output language $\mathrm{OUT}(G) \subseteq \Sigma^*$; v is now on the input tape of the AMFA, between endmarkers. Since G is reduced, the program only has to consider assignments $\alpha : \mathrm{ATT}(X) \to \Sigma^*$ such that $\alpha(a)$ is a substring of v for every $a \in \mathrm{ATT}(X)$, see § 1.5. The parameter α and each variable $\beta[i]$ of type "assignment" is simulated by $2n$ heads, where n is the maximal number of attributes of a nonterminal. The heads may be given names $\langle \gamma, a, q \rangle$ where γ is α or $\beta[i]$, a is an attribute of a nonterminal, and $q \in \{\text{left}, \text{right}\}$. The string $\gamma(a)$ is recorded by putting the $\langle \gamma, a, \text{left} \rangle$-head and the $\langle \gamma, a, \text{right} \rangle$-head on the first and last (respectively) symbol of an occurrence of $\gamma(a)$ in v. The other information in the program (such as the parameter X and the guessed production p) can be kept in the finite control of the automaton. Finally, the automaton has two additional work-heads.

The program can be implemented on the AMFA as follows. Initially, the automaton has all its read-heads at the left end marker, and then moves its $\langle \alpha, d, \text{left}\rangle$-head one square to the right and its $\langle \alpha, d, \text{right}\rangle$-head to the square just before the right endmarker (to obtain the initial assignment α_v). To simulate "$\beta[i] :=$ *any* assignment" in line 1, for each attribute a, the two $\langle \beta[i], a, q\rangle$-heads are moved nondeterministically, i.e., using existential branching, to some input square and then the $\langle \beta[i], a, \text{right}\rangle$-head is moved some more squares to the right. To check, in line 2, the correctness of the assignment $\gamma = \langle \alpha, \beta[1], \cdots, \beta[k]\rangle$ for p, the automaton checks that $\gamma(\langle a_0, j_0\rangle) = u_1\gamma(\langle a_1, j_1\rangle)u_2 \cdots u_m\gamma(\langle a_m, j_m\rangle)u_{m+1}$ for every semantic rule $\langle a_0, j_0\rangle := u_1\langle a_1, j_1\rangle u_2 \cdots u_m\langle a_m, j_m\rangle u_{m+1}$ in RULES(p). This can be done as follows; rename $\gamma = \langle \gamma_0, \gamma_1, \cdots, \gamma_k\rangle$. Let work-head 1 move from the $\langle \gamma_{j_0}, a_0, \text{left}\rangle$-head to the $\langle \gamma_{j_0}, a_0, \text{right}\rangle$-head checking that this substring equals $u_1\gamma(\langle a_1, j_1\rangle) \cdots \gamma(\langle a_m, j_m\rangle)u_{m+1}$. Each u_i is checked using the finite control. Each $\gamma(\langle a_i, j_i\rangle)$ is checked by moving work-head 2 simultaneously from the $\langle \gamma_{j_i}, a_i, \text{left}\rangle$-head to the $\langle \gamma_{j_i}, a_i, \text{right}\rangle$-head. Finally, line 3 is implemented by a k-fold universal branching; in the ith branch, the automaton first moves each $\langle \alpha, a, q\rangle$-head to the same position as the corresponding $\langle \beta[i], a, q\rangle$-head.

This shows how the program can be implemented on an AMFA. We now determine the tree-size used, i.e., the size of the computation tree, in terms of $|v|$, the length of the input string of the AMFA. It should be clear from the above description that each line of the program can be executed in linear time, i.e., in $O(|v|)$ steps (where the constant depends, of course, on the given SAG G). Thus, one execution of the body of test (without the recursive calls) has a (monadic) computation tree of size $O(|v|)$. Furthermore, if the program "constructs" a complete derivation tree t, then test is called s times, where s is the nonterminal size of t (i.e., the number of nodes labeled by a nonterminal). Thus the total tree-size is $O(|v| \cdot s)$. We state this as a lemma.

LEMMA 2.1. *Let G be a reduced* SAG. OUT(G) *is accepted by an* AMFA, *such that $v \in$* OUT(G) *can be accepted in tree-size $O(|v| \cdot s)$, where s is the nonterminal size of a complete derivation tree with meaning v.*

Since AMFA = PTIME (see § 1.6), this already shows that OUT (SAG) $\subseteq$ PTIME (note that for every SAG there is an output-equivalent reduced SAG [18]). It remains to show that s can be taken polynomial in $|v|$; to obtain this, the grammar G has to be transformed (see the next sections).

We conclude this section by considering linear SAG (i.e., the underlying CFG is linear). For a reduced linear SAG the program can be implemented on an ordinary multihead finite automaton (MFA), because in line 3 no "*and*" occurs (i.e., no universal branching is needed), and the (final) recursive call of test can easily be replaced by iteration. Since for each linear SAG there is an output-equivalent reduced linear SAG [18], this proves the following result.

THEOREM 2.2. OUT (linSAG) $\subseteq$ NSPACE (log n).

3. Bounding the size of the derivation tree. To show that OUT (SAG) $\subseteq$ LOG (CF) = P-AMFA it now suffices to show that, in Lemma 2.1, s can be taken polynomial in $|v|$. In this section we show that for certain SAG that "do not make superfluous computations," we can take $s \leqq 6|v|$, i.e., the nonterminal size of a derivation tree is linear in the length of its meaning. To prove this we need a lemma on dependency graphs of derivation trees of SAG, which we will put in the more general framework of string-computing DAGs. Thus, the next subsection does not depend on the previous two sections.

3.1. String-computing DAGs. A DAG is a finite directed acyclic graph. For a DAG one can use the same terminology as for trees. Thus, if there is an edge from node m_1 to node m_2 we say that m_1 is a son of m_2, and m_2 a father of m_1. A root is a node without father and a DAG is *rooted* if it has only one root.

To prove (or define) something for the nodes of a DAG one can use induction on its structure as for trees, i.e., when proving (or defining) it for a node one may assume that it has already been proved (or defined) for the sons of the node.

DAGs are often used to represent computations; here we consider in particular the computation of strings. A *string-computing* DAG is a DAG G of which the nodes are labeled as follows. Let Σ be an alphabet. Each node m of G is labeled with a string label $(m) \in (\Sigma \cup \text{son}(m))^*$, where son (m) is the set of sons of m, such that each son of m occurs at least once in label (m).

For a node m of G, the *result* of m is defined inductively to be the string in Σ^* obtained by replacing, in label (m), each son n of m by the result of n. The *result* of a rooted string-computing DAG is the result of its root.

(It should be clear that the dependency graph of a derivation tree t of a noncircular SAG G is a string-computing DAG. Moreover, the result of each node $\langle a, x\rangle$ is $\alpha(\langle a, x\rangle)$, where α is the correct assignment for t.)

A string-computing DAG is *λ-free* if no node is labeled λ. A node m of a string-computing DAG is *passing* if it has exactly one son n and label $(m) = n$ (i.e., the result of n is passed to m, without computation). We now show that the (nonpassing) size of a DAG can be bounded by its result.

LEMMA 3.1. *Let v be the result of a rooted λ-free string-computing* DAG *G. Let k be the number of nonpassing nodes of G. Then* $2|v| \geqq k+1$.

Proof. Unrolling the DAG G into a tree, we can use the following well-known fact on trees.

Fact. For an arbitrary tree t, $2y(t) \geqq x(t)+1$, where $y(t)$ is the number of leaves of t, and $x(t)$ is the number of nonmonadic nodes of t (i.e., nodes with degree $\neq 1$). In particular, if t is a binary tree (i.e., all nodes have degree 2 or 0), then $2y(t) = x(t)+1$, see [27, p. 399].

We now define the unrolling of G in a formal way. Let $\tau[t_1 \cdots t_n]$ denote a tree with root labeled τ and direct subtrees $t_1, \cdots, t_n$ (a tree consisting of one node labeled τ is denoted τ), and let δ be an arbitrary symbol not in Σ (where Σ is the alphabet of G). For each node m of G we define a labeled tree $t(m)$ inductively as follows:

(a) If label $(m) = a \in \Sigma$, then $t(m) = a$.

(b) If label $(m) = u \in \Sigma^*$ and $|u| \geqq 2$, then $t(m) = \delta[u]$.

(c) If label $(m) = u_1 n_1 u_2 \cdots u_s n_s u_{s+1}$ with $u_i \in \Sigma^*$ and $n_i \in \text{son}(m)$, then $t(m) = \delta[u_1 t_1 u_2 \cdots u_s t_s u_{s+1}]$, where $t_i = t(n_i)$.

Finally, $t(G) = t(m_0)$ where m_0 is the root of G; $t(G)$ is the unrolling of G. Note that in both (b) and (c), δ has |label (m)| sons, i.e., each symbol from Σ in u and u_i counts as a direct subtree (of one node).

It is easy to show, by induction, that v is the yield of $t(G)$. Thus $y(t(G)) = |v|$. It should be clear that every node m of G corresponds to one or more nodes of $t(G)$: the root of $t(m)$ occurs at least once in $t(G)$; here we use the fact that every son of m occurs in label (m), for every m. Furthermore, as can be seen from (a), (b), and (c), a nonpassing node of G corresponds to nonmonadic nodes of $t(G)$. Hence $k \leqq x(t(G))$. Thus, using the Fact, $k+1 \leqq x(t(G))+1 \leqq 2y(t(G)) = 2|v|$.

Note that the Fact is actually a special case of this lemma. For a tree t, consider the following labeling of t: if m is a leaf then label $(m) = e$ (where e is some fixed

symbol), and otherwise label $(m) = n_1 \cdots n_s$ (where $n_1, \cdots, n_s$ are the sons of m). Clearly, for the resulting string-computing DAG, $|v| = y(t)$ and $k = x(t)$.

We finally note that this lemma can also be proved directly (without use of the Fact), by induction. □

3.2. Linear size of derivation tree. Let us be more precise about what we mean by a SAG G which "does not make superfluous computations" (cf. the beginning of this section). First of all, G should not compute attribute values which are not needed for the meaning of the derivation tree, i.e., G should be reduced (see § 1.3). Second, G should not spend its time concatenating the empty string with itself, thus computing λ only. We prevent this by requiring that G is *λ-free*, i.e., it has no semantic rules of the form $\langle a, j\rangle := \lambda$. Third, G should not have too many "passing" semantic rules. A *passing semantic rule* is a semantic rule of the form $\langle a_0, j_0\rangle := \langle a_1, j_1\rangle$. We restrict the number of passing semantic rules as follows. A production $p \in P$ of an AG with underlying CFG (N, T, P, Z) is called a *passing production* if all its semantic rules (i.e., elements of RULES (p)) are passing. We now require that G has no passing final productions and no passing linear productions (cf. § 1.1), i.e., every passing production should have at least two occurrences of a nonterminal in its right-hand side. We now show that for a SAG satisfying all these restrictions the nonterminal size of a derivation tree is linearly bounded by the length of its meaning.

LEMMA 3.2. *Let G be a λ-free reduced* SAG *that has no passing final productions and no passing linear productions. If t is a complete derivation tree of G with nonterminal size s and meaning v, then $s \leqq 6|v|$.*

Proof. Let p, l, and f be the number of occurrences of nonpassing, nonlinear, and final productions in t, respectively. Note that these properties are not independent; in particular, every final production is nonlinear.

The following four inequalities suffice to show the result.

(a) $s \leqq p + l$, because G has no passing linear productions.

(b) $f \leqq p$, because G has no passing final productions.

(c) $2f \geqq l + 1$. Let t_1 be the result of deleting all leaves from t. Then, clearly, the number of leaves of t_1 is f, and the number of nonmonadic nodes of t_1 is l. Hence $2f \geqq l + 1$ by the Fact mentioned in the proof of Lemma 3.1.

(d) $2|v| \geqq p + 1$, by Lemma 3.1. In fact, the dependency graph $D(t)$ is a string-computing DAG and v is the result of the node $\langle d, \text{root}\,(t)\rangle$ of $D(t)$. Since G is reduced and λ-free, $D(t)$ is rooted (with root $\langle d, \text{root}\,(t)\rangle$) and λ-free, and v is its result. A node of $D(t)$, i.e., an attribute of t, is passing iff it is defined by a passing semantic rule. Since there are p occurrences of nonpassing productions in t, at least p nodes of $D(t)$ are defined by a nonpassing semantic rule. Hence the number k of nonpassing nodes of $D(t)$ *is* $\geqq p$. Thus $p + 1 \leqq k + 1 \leqq 2|v|$.

From inequalities (a)-(d) we now conclude that

$$\begin{aligned} s &\leqq p + l && \text{by (a)} \\ &< p + 2f && \text{by (c)} \\ &\leqq 3p && \text{by (b)} \\ &< 6|v| && \text{by (d).} \end{aligned}$$

Note that in this proof we have used the Fact (of the proof of Lemma 3.1) twice. In (c) it was applied to t (without leaves), and in (d) to the unrolled $D(t)$, cf. the proof of Lemma 3.1. □

COROLLARY 3.3. *If G is a λ-free reduced* SAG *with no passing final and no passing linear productions, then* $\mathrm{OUT}(G) \in \mathrm{LOG}(\mathrm{CF})$.

Proof. From Lemmas 2.1 and 3.2 it follows that $\mathrm{OUT}(G)$ can be accepted by an AMFA in tree-size $O(n^2)$, where n is the length of the input. Thus $\mathrm{OUT}(G) \in$ P-AMFA $= \mathrm{LOG}(\mathrm{CF})$, cf. § 1.6. □

4. Removing superfluous computations. To prove that $\mathrm{OUT}(\mathrm{SAG}) \subseteq \mathrm{LOG}(\mathrm{CF})$ it now suffices, by Corollary 3.3, to show the following lemma.

LEMMA 4.1. *For every* SAG G *there is an output-equivalent* SAG G' *such that* (1) G' *is λ-free,* (2) G' *is reduced,* (3) G' *has no passing final productions, and* (4) G' *has no passing linear productions.*

The present section is devoted to the proof of this lemma. Since the constructions used to obtain G' are straightforward and rather standard, we do not prove their correctness.

Proof of (1). Let G be a SAG with output alphabet Σ and CFG (N, T, P, Z). We construct a λ-free SAG G' as follows. Intuitively, G' is obtained by adding to each nonterminal of G the finite amount of information telling which of its attributes have value λ. The productions of G' are then constructed in such a way that this information is correct; the semantic rules are of course changed by replacing attributes with value λ by λ.

Formally G' has the same output alphabet Σ and has CFG (N', T, P', Z') where $N' = \{(X, A) \mid X \in N, A \subseteq \mathrm{ATT}(X)\}$ and $Z' = (Z, \varnothing)$. For every $(X, A) \in N'$, $\mathrm{ATT}((X, A)) = \mathrm{ATT}(X) - A$ (intuitively, the attributes in A have value λ). The division into inherited and synthesized attributes is the same as in G. Let $p: X_0 \to w_1 X_1 \cdots w_k X_k w_{k+1}$ be a production in P, and let $A_j \subseteq \mathrm{ATT}(X_j)$ for $0 \leqq j \leqq k$. Consider the following candidate production $p': (X_0, A_0) \to w_1(X_1, A_1) \cdots w_k(X_k, A_k) w_{k+1}$. If the construction of RULES (p') is successful, then p' is in P' and has the semantic rules of RULES (p'), otherwise p' is not in P'. We construct RULES (p') as follows. For every semantic rule $\langle a_0, j_0 \rangle := u_1 \langle a_1, j_1 \rangle \cdots u_m \langle a_m, j_m \rangle u_{m+1}$ in RULES (p), define the string $w \in (\Sigma \cup \mathrm{ATT}(p))^*$ by $w = u_1 s_1 \cdots u_m s_m u_{m+1}$, where $s_i = \langle a_i, j_i \rangle$ if $a_i \notin A_{j_i}$, and $s_i = \lambda$ if $a_i \in A_{j_i}$. If $a_0 \in A_{j_0}$ and $w \neq \lambda$, then the construction of RULES (p') is unsuccessful. If $a_0 \notin A_{j_0}$ and $w = \lambda$, then the construction of RULES (p') is also unsuccessful. If $a_0 \notin A_{j_0}$ and $w \neq \lambda$, then the semantic rule $\langle a_0, j_0 \rangle := w$ is added to RULES (p'). (Of course, if $a_0 \in A_{j_0}$ and $w = \lambda$, then nothing is added to RULES (p').)

This ends the construction of G'. By construction G' is λ-free. It is left to the reader to prove that there is a bijection between complete derivation trees t of G (with meaning $\neq \lambda$) and complete derivation trees t' of G', such that t and t' have the same meaning. In fact, t' is the same as t, except that a node x of t with label X has label (X, A) in t' where A is the set of attributes of x with value λ. Hence $\mathrm{OUT}(G') = \mathrm{OUT}(G) - \{\lambda\}$, and we have shown (1) of Lemma 4.1.

Proof of (2). This is shown in a more general setting in [18]. For the sake of completeness we repeat the construction for our case. Let G be a λ-free SAG with output alphabet Σ and CFG (N, T, P, Z). We construct a λ-free reduced SAG G' as follows. Intuitively, G' is obtained by adding to each nonterminal of G the finite amount of information telling which of its attributes are used to compute the meaning of the derivation tree. Only these attributes are kept, the other ones are thrown away. Formally, G' has the same output alphabet Σ and has CFG (N', T, P', Z') where $N' = \{(X, A) \mid X \in N, \ A \subseteq \mathrm{ATT}(X)\}$ and $Z' = (Z, \{d\})$. For every $(X, A) \in N$, $\mathrm{ATT}((X, A)) = A$. P' is the set of all $p': (X_0, A_0) \to w_1(X_1, A_1) \cdots w_k(X_k, A_k) w_{k+1}$ such that (i) $p: X_0 \to w_1 X_1 \cdots w_k X_k w_{k+1}$ is in P; (ii) for every semantic rule $\langle a_0, j_0 \rangle :=$

$u_1\langle a_1, j_1\rangle \cdots u_m\langle a_m, j_m\rangle u_{m+1}$ in RULES (p), if $a_0 \in A_{j_0}$, then $a_i \in A_{j_i}$ for $1 \leqq i \leqq m$; and (iii) if $a \in A_j$ for some $1 \leqq j \leqq k$ and there is no semantic rule in RULES (p) that defines $\langle a, j\rangle$, then there is a semantic rule $\langle a_0, j_0\rangle := u_1\langle a_1, j_1\rangle \cdots u_m\langle a_m, j_m\rangle u_{m+1}$ in RULES (p) such that $a_0 \in A_{j_0}$ and $\langle a, j\rangle = \langle a_i, j_i\rangle$ for some $1 \leqq i \leqq m$. For such a p', RULES (p') consists of all semantic rules $\langle a_0, j_0\rangle := u_1\langle a_1, j_1\rangle \cdots \langle a_m, j_m\rangle u_{m+1}$ of RULES (p) such that $a_0 \in A_{j_0}$. Thus RULES $(p') \subseteq$ RULES (p) and so G' is still λ-free. The proof that G' is reduced and output-equivalent to G, can be found in [18] (it uses the noncircularity of G).

Proof of (3). Let G be a λ-free reduced SAG with output alphabet Σ and CFG (N, T, P, Z). We construct a λ-free reduced SAG G' which has no passing final productions. We first need some terminology. A *passing derivation tree* t is one in which all (occurrences of) productions are passing, i.e., all semantic rules that define the attributes of t are passing. For a nonterminal X, let pass $(X) = \{is(t) | t$ is a passing derivation tree with root labeled $X\}$. Thus pass (X) contains all i/s-graphs (see § 1.3) of passing derivation trees with root X; such an i/s-graph contains all information concerning the attribute computation in the tree: if there is an edge from i to s in the i/s-graph of t, then $\langle s,$ root $(t)\rangle$ gets the value of $\langle i,$ root $(t)\rangle$. Note that if $g \in$ pass (X) and $s \in$ SYN-ATT (X), then there is a unique $i \in$ IN-ATT (X) such that there is an edge from i to s in g; let us denote this i by $g(s)$, thus viewing g as a mapping SYN-ATT $(X) \to$ IN-ATT (X).

We now construct G' from G by removing from each complete derivation tree of G the passing subtrees and by short-cutting the attribute computations in these subtrees, using the information in their i/s-graphs. The construction is very similar to the usual method of making a context-free grammar λ-free.

G' has the same output alphabet Σ and has CFG (N, T', P', Z) with $T' = T \cup$ PASS, where PASS $= \cup\{$pass $(X) | X \in N\}$. Thus we use the i/s-graphs of passing derivation trees as terminals. Each nonterminal has the same attributes as in G. P' consists of all productions $p': X_0 \to w_1 Y_1 \cdots w_k Y_k w_{k+1}$ $(k \geqq 0)$ such that, for some $X_1, \cdots, X_k \in N$, $p: X_0 \to w_1 X_1 \cdots w_k X_k w_{k+1}$ is in P, and for every j $(1 \leqq j \leqq k)$, $Y_j = X_j$ or $Y_j \in$ pass (X_j), and, finally, if p is a passing production, then ($k \geqq 1$ and) $Y_j = X_j$ for some j $(1 \leqq j \leqq k)$. The final condition will ensure that G' contains no passing final productions. For production $p': X_0 \to w_1 Y_1 \cdots w_k Y_k w_{k+1}$ as above, RULES (p') is constructed from RULES (p) by the following algorithm (intuitively, the dependency graph of p' is obtained by "pasting together" the dependency graph of p and all graphs $Y_j \in$ pass (X_j)). Let S be a variable of type "set of semantic rules".

(a) Initialize S to the set of all semantic rules of RULES (p) that define attributes $\langle a, j\rangle$ with $Y_j = X_j$, $0 \leqq j \leqq k$. (We now have semantic rules defining all (appropriate) attributes in p', but their right-hand sides still contain (synthesized) attributes of nonterminals X_j that have been replaced by $Y_j \in$ pass (X_j). This is taken care of in the next step: each such synthesized attribute a has the same value as the corresponding inherited attribute $Y_j(a)$, which is again defined by a semantic rule in RULES (p).

(b) Repeat the following until it cannot be repeated any more. In some semantic rule of S, replace an occurrence of some $\langle a, j\rangle$ such that $1 \leqq j \leqq k$, $a \in$ SYN-ATT (X_j), and $Y_j \in$ pass (X_j), by the right-hand side of the semantic rule defining $\langle Y_j(a), j\rangle$ in RULES (p). (When the repetition stops, the right-hand sides of semantic rules in S only contain attributes $\langle a, j\rangle$ with $Y_j = X_j$, as required.)

(c) RULES (p') is defined to be S.

This ends the construction of RULES (p'). It should be clear that the repetition in (b) always halts, because G is noncircular.

It is not difficult to show that the complete derivation trees of G' are obtained from those of G by replacing (maximal) passing subtrees by their i/s-graphs. If t' of G' corresponds in this way to t of G, then $D(t')$ is obtained from $D(t)$ by short-cutting the paths through each passing subtree. Thus it should be clear that G and G' are output-equivalent, and that G' is still reduced, noncircular, and λ-free. We finally note that if p and p' are related as in the definition of P' above, then p' is passing if and only if p is passing (this uses the fact that G is reduced). Hence G' has no passing final productions.

Proof of (4). Let G be a λ-free reduced SAG with no passing final productions, with output alphabet Σ and CFG (N, T, P, Z). We construct a SAG G' with all these properties and no passing linear productions. The idea of the construction is very similar to that in the proof of (3): We short-cut the attribute computations in those parts of the derivation tree that consist of passing linear productions only. Again we need some additional terminology. A *linear partial derivation tree from X to Y* (with $X, Y \in N$) is a partial derivation tree t of which the root is labeled X, a leaf is labeled Y, and all productions occurring in t are linear. If t is a linear partial derivation tree from X to Y, then the *passing graph* pass (t) of t has the set of nodes $\{\langle a, -1\rangle | a \in \text{ATT}(X)\} \cup \{\langle a, 0\rangle | a \in \text{ATT}(Y)\}$, and there is an edge from $\langle a, i\rangle$ to $\langle b, j\rangle$ $(i, j \in \{-1, 0\})$ iff there is an oriented path in $D(t)$ from $\langle a, x(i)\rangle$ to $\langle b, x(j)\rangle$, where $x(-1)$ is the root of t and $x(0)$ is the leaf of t labeled Y. Note that pass (t) is very close to the dependency graph of some production $X \to Y$. For nonterminals X and Y, let pass $(X, Y) =$ {pass $(t) | t$ is a linear partial derivation tree from X to Y, and all productions occurring in t are passing}. It is left as an exercise to the reader to show that the sets pass (X, Y) are computable (in a way similar to the i/s-graphs of a nonterminal). If $g \in$ pass (X, Y) and $\langle b, j\rangle \in \{\langle c, 0\rangle | c \in \text{IN-ATT}(Y)\} \cup \{\langle c, -1\rangle | c \in \text{SYN-ATT}(X)\}$, then there is a unique $\langle a, i\rangle \in \{\langle c, 0\rangle | c \in \text{SYN-ATT}(Y)\} \cup \{\langle c, -1\rangle | c \in \text{IN-ATT}(X)\}$ such that there is an edge from $\langle a, i\rangle$ to $\langle b, j\rangle$ in g. We will denote this $\langle a, i\rangle$ by $g(\langle b, j\rangle)$. See the symbolic picture in Fig. 2.

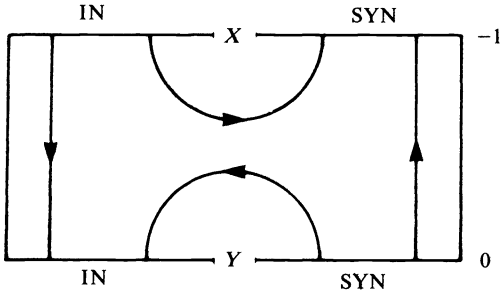


FIG. 2. *Dependencies between attributes in a passing graph.*

We now construct G' from G by removing from the complete derivation trees of G the linear partial subtrees that contain passing productions only. The construction is similar to that of the removal of monadic productions $(X_0 \to X_1)$ from a CFG. G' has the same output alphabet Σ and has CFG (N, T', P', Z) with $T' = T \cup \text{PASS}$, where $\text{PASS} = \cup \{\text{pass}(X, Y) | X, Y \in N\}$. Let P_0 be the set of all productions in P that are not passing linear. P' consists of all productions of P_0, and all productions $X \to g\beta$, where $Y \to \beta$ is in P_0 and $g \in$ pass (X, Y). Each nonterminal has the same attributes as in G. For $p \in P_0$, the semantic rules of p are the same as in G. For a production $p': X \to g\beta$ obtained from $p: Y \to \beta$ and $g \in$ pass (X, Y), the set RULES (p') is construc-

ted from RULES (p) in the following steps (during these steps symbols $\langle a, -1\rangle$, $a \in \text{ATT}(X)$, occur in the semantic rules; at the end these are replaced by $\langle a, 0\rangle$).

(a) Remove from RULES (p) the semantic rules defining the attributes $\langle s, 0\rangle$ with $s \in \text{SYN-ATT}(Y)$.

(b) To the resulting set add all semantic rules $\langle s, -1\rangle := g(\langle s, -1\rangle)$, $s \in$ SYN-ATT (X).

(c) In the resulting set, repeat each of the following steps (i) and (ii) until they cannot be repeated any more.

(i) In every semantic rule in the set, replace every occurrence of $\langle i, 0\rangle$, $i \in$ IN-ATT (Y), by $g(\langle i, 0\rangle)$.

(ii) In every semantic rule in the set, replace every occurrence of $\langle s, 0\rangle$, $s \in$ SYN-ATT (Y), by the right-hand side of the rule defining $\langle s, 0\rangle$ in (the original) RULES (p).

(d) Replace every occurrence of $\langle a, -1\rangle$, $a \in \text{ATT}(X)$, by $\langle a, 0\rangle$.

This ends the construction of RULES (p'). Note again that step (c) halts because G is noncircular. Note that the dependency graph of p' is obtained by "pasting together" g and the dependency graph of p.

As in the proof of (3) it can be argued that G and G' are output-equivalent, and that G' is still λ-free and reduced. Also, if p and p' are as above, then p' is passing iff p is passing. This shows that G' has no passing linear and no passing final productions.

This concludes the proof of (4) and thus the proof of Lemma 4.1. □

5. Main theorem. We have now shown the main result of this paper.

THEOREM 5.1. OUT (SAG) $\subseteq$ LOG (CF).

Proof. Immediate from Corollary 3.3 and Lemma 4.1. □

Note that the inclusion is effective (because all constructions in this paper are effective).

In Fig. 3 we display an inclusion diagram of some of the discussed classes of languages together with some of their subclasses (cf. § 1.6 and Theorem 2.2).

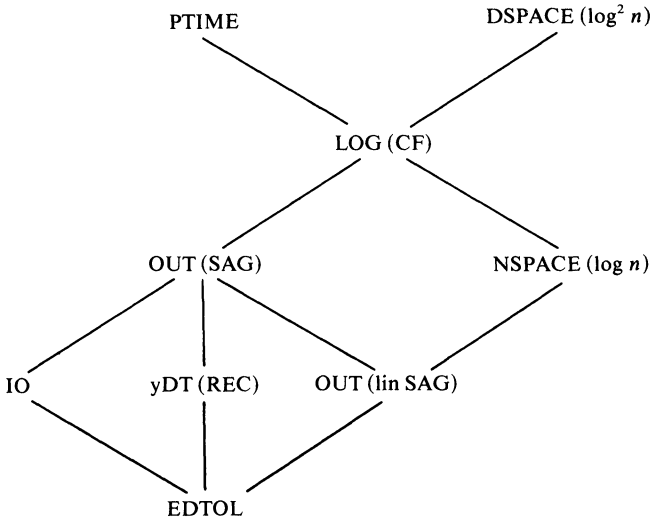


FIG. 3. *Inclusion diagram of language classes.*

IO is the class of inside-out macro languages [19]; IO $\subseteq$ OUT (SAG) was shown in [10], see also [13]. The inclusion of IO in LOG (CF) was shown in [2], and the idea of the present paper came from combining these two results. yDT (REC) is the class of deterministic topdown tree transformation languages (see [32]), i.e., the class of yields of output languages of deterministic top-down tree transducers (with a recognizable tree language as input language). It equals the class of ranges of generalized syntax-directed translation schemes (GSDTS, [1]). For the inclusion of yDT (REC) in OUT (SAG), see [20], [8], [13]; in fact yDT (REC) is the class of output languages of SAG which have synthesized attributes only. EDTOL, one of the classes of languages generated by L systems [34], is equal to the class yDT (REC), where the input trees are all monadic [16]; hence EDTOL is the class of output languages of linSAG which have synthesized attributes only. For the fact that EDTOL $\subseteq$ IO, see [9]. The inclusion of EDTOL in NSPACE ($\log n$) was shown in [25] (see also [38], [21]) and the proof of IO $\subseteq$ LOG (CF) in [2] was partly based on the proof in [25]. The idea of Theorem 2.2 came from reading [25]. Actually, our notion of assignment is an immediate generalization of the one used there.

The fact that yDT (REC) is in PTIME and in DSPACE ($\log^2 n$) did not seem to be known in the literature. For the corresponding class yT (REC) of nondeterministic topdown tree transformation languages, it is shown in [33] that yT (REC) $\subseteq$ NPTIME and in [4] that yT (REC) $\subseteq$ DSPACE (n). We note that yDT (REC) contains yB (REC), the class of nondeterministic bottom-up tree transformation languages [11]; hence these are also in LOG (CF) (to determine the complexity of the languages in yB (REC) was put as a question in [33]). Also, yDT (REC) contains the closure of the context-free languages under homomorphic replications [16], shown to be included in LOG (CF) in [37]. Finally, we note that yDT (REC) contains the images of the context-free languages under 2-way deterministic *gsm*-mappings [16].

The diagram of Fig. 3 could have been extended with yDT (IOT), the images of the IO context-free tree languages under deterministic top-down tree-to-string transductions; IO and yDT (REC) are both included in yDT (IOT), and yDT (IOT) is included in OUT (SAG); in fact, yDT (IOT) is equal to the class of output languages of L-SAG, i.e., SAG whose attributes can be evaluated in one pass from left to right, see [13].

In this paper we do not wish to prove the correctness of the diagram below OUT (SAG), i.e., to establish proper inclusions and incomparabilities. We just observe that EDTOL $\subsetneq$ OUT (linSAG); in fact, the language generated by the example linSAG in § 1.5 is not in EDTOL and not even in yDT (REC), see [12]. We finally note that it is very probable that it can be shown that CF is not included in OUT (linSAG), along the lines of the proof in [16] of the fact that CF is not included in EDTOL.

6. Conclusion. The Main Theorem shows that there is a close relationship between string-valued attribute grammars and alternating multihead finite automata, or, equivalently, multihead pushdown automata. Actually, by adding semantic constraints to SAG, it is possible to define a class of attribute grammars so that the corresponding class of output languages equals AMFA (=PTIME). It is to be expected that more relationships can be established between various types of attribute grammars and alternating automata, or automata with some type of pushdown storage such as, e.g., the auxiliary pushdown automata [7]. Note that in [24] DSPACE (n) auxiliary pushdown automata are used to determine the complexity of the circularity problem for AG; see also [15], [23] for the use of ASPACE (n) Turing machines for the same purpose. These automata may also be useful to study the complexity of the general membership problem for SAG.

Acknowledgments. I thank Peter Asveld and Gilberto Filè for many valuable comments.

REFERENCES

[1] A. V. AHO AND J. D. ULLMAN, *The Theory of Parsing, Translation, and Compiling*, two volumes, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[2] P. R. J. ASVELD, *Time and space complexity of inside-out macro languages*, Internat. J. Comput. Math., 10 (1981), pp. 3-14.

[4] B. S. BAKER, *Generalized syntax directed translation, tree transducers, and linear space*, this Journal, 7 (1978), pp. 376-391.

[5] A. K. CHANDRA, D. C. KOZEN AND L. J. STOCKMEYER, *Alternation*, J. Assoc. Comput. Mach., 28 (1981), pp. 114-133.

[7] S. A. COOK, *Characterizations of pushdown machines in terms of time-bounded computers*, J. Assoc. Comput. Mach., 18 (1971), pp. 4-18.

[8] B. COURCELLE AND P. FRANCHI-ZANNETTACCI, *Attribute grammars and recursive program schemes*, Theoret. Comput. Sci., 17 (1982), pp. 163-191 and pp. 235-257.

[9] P. J. DOWNEY, *Formal languages and recursion schemes*, Ph.D. Thesis TR 16-74, Harvard Univ., Cambridge, MA, 1974.

[10] J. DUSKE, R. PARCHMANN, M. SEDELLO AND J. SPECHT, IO-*macrolanguages and attributed translations*, Inform. and Control, 35 (1977), pp. 87-105.

[11] J. ENGELFRIET, *Bottom-up and top-down tree transformations—a comparison*, Math. Syst. Theory, 9 (1975), pp. 198-231.

[12] ———, *Three hierarchies of transducers*, Math. Syst. Theory, 15 (1982), pp. 95-125.

[13] J. ENGELFRIET AND G. FILÈ, *The formal power of one-visit attribute grammars*, Acta Informatica, 16 (1981), pp. 275-302.

[15] ———, *Passes and paths of attribute grammars*, Inform. and Control, 49 (1981), pp. 125-169.

[16] J. ENGELFRIET, G. ROZENBERG AND G. SLUTZKI, *Tree transducers, L systems, and two-way machines*, J. Comput. System Sci., 20 (1980), pp. 150-202.

[17] W. J. ERNI, *On the time and tape complexity of hyper* (1)—AFL's, Proc. 4th ICALP at Turku, Lecture Notes in Computer Science 52, Springer-Verlag, Berlin, 1977, pp. 230-243.

[18] G. FILÈ, *Interpretation and reduction of attribute grammars*, Acta Informatica, 19 (1983), pp. 115-150.

[19] M. J. FISCHER, *Grammars with macro-like productions*, Ph.D. Thesis, Harvard Univ., Cambridge, MA, 1968.

[20] Z. FÜLÖP, *On attributed tree transducers*, Acta Cybernetica, 5 (1981), pp. 261-279.

[21] T. HARJU, *A polynomial recognition algorithm for the* EDTOL *languages*, EIK, 13 (1977), pp. 169-177.

[22] J. E. HOPCROFT AND J. D. ULLMAN, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[23] M. JAZAYERI, *A simple construction for showing the intrinsically exponential complexity of the circularity problem for attribute grammars*, J. Assoc. Comput. Mach., 28 (1981), pp. 715-720.

[24] M. JAZAYERI, W. F. OGDEN AND W. C. ROUNDS, *The intrinsically exponential complexity of the circularity problem for attribute grammars*, Comm. ACM, 18 (1975), pp. 697-706.

[25] N. D. JONES AND S. SKYUM, *Recognition of deterministic* ETOL *languages in logarithmic space*, Inform. and Control, 35 (1977), pp. 177-181.

[26] D. E. KNUTH, *Semantics of Context-free languages*, Math. Syst. Theory, 2 (1968), pp. 127-145; *Correction*, Math. Syst. Theory, 5 (1971), 95-96.

[27] ———, *The Art of Computer Programming*; *Volume* 1: *Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1979.

[28] C. H. A. KOSTER, *Affix grammars*, in: Algol 68 Implementation, J. E. Peck, ed., North-Holland, Amsterdam, 1971, pp. 95-109.

[29] R. E. LADNER, R. J. LIPTON AND L. J. STOCKMEYER, *Alternating pushdown and stack automata*, this Journal, 13 (1984), pp. 135-155.

[30] P. M. LEWIS, R. E. STEARNS AND J. HARTMANIS, *Memory bounds for the recognition of context-free and context-sensitive languages*, Proc. 6th Annual IEEE Symposium on Switching Circuit Theory and Logical Design, 1965, pp. 191-212.

[31] O. LEHRMANN MADSEN, *On defining semantics by means of extended attribute grammars*, in Semantics-Directed Compiler Generation, N. D. Jones, ed., Lecture Notes in Computer Science 94, Springer-Verlag, Berlin, 1980, pp. 259-299.

[32] W. C. ROUNDS, *Mappings and grammars on trees*, Math. Syst. Theory, 4 (1970), pp. 257-287.

[33] W. C. ROUNDS, *Complexity of recognition in intermediate-level languages*, Proc. 14th Ann. IEEE Symposium on Switching and Automata Theory, 1973, pp. 145-158.

[34] G. ROZENBERG AND A. SALOMAA, *The Mathematical Theory of L Systems*, Academic Press, New York, 1980.

[35] W. L. RUZZO, *Tree-size ounded alternation*, J. Comput. System Sci., 21 (1980), pp. 218-235.

[36] I. H. SUDBOROUGH, *On the tape complexity of deterministic context-free languages*, J. Assoc. Comput. Mach., 25 (1978), pp. 405-414.

[37] ———, *The complexity of the membership problem for some extensions of context-free languages*, Internat. J. Comput. Math., 6 (1977), pp. 191-215.

[38] ———, *The time and tape complexity of developmental languages*, Proc. 4th ICALP at Turku, Lecture Notes in Computer Science 52, Springer-Verlag, Berlin, 1977, pp. 509-523.

[39] D. A. WATT, *The parsing problem for affix grammars*, Acta Informatica, 8 (1977), pp. 1-20.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
006

UPPER AND LOWER TIME BOUNDS FOR PARALLEL RANDOM ACCESS MACHINES WITHOUT SIMULTANEOUS WRITES*

STEPHEN COOK†, CYNTHIA DWORK‡ AND RÜDIGER REISCHUK§

Abstract. One of the frequently used models for a synchronous parallel computer is that of a parallel random access machine, where each processor can read from and write into a common random access memory. Different processors may read the same memory location at the same time, but simultaneous writing is disallowed. We show that even if we allow nonuniform algorithms, an arbitrary number of processors, and arbitrary instruction sets, $\Omega(\log n)$ is a lower bound on the time required to compute various simple functions, including sorting n keys and finding the logical "or" of n bits. We also prove a surprising time upper bound of $.72 \log_2 n$ steps for these functions, which beats the obvious algorithms requiring $\log_2 n$ steps.

If simultaneous writes are allowed, there are simple algorithms to compute these functions in a constant number of steps.

Key words. parallel computation, parallel random access machines, time lower bounds, sorting

1. Introduction.

In this paper we are concerned with the time required to perform sorting and other simple tasks on a synchronous parallel random access shared-memory computer. Various models for such computers have been proposed, distinguished mainly by whether a shared-memory cell can be read and/or written into by more than one processor at once. Following the terminology of Borodin and Hopcroft [BH], we shall mainly be concerned here with the PRAM [FW], which allows simultaneous reads but disallows simultaneous writes, and the WRAM, which allows both simultaneous reads and simultaneous writes, but in the latter case all processors must write the same thing [SV].

It is easy to see that the task of sorting n keys drawn from an arbitrary totally ordered list can be performed by a WRAM with an exponential number of processors in constant time (see Theorem 2 below). However, for a PRAM the best known time is $O(\log n)$ (see [BH] and Theorem 3 below).

The main result of this paper (Theorem 7) is that for PRAM's, $\log n$ is a lower time bound not only for sorting, but for such simple problems as computing the logical "or" of n bits. The lower bound applies even for nonuniform algorithms with an arbitrary number of processors, where each processor may execute arbitrary instructions. The only conditions on the parallel machine are that in one step each processor can read at most one memory cell and write into at most one memory cell (and of course no two processors can write into the same cell at once). Notice that this implies that even algorithms like bucket sort cannot be faster, except for some trivial cases; for example, if all keys are different and known beforehand.

It is interesting to note that the lower bound does not apply to the task of merging two ordered lists of size n. In fact, in [BH] it is shown that merging can be done in $O(\log \log n)$ steps by a PRAM with n processors, and this is also a lower bound even for WRAM's, provided only n processors are allowed.

* Received by the editors December 21, 1983, and in final form September 18, 1984.

† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4. This author's research was supported by the Killam Foundation of Canada.

‡ Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02138. This author's research was supported by the National Science Foundation under grant 81-01220 and the Office of Naval Research under grant ONR-N000 14-76-C-0018.

§ Fakultät für Mathematik, Universität Bielefeld, Bielefeld, West Germany. This author's research was conducted during visits to the IBM Research Laboratory, San Jose and Universität des Saarlandes, Saarbrücken.

In § 2 we describe our version of PRAM. In § 3 five simple problems, including sorting and logical "or", are defined. We show that all five require essentially the same time on a PRAM, and all can be solved in constant time on a WRAM. In § 4 a surprising algorithm is presented for computing the logical "or" of n bits (and hence solving the five problems of § 3) by a PRAM in only about $.72 \log_2 n$ steps. The algorithm beats the obvious method requiring $\log_2 n$ steps by exploiting the fact that a processor can transmit information by *not* writing into a cell. In § 5 a lower bound of about $.442 \log_2 n$ steps is proved (Theorem 7) for any PRAM which computes the logical "or". The proof is somewhat subtle because of the possibility just mentioned of transmitting information by not writing. To clarify the situation, a stronger lower bound of $\log_2 n$ steps is proved by a simpler argument in Theorem 4 when this possibility is disallowed. In Theorem 5 a lower bound of about $.72 \log_2 n$ steps is proved under the assumption that at most one fixed processor can write into a given cell at a given time step, regardless of the input. This result shows that the algorithm in Theorem 3 is exactly optimal for such restricted machines. Theorem 6 helps explain why the proof of Theorem 7 needs to be so complicated, and Theorem 8 shows that the upper bound of Theorem 3 can be beaten in case the function computed differs from the logical "or" on a small fraction of inputs.

The results in § 3 first appeared in [R], and the upper bound in Theorem 3 and versions of the lower bound in Theorem 7 were first proved in [CD] and in [R] independently.

2. The nonuniform PRAM model. The parallel machines which mainly concern us here are generalizations of the PRAM's of Fortune and Wyllie [FW]. For our purposes, a PRAM consists of a collection $\{P(1), P(2), \cdots\}$ of *processors*, a collection $\{M(1), M(2), \cdots\}$ of common memory cells, and an execution time T. At each step, each processor can read from one memory cell, do some computing, and write into one memory cell. Any number of processors can read a given memory cell at once, but we allow at most one processor to write into a given memory cell in one step. In order for the PRAM to compute a function f with n input variables $(X_1, \cdots, X_n)$ and m output variables $(Y_1, \cdots, Y_m)$ the input values $(X_1, \cdots, X_n)$ are stored initially in cells $M(1), \cdots, M(n)$, with all cells $M(i)$ containing 0 for $i > n$. After T steps, cells $M(1), \cdots, M(m)$ should contain the m components of $f(X_1, \cdots, X_n)$.

No restrictions are placed on the kind of computing done by each processor in one step, except it must be deterministic (see § 5 for a more formal definition). We even allow the number of processors and memory cells to be infinite, although the proof of Theorem 7 places a finite bound (exponential in T) on the number which can actually affect a computation. Also, our PRAMS are nonuniform in the sense that we allow a different PRAM for each value of n, the number of input variables, and the processors for a given value of n may have different programs. Thus our PRAMs are really extreme generalizations of those in [FW]. This makes the lower bound proved in § 5 more potent. In fact, the lower bound depends on the limited rate at which processors can exchange information, and not on any limitations of the processors themselves.

3. Reductions and upper bounds for WRAMs. Let us consider the following problems:

P1: Sort n keys $X_1, \cdots, X_n$ drawn from an arbitrary total ordered set.

P2: Sort n numbers $X_1, \cdots, X_n$, each either 0 or 1.

P3: Compute the sum of n numbers $X_1, \cdots, X_n$, each either 0 or 1.

P4: Compute the logical "or" of n bits $X_1, \cdots, X_n$.

P5: Let $W = w_1 \cdots w_n$ be a binary string of length n. Compute the function $id(W; *, \cdots, *): \{0,1\}^n \to \{0,1\}$ which is defined by: $id(W; X_1, \cdots, X_n) = 1$ iff for all i $w_i = X_i$.

DEFINITION. Let TIME (Pi, n) be the minimum T for which some PRAM solves problem Pi on n inputs in time T $(i = 1, \cdots, 5)$.

THEOREM 1. TIME (Pi, n) = TIME (Pj, n) + $O(1)$ for $1 \leqq i, j \leqq 5$. *More precisely,*

1. TIME (P1, n) $\leqq$ TIME (P3, n) + 3.
2. TIME (P3, n) $\leqq$ TIME (P5, n) + 2.
3. TIME (P5, n) $\leqq$ TIME (P4, n) + 2.
4. TIME (P4, n) $\leqq$ TIME (P2, n).
5. TIME (P2, n) $\leqq$ TIME (P1, n).

Proof. To prove the first inequality we must reduce sorting arbitrary keys to addition of n bits. In the first two steps each pair of keys X_i, X_j $(1 \leqq i, j \leqq n)$ is compared by one processor which writes 1 in cell M $(ni + j)$ iff $x_i > x_j$ or $x_i = x_j$ and $i \geqq j$; otherwise 0 is written. Now for each i, the bits in locations $ni + 1, \cdots, ni + n$ are taken as inputs for a PRAM which computes their sum S_i. In the final step, the original input X_i is written into location $M(S_i)$.

To prove the second inequality, we must reduce addition to the $id(W; *, \cdots, *)$ function. For each of the 2^n possible values of $W = w_1 \cdots w_n$ take a PRAM to compute $id(W; X_1, \cdots, X_n)$ (after first copying over the inputs $X_1, \cdots, X_n$ in one step). In the final step, the unique PRAM which has found $id(W; X_1, \cdots, X_n)$ to be 1 writes $w_1 + \cdots + w_n$ into cell $M(1)$.

To prove the third inequality, for each i such that $w_i = 1$ complement X_i in one step, compute P4 of the result, and complement the output.

To prove the fourth inequality note that the logical "or" of $X_1, \cdots, X_n$ is 1 iff the largest of $X_1, \cdots, X_n$ is 1.

The last inequality is obvious. □

Now suppose a WRAM is the same as a PRAM, except write conflicts are possible provided the same symbol is written. Let WTIME (Pi, n) be defined analogously to TIME (Pi, n). Then clearly all the inequalities of Theorem 1 hold when TIME is replaced by WTIME. Furthermore, there are easy algorithms which show P4 can be solved in one step and P5 can be solved in two steps on a WRAM. Since the first and second inequalities of Theorem 1 can be improved for our specific algorithms, we have the following.

THEOREM 2.

1. WTIME (P4, n) = 1 *and* n *processors suffice.*
2. WTIME (P5, n) $\leqq$ 2 *and* n *processors suffice.*
3. WTIME (P3, n) $\leqq$ 3 *and* $n2^n$ *processors suffice.*
4. WTIME (P1, n) $\leqq$ 4 *and* $n^2 2^n$ *processors suffice.*

4. An upper bound for PRAMs. There is a straightforward algorithm for a PRAM to compute the logical "or" of n bits in $\lceil \log_2 n \rceil + 1$ steps. At first glance it may appear that this is close to an obvious lower bound, since at each step apparently each processor, and therefore each memory cell, can at most double the number of input bits it "knows about". This argument is false, however, because a processor can communicate information to a memory cell by *not* writing into it. This fact is exploited in the proof of Theorem 3 below to beat the $\lceil \log_2 n \rceil + 1$ algorithm. It is this fact also which makes a lower bound even of $\Omega(\log n)$ far from obvious, and causes most of the complication in the proof of Theorem 7 in the next section.

Let the Fibonacci numbers be defined recursively by $F_0 = 0$, $F_1 = 1$, and $F_{m+2} = F_{m+1} + F_m$ for $m \geqq 0$. Then

$$F_m = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^m - \left(\frac{1-\sqrt{5}}{2}\right)^m\right].$$

THEOREM 3. *A* PRAM *can compute the logical "or" of F_{2T+1} input bits in T steps using F_{2T+1} processors and memory cells, with no read (or write) conflicts. Thus any of the functions* P1 *to* P5 *can be computed by a* PRAM *on n arguments in at most* $\log_{2.618} n + O(1)$ *steps.*

The last sentence follows from the first using Theorem 1 and the fact that $F_{2T} = \Omega((2.618)^T)$. To describe the PRAM claimed in the first sentence, let $T \geqq 1$ be arbitrary and choose $n = F_{2T+1}$. Recall that initially the input bits $X_1, \cdots, X_n$ are stored in cells $M(1), \cdots, M(n)$, respectively. Each processor $P(i)$ has local variables Y_i (Boolean) and t (integer) which are initially 0. At step t, $t = 0, 1, \cdots, T-1$ processor $P(i)$, $i = 1, \cdots, n$, executes the instructions

if $i + F_{2t} \leqq n$ *then* $Y_i \leftarrow [Y_i$ *or* $M(i + F_{2t})]$;
if $(i > F_{2t+1}$ *and* $Y_i = 1)$ *then* $M(i - F_{2t+1}) \leftarrow 1$;
$t \leftarrow t + 1$;

Note that at each step each processor reads at most one cell $M(j)$, $1 \leqq j \leqq n$, and then writes into at most one cell $M(k)$, $1 \leqq k \leqq n$. Further the read cells are distinct for distinct processors, and similarly for the write cells.

The time saving over the obvious algorithm comes from avoiding reading $M(i - F_{2t+1})$.

We will prove by induction on t, that for $0 \leqq t \leqq T$, before step t

$$\left.\begin{array}{ll} (*) & Y_i = X_i \vee X_{i+1} \vee \cdots \vee X_{i+F_{2t}-1}, \\ & \text{and} \\ (**) & M(i) = X_i \vee X_{i+1} \vee \cdots \vee X_{i+F_{2t+1}-1} \end{array}\right\} 1 \leqq i \leqq n = F_{2T+1}.$$

To make sense of these equations, we interpret $X_j = 0$ for $j > n$.

For $t = 0$, $(*)$ reduces to $Y_i = 0$ (since the empty disjunct is 0) which is true initially by assumption. Similarly $(**)$ reduces to $M(i) = X_i$, which is also true initially.

For the induction step, let Y_i^t and $M^t(i)$ be the contents of Y_i and $M(i)$, respectively, before execution of step t. Let $X(i, l)$ stand for $X_i \vee X_{i+1} \vee \cdots \vee X_{i+l-1}$. Then the induction hypothesis states that $Y_i^t = X(i, F_{2t})$ and $M^t(i) = X(i, F_{2t+1})$. The first statement of step t gives (interpreting $M^t(j)$ as 0 for $j > n$)

$$\begin{aligned} Y_i^{t+1} &= Y_i^t \vee M^t(i + F_{2t}) \\ &= X(i, F_{2t}) \vee X(i + F_{2t}, F_{2t+1}) \\ &= X(i, F_{2t} + F_{2t+1}) \\ &= X(i, F_{2(t+1)}). \end{aligned}$$

The second statement of step t gives (interpreting Y_j^{t+1} as 0 for $j > n$)

$$\begin{aligned} M^{t+1}(i) &= M^t(i) \vee Y_{i+F_{2t+1}}^{t+1} \\ &= X(i, F_{2t+1}) \vee X(i + F_{2t+1}, F_{2t+2}) \\ &= X(i, F_{2t+1} + F_{2t+2}) \\ &= X(i, F_{2(t+1)+1}). \end{aligned}$$

This completes the proof of formulas (*) and (**). In particular, $M(1) = X_1 \vee \cdots \vee X_{F_{2T+1}}$ before step T (that is, after T steps, if we start counting from 1 instead of 0). □

Note that if $Y_i = 0$, the second statement of the program communicates this fact to cell $M(i - F_{2t+1})$ by *not* writing into it.

5. Lower bounds for PRAMs. In this section we prove that a PRAM requires $\Omega(\log n)$ steps to compute the logical "or" of n bits. The lower bound applies to each of the problems P1 to P5 defined in § 3, and holds independent of the numbers of processors and cells available. Because the proof is slippery, we first provide the following careful definition of PRAM.

DEFINITION. A PRAM consists of a set $\Pi = \{P(1), P(2), \cdots\}$ of *processors*, a set $\Gamma = \{M(1), M(2), \cdots\}$ of *cells*, an alphabet Σ, a number n of inputs, and an execution time T. Each processor $P(i)$ consists of a state set Q_i (any of the sets Π, Γ, Σ, Q_i may be infinite), and functions $\rho_i : Q_i \to N^+$, $\tau_i : Q_i \to N$, $\sigma_i : Q_i \to \Sigma$, and $\delta_i : Q_i \times \Sigma \to Q_i$. Here for $q \in Q_i$, $\rho_i(q)$ is the index of the next cell to be read, $\tau_i(q)$ is the index of the next cell written into ($\tau_i(q) = 0$ indicates no cell is written into), $\sigma_i(q)$ is the symbol written, and δ_i is the state transition function.

At each time $t = 0, 1, \cdots, T$ each processor $P(i)$ is in a state $q_i^t \in Q_i$ and each cell $M(i)$ contains a symbol $s_i^t \in \Sigma$. At time $t = 0$, cells $M(1), \cdots, M(n)$ contain the inputs $X_1, \cdots, X_n$. That is, $s_i^0 = X_i$, $i = 1, \cdots, n$, and $s_i^0 = b_0$ for $i > n$, where b_0 is some distinguished (blank) member of Σ. All processors are initially in the distinguished state q_0, i.e. $q_i^0 = q_0$ for all i. In general,

$$q_i^{t+1} = \delta_i(q_i^t, s_j^t) \quad \text{where } j = \rho_i(q_i^t),$$

and

$$s_i^{t+1} = \begin{cases} \sigma_j(q_j^{t+1}) & \text{if } \tau_j(q_j^{t+1}) = i, \\ s_i^t & \text{if } \tau_j(q_j^{t+1}) \neq i \text{ for all } j. \end{cases}$$

It is a condition of correctness of the PRAM that for $t = 0, 1, \cdots, T-1$, $\tau_j(q_j^{t+1})$ and $\tau_k(q_k^{t+1})$ are either both zero or distinct for all $j \neq k$. The value $f(X_1, \cdots, X_n)$ of the function f computed by the PRAM is the contents s_1^T of cell $M(1)$ at time T. □

Notation. For a binary string $I = X_1 X_2 \cdots X_n$ of length n we denote by $I(i)$ ($1 \leqq i \leqq n$) that string which differs from I exactly at position i. If $f : \{0,1\}^n \to \{0,1\}$, then I is a *critical input* for f iff $f(I) \neq f(I(i))$ for $i = 1, 2, \cdots, n$.

For example $(0, 0, \cdots, 0)$ is a critical input for P4 (logical "or"), and in fact each of the functions P1, $\cdots$, P5 has a critical input when restricted to $\{0, 1\}$ with the ouput suitably reinterpreted in $\{0, 1\}$. Therefore the lower bounds given in Theorems 4, 5, and 7 below apply to the functions P1, $\cdots$, P5.

Before proving the lower bound for general PRAMs let us prove a stronger and more intuitive lower bound for a restricted class of PRAMs. We will call a PRAM *oblivious* iff the cell written into by each processor (and whether or not it is written into) is independent of the input, but depends only on t and the processor number. In the above formal definition, this amounts to saying that for each j, the value $\tau_j(q_j^{t+1})$ depends only on t, and not otherwise on q_j^{t+1} as the inputs $X_1, \cdots, X_n$ vary.

THEOREM 4. *If $n \geqq 2$ and $f : \{0,1\}^n \to \{0,1\}$ has a critical input, then every oblivious* PRAM *which computes f requires at least $1 + \log_2 n$ steps.*

The proof is relatively straightforward, but we shall define the relevant concepts with some care, since they will be used again in the subtler proof of Theorem 7.

DEFINITION. An input index i *affects* a processor P (respectively, a cell M) at time t *with* I iff the state of P (respectively, the contents of M) at time t with input

configuration I differs from the state of P (respectively, the contents of M) at t with input configuration $I(i)$.

Let $K(P, t, I)$ (respectively, $L(M, t, I)$) be the set of input indices which affect processor P (respectively cell M) at t with I. Let K_t and L_t satisfy the recurrence equations:

$$K_0 = 0, \tag{1}$$
$$L_0 = 1, \tag{2}$$
$$K_{t+1} = K_t + L_t, \tag{3}$$
$$L_{t+1} = K_{t+1}. \tag{4}$$

LEMMA 1. *$|K(P, t, I)| \leqq K_t$ and $|L(M, t, I)| \leqq L_t$ for all P, M, t and I.*

The proof of the lemma is by induction on t. When $t = 0$, $K(P, t, I)$ is empty and $L(M(i), t, I) = \{i\}$. In general, if processor P reads cell M at time $t+1$ with input I, then $K(P, t+1, I) \subseteq (L(M, t, I) \cup (P, t, I))$, so equation (3) is appropriate.

The only subtle point concerns equation (4). We distinguish two cases: either some processor P writes into cell M at time $t+1$ or no processor writes. Since the PRAM is oblivious, the determination of P and the determination of which case holds depends only on M and t and not on I. In the first case, we have $L(M, t+1, I) \subseteq K(P, t+1, I)$, and in the second case we have $L(M, t+1, I) = L(M, t, I)$. Thus $|L(M, t+1, I)| \leqq L_{t+1}$ by (4) and the induction hypothesis. □

To prove Theorem 4 from the lemma, note that the solution to the recurrence equations is $K_t = L_t = 2^{t-1}$, for $t \geqq 1$. If I is the critical input, then at the final step T every one of the n input indices must affect the output cell $M(1)$, so $|L(M(1), T, I)| = n$. Hence by the lemma, $n \leqq 2^{T-1}$, so $T \geqq 1 + \log_2 n$. □

Of course Theorem 4 is false when the PRAM is not required to be oblivious, as the upper bound in Theorem 3 demonstrates. In particular, to compute the logical "or" of 2 bits ($n = 2$), the PRAM in Theorem 3 requires only 1 step, whereas according to Theorem 4, an oblivious PRAM requires at least 2 steps. In fact, the nonoblivious PRAM reads cell $M(2)$ during its first step, and manages to store $M(1) \vee M(2)$ in $M(1)$ by using its option of not writing if $M(2) = 0$.

The algorithm in Theorem 3 is particularly simple because for each time t and each cell $M(i)$, there is at most one processor $P(j)$ which can write into $M(i)$ at t no matter what the input (although whether or not $P(j)$ writes may depend on the input). Let us call a PRAM *semi-oblivious* if it satisfies this restriction. It turns out that for semi-oblivious machines the upper bound in Theorem 3 is optimal.

Recall that F_m is the mth Fibonacci number (defined before Theorem 3).

THEOREM 5. *If $f: \{0, 1\}^n \to \{0, 1\}$ has a critical input and $n = F_{2T+1}$, then every semi-oblivious* PRAM *which computes f requires at least T steps, no matter how many processors and memory cells it uses. Thus the time bound in Theorem* 3 *is exactly optimal for semi-oblivious* PRAMs.

Proof. Let $K(P, t, I)$ and $L(M, t, I)$ be as in the proof of Theorem 4. Let $\hat{K}_t$ and $\hat{L}_t$ satisfy the recurrence equations:

$$\hat{K}_0 = 0, \tag{5}$$
$$\hat{L}_0 = 1, \tag{6}$$
$$\hat{K}_{t+1} = \hat{K}_t + \hat{L}_t, \tag{7}$$
$$\hat{L}_{t+1} = \hat{K}_{t+1} + \hat{L}_t = \hat{K}_t + 2\hat{L}_t. \tag{8}$$

LEMMA 2. *$|K(P, t, I)| \leqq \hat{K}_t$ and $|L(M, t, I)| \leqq \hat{L}_t$ for all P, M, t, and I.*

The proof of Lemma 2 is the same as the proof of Lemma 1, except that there is an extra complication in establishing the bound on $|L(M, t+1, I)|$. Again we distinguish two cases: either some processor P writes into cell M at time $t+1$ or no processor

writes. Since the PRAM is semi-oblivious, the determination of P depends only on M and t and not on I, but the determination of which case holds may depend upon I. In the first case we have $L(M, t+1, I) \subseteq K(P, t+1, I)$. In the second case, we have $L(M, t+1, I) \subseteq (K(P, t+1, I) \cup L(M, t, I))$, since a change in one input bit could cause P to write into M. Thus $|L(M, t+1, I)| \leqq \hat{L}_{t+1}$ by (8) and the induction hypothesis. □

To prove Theorem 5 from Lemma 2, note that the solution to the recurrence equations is $\hat{K}_t = F_{2t}$ and $\hat{L}_t = F_{2t+1}$. As before, if I is the critical input, then every one of the $n = F_{2T+1}$ input indices must affect the output cell $M(1)$ at the final step u, so $|L(M(1), u, I)| = n$. Therefore $\hat{L}_u \geqq n$, so $F_{2u+1} \geqq F_{2T+1}$, so $u \geqq T$. □

Note that the upper bound of F_{2t} on $|K(P, t, I)|$ is exactly met in the algorithm of Theorem 3.

The reader may have noticed that the argument I in $K(P, t, I)$ and $L(M, t, I)$ in the above two theorems is not really necessary, since I can be assumed to be the critical input throughout. This means that even if the no-write-conflict rule is relaxed to apply only for the critical input I and to the inputs $I(i)$ which differ in only one place from I, the lower bounds in Theorems 3 and 4 still apply. This relaxation cannot be allowed for general (as opposed to oblivious and semi-oblivious) PRAMs however, as the following result shows.

THEOREM 6. *For all $k, n \geqq 1$ there is a* WRAM *which computes the logical "or" of n bits in $2k-1$ steps and which has no write conflicts for inputs with at most k 1's.*

Proof. For each subset $\{i_1, \cdots, i_k\}$ of $\{1, 2, \cdots, n\}$ with $i_1 < i_2 < \cdots < i_k$ there is a processor $P(i_1, \cdots, i_k)$ which at the first k steps reads input cells $M(i_1), \cdots, M(i_k)$ and at step k writes 1 into $M(1)$ iff $M(i_1) = \cdots = M(i_k) = 1$. At step $k+1$, for each $i_1 < i_2 < \cdots < i_{k-1}$ there is a processor $P(i_1, \cdots, i_{k-1})$ which has read input cells $M(i_1), \cdots, M(i_{k-1})$ and which now reads $M(1)$ and writes 1 in $M(1)$ iff $(M(1) = 0$ and $M(i_1) = \cdots = M(i_{k-1}) = 1)$. Similarly, for steps $k+2, \cdots, 2k-1$ at step $k+t$ some processor writes a 1 in $M(1)$ iff no 1 has been written before and there are at least (and therefore exactly) $k-t$ 1's in the input. After $2k-1$ steps the correct output appears in $M(1)$, and there have been no write conflicts if the input has at most k 1's. Note that the bound in the theorem can be improved to $k + O(\log k)$ by using the technique of Theorem 3 in the first steps. □

Theorem 6 shows that a lower bound proof of T steps for general PRAMs must consider inputs I which differ from the critical input in at least $\frac{1}{2}T$ places. Hence in the proof below the argument I for $K(P, t, I)$ and $L(M, t, I)$ is really necessary.

THEOREM 7. *If $f: \{0,1\}^n \to \{0,1\}$ has a critical input, then every* PRAM *which computes f requires at least $\log_b n$ steps, where $b = \frac{1}{2}(5+\sqrt{21}) = 4.79 \cdots$.*

COROLLARY. *Every* PRAM *which computes one of the functions* P1, $\cdots$, P5 *requires at least $\log_b n$ steps.*

Proof of Theorem 7. Let $K(P, t, I)$ and $L(M, t, I)$ be as in the proof of Theorem 4, and let $\bar{K}_t$ and $\bar{L}_t$ satisfy the recurrence equations:

$$\bar{K}_0 = 0, \tag{9}$$

$$\bar{L}_0 = 1, \tag{10}$$

$$\bar{K}_{t+1} = \bar{K}_t + \bar{L}_t, \tag{11}$$

$$\bar{L}_{t+1} = 3\bar{K}_t + 4\bar{L}_t. \tag{12}$$

LEMMA 3. *$|K(P, t, I)| \leqq \bar{K}_t$ and $|L(M, t, I)| \leqq \bar{L}_t$ for all P, M, t, and I.*

To prove Theorem 7 from Lemma 3, note that the solution to the recurrence equations is

$$\bar{K}_t = \frac{b^t}{\sqrt{21}} = \frac{\bar{b}^t}{\sqrt{21}}, \qquad \bar{L}_t = \frac{3+\sqrt{21}}{2\sqrt{21}}\, b^t + \frac{-3+\sqrt{21}}{2\sqrt{21}}\, \bar{b}^t,$$

where $b=\frac{1}{2}(5+\sqrt{21})$ and $\bar{b}=\frac{1}{2}(5-\sqrt{21})$. As before, if I is the critical input, then at the final step T every one of the n input indices must affect the output cell $M(1)$, so $|L(M(1), T, I)|=n$. Hence by the lemma, $n \leqq \bar{L}_T \leqq b^t$, and the theorem follows.

Now we prove Lemma 3 for all P, M, t and I by induction on t. The base case $t=0$, and the part of the induction step showing $|K(P, t+1, I)| \leqq \bar{K}_{t+1}$ are the same as in the proofs of Lemmas 1 and 2. To get a bound on $|L(M, t+1, I)|$ we distinguish two cases.

Case A. Some processor P writes into M with input I at time $t+1$. Then index i can only affect M at $t+1$ with I if i affects P at $t+1$ with I. Hence $|L(M, t+1, I)| \leqq |K(P, t+1, I)| \leqq \bar{K}_{t+1}=\bar{K}_t+\bar{L}_t<\bar{L}_{t+1}$.

Case B (the interesting case). No processor writes into M with input I at time $t+1$.

DEFINITION: Input index i *causes* processor P to write into M at $t+1$ with I iff P writes into M with $I(i)$ at time $t+1$.

Thus index i can only affect M at $t+1$ with I if i affects M at t with I or if i causes some P to write into M at $t+1$ with I. Hence

$$L(M, t+1, I) \subseteq (L(M, t, I) \cup Y(M, t+1, I)), \tag{13}$$

where $Y=Y(M, t+1, I)$ is the set of indices i which cause some P to write into M at $t+1$ with I.

Our goal now is to obtain a bound on $|Y|$. Let $Y=\{u_1, \cdots, u_r\}$, where $r=|Y|$, and suppose index u_i causes processor Q_i to write into M with I, for $i=1, 2, \cdots, r$.

LEMMA 4. *For all pairs $u_i, u_j \in Y$ with $Q_i \neq Q_j$, either u_i affects Q_j at $t+1$ with $I(u_j)$ or u_j affects Q_i at $t+1$ with $I(u_i)$.*

The reason for Lemma 4 is simply that if neither condition holds, then with input $I(u_i)(u_j)$ both Q_i and Q_j will write into M at time $t+1$. □

Now consider the bipartite graph G whose two node sets are $\{u_1, \cdots, u_r\}$ and $\{v_1, \cdots, v_r\}$, where the v's are any r distinct new objects. Let us say that there is an edge between u_i and v_j iff u_i affects Q_j at $t+1$ with $I(u_j)$. Let e be the number of edges in G. Then the degree of node v_j is at most $|K(Q_j, t+1, I(u_j))| \leqq \bar{K}_{t+1}$. Hence

$$e \leqq r\bar{K}_{t+1}. \tag{14}$$

Now let us obtain a lower bound on the number of pairs u_i, u_j in Y with $Q_i \neq Q_j$, as in Lemma 4. There are $r=|Y|$ choices for u_i, and given u_i there are at least $r-\bar{K}_{t+1}$ choices for u_j, since at most $|K(Q_i, t+1, I)| \leqq \bar{K}_{t+1}$ indices can cause Q_i to write into M at $t+1$ with I. Therefore by Lemma 4 we have

$$\tfrac{1}{2}r(r-\bar{K}_{t+1}) \leqq e. \tag{15}$$

From (14) and (15) we obtain $\frac{1}{2}r(r-\bar{K}_{t+1}) \leqq r\bar{K}_{t+1}$ or $r \leqq 3\bar{K}_{t+1}=3\bar{K}_t+3\bar{L}_t$ by (11). Since $r=|Y(M, t+1, I)|$, we have by (13) and the induction hypothesis $|L(M, t+1, I)| \leqq 3\bar{K}_t+4\bar{L}_t=\bar{L}_{t+1}$, as required. □

There is a gap between the upper bound of $\log_{2.618} n$ steps in Theorem 3 and the lower bound of $\log_{4.79} n$ steps in Theorem 7. According to Theorem 5, the only way to improve the upper bound is by a PRAM for which two different processors write into the same cell at the same step for different inputs. We do not know how to design such a PRAM to improve the upper bound for the logical "or", but it can be improved for a function which resembles the "or" and has a critical input.

THEOREM 8. *Let n be a power of 3. Then there is a function $f: \{0, 1\}^n \to \{0, 1\}$ with a critical input which can be computed by a* PRAM *in $1+\log_3 n$ steps with n processors.*

Proof. The function f (computed by the program below) will be the logical "or" of the inputs when at most two input bits are 1, so that $(0, 0, \cdots, 0)$ is a critical input.

In accordance with Theorem 6, at each step $k>1$ there are up to three different processors which may write into a given cell, depending on the input.

It will be convenient to number the processors and memory cells from 0 instead of 1. Thus initially input bits $X_1, \cdots, X_n$ are stored in cells $M(0), \cdots, M(n-1)$. After $1+\log_3 n$ steps the output appears in $M(0)$. Each processor $P(i)$ has a local variable Y_i initially 0, and $M(i)=0$ for $i \geqq n$. For $0 \leqq i < n$, each processor $P(i)$ executes the following steps:

Step 1 of processor $P(i)$:

(a) *if* $i \equiv 0 \bmod 3$ *then* $Y_i \leftarrow M(i)$
(b) *if* $i \equiv 1 \bmod 3$ *then* $Y_i \leftarrow M(i-1)$
(c) *if* $i \equiv 2 \bmod 3$ *and* $M(i-1)=1$ *then* $M(i) \leftarrow 1$.

Step k of processor $P(i)$ for $k>1$:

(a) *if* $i \equiv 0 \bmod 3^k$ *then* $Y_i \leftarrow [Y_i$ *or* $M(i+3^{k-1}-1)]$ *and* $M(i) \leftarrow [Y_i$ *or* $M(i+3^{k-1}-1)]$
(b) *if* $i \equiv 1 \bmod 3^k$ *then* $Y_i \leftarrow [Y_i$ *or* $M(k+3^{k-1}-2)]$
(c) *if* $i \equiv 3^{k-1} \bmod 3^k$ *and* $Y_i=1$ *and* $M(i+3^{k-1}-1)=0$ *then* $M(i-2*3^{k-1}-1) \leftarrow 1$
(d) *if* $i \equiv 3^{k-1}+1 \bmod 3^k$ *and* $Y_i=0$ *and* $M(i+3^{k-1}-1)=1$ *then* $M(i+2*3^{k-1}-2) \leftarrow 1$
(e) *if* $i \equiv 2*3^{k-1} \bmod 3^k$ *and* $Y_i=0$ *and* $M(i-1)=1$ *then* $M(i+3^{k-1}-1) \leftarrow 1$.

To show that there are no write conflicts, observe that for any $k \geqq 1$ and any $c=0, 1, \cdots$, at step k processors with numbers between $c*3^k$ and $(c+1)*3^k-1$ access only memory locations in that same range for reads and writes. In that range for $k>1$ only location $M(c*3^k)$ in instruction a) and $M((c+1)*3^k-1)$ in instructions c), d) and e) are possibly written into. To see that only one of the 3 processors addressed in instructions c), d) and e) actually writes into cell $M((c+1)*3^k-1)$, we check that the "ands" of the second and third conditions of the *if* clauses in instruction c), d) and e) are mutually exclusive. This is because for each value of c,

(1) $M(i+3^{k-1}-1)$ in c) and $M(i-1)$ in e) are the same cell,
(2) the value of Y_i in c) and the value of Y_i in d) are the same, by the lemma below, and
(3) the value of $M(i+3^{k-1}-1)$ in d) and the value of Y_i in e) are the same, by the lemma below.

We now give a recursive definition of the functions computed in various cells and variables Y_i. To this end, we shall use the notation $x[i, l]$ to mean $x[i], x[i+1], \cdots, x[i+l-1]$, and the notation $x[-l, i]$ to mean $x[i-l+1], x[i-l+2], \cdots, x[i]$. For $k \geqq 1$ let f^k and g^k be Boolean functions of 3^{k-1} and $2*3^{k-1}$ variables, respectively, defined by

$$f^1(x)=x,\ g^1(x[1], x[2])=x[1]\ \textit{or}\ x[2],$$

$$f^k(x[1, 3^{k-1}])=f^{k-1}(x[1, 3^{k-2}])\ \textit{or}\ g^{k-1}(x[-2*3^{k-2}, 3^{k-1}]),$$

$$g^k(x[1, 2*3^{k-1}])=z_4\ \textit{or}\ ([z_1\ \textit{or}\ z_2\ \textit{or}\ z_3]\ \textit{and}\ [\bar{z}_1\ \textit{or}\ \bar{z}_2\ \textit{or}\ \bar{z}_3])$$

where $\bar{z}_i = \textit{not}\ z_i$ and

$$z_1=f^{k-1}(x[1, 3^{k-2}]),$$

$$z_2=g^{k-1}(x[-2*3^{k-2}, 3^{k-1}]),$$

$$z_3=f^{k-1}(x[3^{k-1}+1, 3^{k-2}]),$$

$$z_4=g^{k-1}(x[-2*3^{k-2}, 2*3^{k-1}]).$$

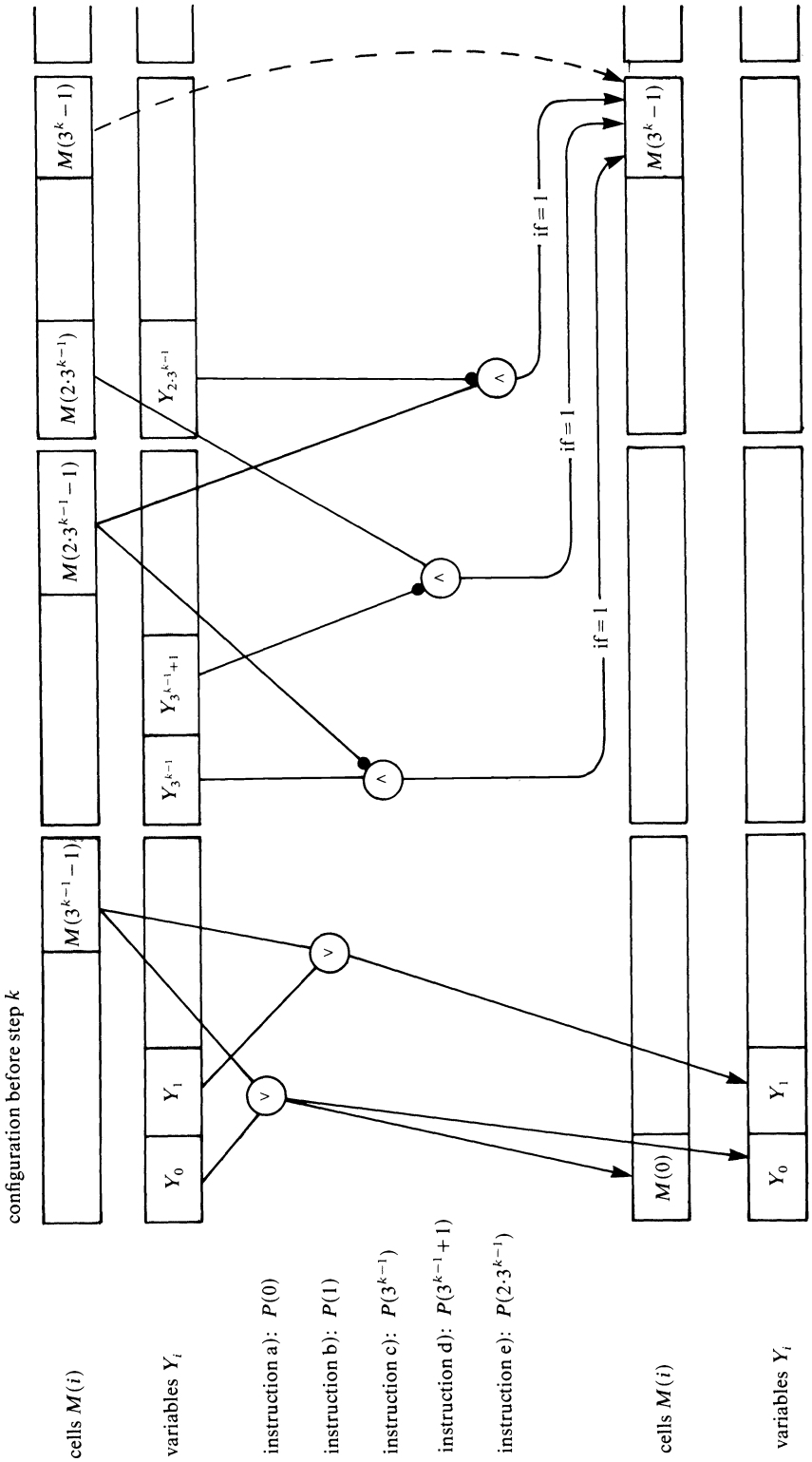

configuration before step k
cells $M(i)$
variables Y_i
instruction a): $P(0)$
instruction b): $P(1)$
instruction c): $P(3^{k-1})$
instruction d): $P(3^{k-1}+1)$
instruction e): $P(2\cdot 3^{k-1})$
$M(3^{k-1}-1)$
$M(2\cdot 3^{k-1}-1)$
$M(2\cdot 3^{k-1})$
$M(3^k-1)$
Y_0
Y_1
$Y_{3^{k-1}}$
$Y_{3^{k-1}+1}$
$Y_{2\cdot 3^{k-1}}$
if = 1
if = 1
if = 1
$M(0)$
$M(3^k-1)$
configuration after step k
cells $M(i)$
variables Y_i

FIG. 1

For $i \geqq 0$ and $k \geqq 1$ let $m(i, k)$ be the value of cell $M(i)$ and let $y(i, k)$ be the value of local variable Y_i after execution of step k.

LEMMA 5. 1) *For* $i \equiv 0 \bmod 3^k$ $y(i, k) = m(i, k) = y(i+1, k) = f^k(M(i, 3^{k-1}))$.

2) *For* $i \equiv -1 \bmod 3^k$ $m(i, k) = g^k(M(-2*3^{k-1}, i))$.

The proof is by induction on k. For $k = 1$, the equations in 1) become $y(i, 1) = m(i, 1) = y(i+1, 1) = M(i)$, and the equation in 2) becomes $m(i, 1) = (M(i-1)$ *or* $M(i))$. These are evident from the instructions for step 1.

For $k > 1$, the instructions a) and b) of step k update the values of Y_i and $M(i)$ appropriately for (1). To see that (2) holds, refer to Fig. 1, and note that if $i = (c+1)*3^k - 1$ and if we set $z_1 = f^{k-1}(M(c*3^k + 3^{k-1}, 3^{k-2}))$, $z_2 = g^{k-1}(M(-2*3^{k-2}, c*3^k + 2*3^{k-1}))$, $z_3 = f^{k-1}(M(c*3^k + 2*3^{k-1}, 3^{k-2}))$, and $z_4 = g^{k-1}(-2*3^{k-2}, i)$ then

$$m(i, k) = z_4 \quad \textit{or} \quad (z_1 \text{ and } z_2) \quad \textit{or} \quad (z_1 \text{ and } z_3) \quad \textit{or} \quad (z_3 \text{ and } z_2).$$

The last expression is equivalent to the expression in the definition of g^k. □

From the lemma we see that $f^{k+1}(X_1, \cdots, X_{3^k})$ is computed by processors $P(0), \cdots, P(3^k - 1)$ in $k+1$ steps. It is easy to check by induction on k from the definitions of f^k and g^k that both functions behave like the logical "or", provided at most two of their arguments are 1. Therefore $(0, \cdots, 0)$ is a critical input for f^k. □

It is interesting to estimate the fraction of inputs for which f^k and g^k differ from the logical "or". Let $p(k)$ (respectively $q(k)$) denote the ratio between the number of inputs on which f^k (respectively g^k) equal 0 and the total number of inputs. These two functions have the following recurrence relations:

$$p(1) = 0.5, \qquad q(1) = 0.25,$$

$$p(k+1) = p(k)*q(k) \quad \text{and} \quad q(k+1) = p(k)^{2}*q(k)^{2} + q(k)*(1 - p(k))^{2}*(1 - q(k)).$$

By simple arithmetic calculation we get

$$\begin{aligned} p(k+2)/p(k+1) = q(k+1) &= q(k)[p(k)^{2}*q(k) + (1 - p(k))^{2}*(1 - q(k))] \\ &\leqq q(k). \end{aligned}$$

This implies that $q(k)$ is monotone decreasing, and for fixed l $p(k)$ can be bounded by $O(q(l)^k)$. Evaluating the recurrence relations for some values yields $q(7) \leqq 0.04$. Thus $p(k) = O(.04^k)$.

Acknowledgments. Les Valiant got the first two authors started out right on their version of the lower bound proof, and Allan Borodin, John Gilbert, John Hopcroft, and Les Valiant all provided helpful criticisms of early attempts at that proof.

REFERENCES

[BH] A. BORODIN AND J. E. HOPCROFT, *Routing, merging, and sorting on parallel models of computation*, extended abstract. Proc. 14th ACM STOC (May 1982), pp. 338-344, J. Comput. System Sci., to appear.

[CD] S. A. COOK AND C. DWORK, *Bounds on the time for parallel* RAM's *to compute simple functions*, Proc. 14th ACM STOC (May 1982), pp. 231-233.

[FW] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, Proc. 10th ACM STOC (May 1978), pp. 114-118.

[R] R. REISCHUK, *A lower time-bound for parallel random access machines without simultaneous writes*, IBM Research Report RJ 3431 (March 1982).

[SV] Y. SHILOACH AND U. VISHKIN, *Finding the maximum, merging, and sorting in a parallel computation model*, J. Algorithms, 2 (1981), pp. 88-102.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
007

THE BOYER–MOORE–GALIL STRING SEARCHING STRATEGIES REVISITED*

ALBERTO APOSTOLICO† AND RAFFAELE GIANCARLO‡

Abstract. Based on the Boyer-Moore-Galil approach, a new algorithm is proposed which requires a number of character comparisons bounded by 2n, regardless of the number of occurrences of the pattern in the textstring. Preprocessing is only slightly more involved and still requires a time linear in the pattern size.

Key words. string searching, pattern matching, shift functions, text editing, analysis of algorithms

1. Introduction. The string searching problem is to find all occurrences of a given *pattern* y in a given *text* x, both y and x being strings over a finite alphabet.

Letting $|x| = n$ and $|y| = m$, brute force procedures that involve $\Theta(nm)$ comparisons in the worst case can be quickly developed. However, as the copious literature [1]-[8] devoted to this subject over the past decade shows, the bound can be lowered to $\Theta(n)$, provided some preprocessing of the pattern is allowed. As pointed out by Boyer and Moore [2], the time spent in the preprocessing generally plays a secondary role in the overall design perspective. However, it is fortunate that all preprocessing strategies set up so far perform in time $\Theta(m)$.

As is well known, one of the first string searching algorithms was proposed in [2]. Unlike the Knuth-Morris-Pratt algorithm [6], it compares y with x starting from the right end of y. The performance of this algorithm is quite good on the average case, where it performs in $O(n/m)$. On the other hand, it displays a worst case running time $\Theta(n^2)$.

Improving over the Boyer-Moore (BM) algorithm, Knuth, Morris and Pratt [6] also set up a modified version of it that performs at most $6n$ character comparisons, if the pattern does not appear in the text. More recently, Guibas and Odlyzko [5] narrowed that bound to $4n$ and conjectured it is $2n$. Zvi Galil [3] presented a new version of the modified BM algorithm and, by using the Guibas and Odlyzko result, showed a $14n$ character comparison worst case running time for his algorithm. This version is obtainable by the former one in a straightforward manner, even though it is not straightforward to prove its correctness.

As pointed out in [6], the analysis of the BM procedure is not simple. This is due to the fact that, when the BM algorithm shifts the pattern to the right, it does not retain any information about characters already matched. Based on this observation, Knuth, Morris and Pratt [6] suggested that the algorithm be made less oblivious by arranging the various situations that could arise in the course of the pattern matching process into a suitable table of "states". Problem is that the number of "states" in such a generalization of the BM strategy can be quite large (the obvious upper bound is 2^m, but it is not known how tight a bound this is). Thus the work involved in

* Received by the editors February 21, 1983, and in revised form November 8, 1984. This work was supported in part by the Italian Ministry of Education. Additional support was provided by the Italian National Council for Research and by NATO under research grant no. 039.82. An extended abstract related to this paper was presented at the 20th Annual Allerton Conference on Communication, Control, and Computing, Monticello, Illinois, October 6-8, 1982.

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

‡ Dipartimento di Informatica ed Applicazioni, The University of Salerno, I84100 Salerno, Italy.

preparing that table is prohibitive in practice. There is room to suspect that a good portion of the tables is not needed in general. Galil's algorithm can be regarded in fact as a nonoblivious version of the BM strategy which only exploits two "states".

We present here still another upgrade of the BM that keeps track of which substrings of the pattern matched which substrings of the text during previous alignments, and exploits such recordings later in the matching process. If we allow for at most one recording per shift, then the number of such states is obviously bounded by $n-m+1$. The resulting algorithm works in linear time and displays three interesting features:

(1) It performs at most $2n-m+1$ character comparisons.
(2) The proof of linearity is very simple.
(3) *dd* heuristics (in the sense of [6]) can be used instead of *dd'*, not affecting (1), (2).

The first feature conveys in our view the most interesting result of this paper: indeed it is seen to follow from the even stronger finding that no character of the text needs to be accessed more than twice. The inspection of text characters is the main (and obviously unavoidable) means by which information is acquired during any pattern matching process, so that the number of character comparisons performed is customarily considered especially significant. We shall show, however, that even taking into account the other comparisons (with the exception of those hidden in the control structure) yields the palatable bound of $11n$.

This paper is organized as follows. In § 2 we review briefly the salient features of the BM and some of its derivations. Section 3 is devoted to the exposition of our method, under the assumption that the information conveyed by preprocessing is already available. This latter problem is addressed in § 4.

2. The Boyer–Moore approach to pattern matching. We will assume that the input x (y) is stored into the array *text* [1: n] (*pattern* [1: m]).

The obvious way to locate all occurrences of y in x is by repeated aligning and checking from left to right. One innovative feature of the BM strategy is in that, for each alignment of the two strings, character comparisons are performed from right to left, starting at the right end of the pattern. As is well known, this contributes a significant speed-up in cases of mismatch (cfr. [6]), even though it leads to a quadratic worst case behavior. A compact presentation of the BM algorithm is given in [3]. We report it below for the convenience of the reader.

```
Procedure BM   * i (j) points to the current character                *
               * of the pattern (text); s [character, i] is the auxiliary *
               * 'shift' function.                                    *
j := m;
do while j ≦ n
    begin
        do i := m to 0 by −1 until pattern [i] ≠ text [j − m + i]
        if i = 0 then begin output (match at j − m + 1) end
                  else j := j + s [text [j − m + i], i]
    end
```

Tables such as s are usually referred to as *shift* functions. In [3], [6] s is formally defined as follows:

$$s[\text{character}, i] = \max\{s.\text{match}[i], s.\text{occ}[\text{character}, i]\}$$

where:

$$s.\text{match}[i] = \min\{t / t \geqq 1 \text{ and } (t \geqq i \text{ or } pattern[i-t] \neq pattern[i]) \text{ and } ((t \geqq k \text{ or } pattern[k-t] = pattern[k]) \text{ for } i < k \leqq m\}$$

(this is called dd' in [6])
and

$$s.\text{occ}[\text{character}, i] = \min\{t - m + i / t = m \text{ or } (0 \leqq t < m \text{ and } pattern[m-t] = \text{character})\}.$$

The s.match portion ensures that (1) when moved to the right the pattern will match all previously matched characters, and (2) the character of the text that causes the mismatch will be aligned with a different character of the pattern.

The s.occ heuristics causes $text[j-m+i]$ (i.e. the mismatching character) to be aligned with the closest matching character of the pattern.

The shift function s', originally introduced in [2], neglects the (2) heuristics. Instead of s.match$[i]$, we have there (cfr. the dd function in [6]):

$$s'.\text{match}[(i)] = \min\{t / t \geqq 1 \text{ and } (t \geqq k \text{ or } pattern[k] = pattern[k-t]) \text{ for } i < k \leqq m\}$$

and, correspondingly:

$$s'[\text{character}, i] = \max\{s'.\text{match}[i], s.\text{occ}[\text{character}, i]\}.$$

Both s and s' can be computed in $O(m)$ steps. The reader is referred to [6], [7] for the details of such constructions.

It is convenient to extend s (s') to deal with the case $i = 0$, as follows:

$$s[\text{character}, 0] = s'[\text{character}, 0] = \min\{t / t \geqq 1 \text{ and } pattern[k] = pattern[k+t] \text{ for } i \leqq k \leqq m - t\}.$$

This helps to resume efficiently the pattern matching process following the detection of an occurrence of the pattern.

One more improvement is derived from the observation that if the pattern is *periodic* (i.e. $y = u^k u'$, with $k > 1$ and u' a prefix of u), consecutive overlapping occurrences can be detected at once. Indeed, let u be the shortest string such that $y = u^k u'$ and let $k > 1$. Also let t denote the length of u and take $l_0 = m - t + 1$. By combining the above observations, Z. Galil set up the following modified procedure BM′ [3]:

```
Procedure BM′
j := m; l := 0;
do while j ≦ n
    begin
        do i := m to l by −1 until pattern[i] ≠ text[j − m + i]
        if i = l then
            begin
                output (match at j − m + 1); l := l0; i := 0
            end
            else l := 0
        j := j + s[text[j − m + i], i]
    end
```

The BM′ takes linear time even in the worst case. For a periodic pattern in the form $u^k u'$, the shift following a complete match must lead the prefix $u^{(k-1)}u'$ of the pattern to be aligned with the position of the text previously matched against the suffix of the pattern of the same form. This corresponds to singling out and exploiting exactly one of the many possible "states" described in [6] (all other configurations could be thought of at this point as funneled into a single "superstate".

3. The algorithm. It is convenient to give first an informal outline of our approach. To start with, consider the situation of Fig. 1, which depicts one possible "instantaneous description" of the pattern matching process: the pattern has undergone, say, t shifts, and $m-i$ successful character matches have been performed.

According to BM (BM′), if $text[j-m+i] \neq pattern[i]$ the pattern will undergo one more shift as prescribed by the s function. Letting the value of s be $s=k$, Fig. 2 displays the situation that would arise if k more successful matches are performed. Plainly stated, BM (BM′) would keep trying to extend the matched region to the left. In view of the matches achieved during the stage of Fig. 1 (dotted region in the text), however, it is immediately seen that two possibilities are open at this point:

(A) The dotted portion of the pattern is also a suffix of the pattern. In principle, this region could be skipped at once, resuming comparisons at the two characters immediately preceding the dotted areas.

(B) The dotted portion of the pattern is not a suffix of the pattern, in which case one more shift could be imposed right away.

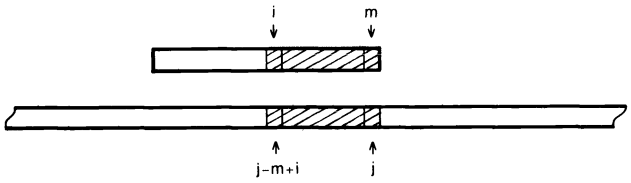


FIG. 1

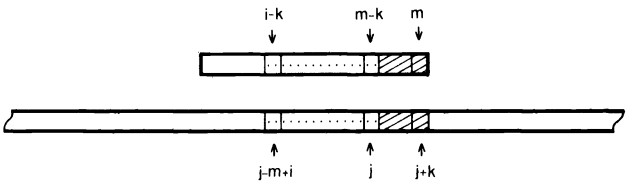


FIG. 2

Thus, if track is kept of past matched segments of the text, and if the structure of the pattern is a priori known, then the characters falling within these segments need not be reaccessed at subsequent stages. It should be pointed out that, unlike case (A) above, the segments of the text to be skipped at some stage may be more than one, in general. However, we show in this paper that the simple observation above does in fact contribute a substantial saving on the number of character comparisons needed in the process.

In order to proceed to a more formal description of our algorithm we need some means to keep track of which segments of the text matched some suffix of the pattern. In addition, we have to devise a tool—based on the structure of the pattern—that shall enable us to exploit such recorded information in a fast way.

To simplify our description at this stage, we will solve the first problem via the auxiliary array *skip* [1: n] initialized to 0 and such that whenever in the course of the matching it turns out that, say, *text* [$l-k+1$: l] = *pattern* [$m-k+1$: m] then *skip* [l] is set to k. We will show later that a much more space efficient implementation of this bookkeeping is possible, as the reader might already suspect.

The second problem calls for the introduction of the Boolean function Q: $\{1, 2, \cdots, m\} \times \{1, 2, \cdots, m\} \to \{\text{true, false}\}$, defined as follows:

$$Q[i, k] = \begin{cases} \text{true} & \text{iff } (k \leqq i \text{ and } \textit{pattern}\,[i-k+1\text{: } i] \neq \textit{pattern}\,[m-k+1\text{: } m]) \\ & \text{or } (k > i \text{ and } \textit{pattern}\,[1\text{: } i] \neq \textit{pattern}\,[m-i+1\text{: } m]), \\ \text{false} & \text{otherwise.} \end{cases}$$

We defer to § 4 the actual construction of Q.

The role of the above two implements is transparent. Indeed, assume that *skip* [l] > 0 and Q[i, *skip* [l]] is false. Then either *text* [$l-$*skip* [l]$+1$: l] = *pattern* [$i-$*skip* [l]$+1$: i] and $i >$ *skip* [l], or *text* [$l-i+1$: l] = *pattern* [1: i] and $i \leqq$ *skip* [l]. In the first case a text segment has been bumped into, which falls entirely within the pattern and which is known to match the pattern in its current position. Otherwise an occurrence of the entire pattern has been detected. We shall see that the management of this latter case embodies the ideas in [3].

The listing of the procedure BM″, which is given below, features the function s' in place of s. This has to do with the computations of the shifts that have to follow the detection of the condition: Q[i, *skip* [l]] = true. In this case it is known that an already visited segment of the text does not match the substring w of the pattern currently aligned with it, yet it is not known where exactly a character mismatch is located. On the other hand, the function s (s') takes characters and not substrings as one of its arguments. We stipulate in this case to impose a shift based on the value returned by s' in correspondence with the rightmost character of the string w. Notice that this extension of the function s' cannot result in a longer shift, compared to that based on the character that actually causes mismatch. We leave it as an exercise for the reader to show that, in unorthodox circumstances such as above, s could not consistently handle the shift. Although one could envision to use both tables, we elect here to give up the more informative shift function s (dd' in [6]), in favor of the conceptually simpler version s' (dd in [6]). Fortunately, this has no influence on the upper bound on the number of character comparisons for our strategy. The construct *andif* in the listing of BM″ is assumed not to check the second condition if the first is false.

```
Procedure BM″
j := m;
do while j ≦ n
        begin
                do i := m to 0 by −max (1, skip [j − m + i])
                    until Q[i, skip [j − m + i]] or ((skip [j − m + i] = 0) andif (pattern [i]
                    = text [j − m + i]))
                if i ≦ 0 then begin output (match at j − m + 1); i := 0 end
                skip [j] := m − i; j := j + s′ [text [j − m + i], i]
        end
```

As mentioned, the BM″ turns out to embody the ideas in [3]. In fact it behaves like BM′, soon after detecting an occurrence of a periodic pattern of the form $u^k u'$ ($k > 1$).

In this case *skip* [*j*] is set to *m*, resulting in a shift of length $t = |u|$. Since $Q[m-t, m]$ is false, BM″ will detect a new occurrence of the pattern after only *t* more successful matches.

THEOREM 1. *BM″ detects all occurrences of pattern* [1: *m*] *in text* [1, *n*] *by performing at most* $2n-m+1$ *character comparisons.*

Proof. The preceding discussion and the listing of BM″ establish that all the occurrences of the pattern in the text are indeed detected. The construct *andif* does not check the second condition if the first is false. Each comparison between a character of the text and a character of the pattern may result in either a match or a mismatch. If they match, then the text character will be skipped later, whence each text character can be involved in a matching comparison at most once. It is easily seen that the overall number of mismatching comparisons cannot exceed $n-m+1$. Indeed, each time a mismatch is detected this causes a shift to be performed, and there are at most $n-m+1$ shifts. Thus the number of character comparisons performed by BM″ is at most $2n-m+1$. □

Theorem 1 conveys the main result of this paper. Such gain in efficiency in terms of character comparisons is largely traded in exchange for a somewhat more complicated preprocessing. The reader might also suspect that the savings on character comparisons boosts the number of the other comparisons, some of which could be taken as surrogates for the former ones. Thus, it is of interest to account for the comparisons needed to check *skip* and *Q*. The condition *skip* $[j-m+i]=0$ is obviously detected in one comparison. We will show later than it takes two comparisons to check that $Q[i, skip[j-m+i]]$ is true. Both conditions are tested exactly each time a character comparison is performed, plus each time *skip* $[j-m+i]>0$. Since this latter circumstance can occur at most $n-m+1$ times, we derive that the checks of *Q* and *skip* cannot exceed a total of $3n-2m+2$, which yields $3(3n-2m+2)=9n-6m+6$ comparisons. Thus the number of both character and noncharacter comparisons is bounded by $11n-7m+7$, which is still slightly better than the $14n$ in [3]. This figure can be lowered further, at the expense of a more involved construction. This task, however, goes beyond the scope of this paper.

The auxiliary array *skip* [1: *n*] could be substituted by a circular array of size *m* in a straightforward way. An even better approach is to make use of a doubly linked list, as follows. Let *text* [*j*] be currently aligned with *pattern* [*m*]. Whenever a mismatch occurs following $k \geqq 1$ successful matches (possibly both of characters and string segments) the right end of the list is updated by appending a new record that stores the values of *j* and *k*. Those records that account for the segments falling within the span (*k*) of the newcomer record are disposed of. Finally, the leftmost record is released whenever the total number of records exceeds *m*. The details of this construction are quite standard and we leave them for the reader as an exercise. Having stored the value of the text index *j* each time a record is created makes it also trivial to check later as to whether or not the information stored in it is consistent with the current alignment of *pattern* and *text*. One nice feature of this implementation is its payoff in terms of space occupancy. In absolute terms, this latter is obviously bounded by $O(m)$. We notice, however, that a new entry is appended to the list only following at last one successful character match. The number *cc* of such matches can be very small, yielding an $O(cc)$ bound that might, in some instances, be better than the former.

4. Preprocessing. The analysis following Theorem 1 relies on the assumption that the truth value of $Q[i, k]$ can be retrieved in exactly two comparisons. We show now how this is made possible by a suitable preprocessing of the pattern.

Let v be a generic string of m characters. For simplicity, we will denote $v[i+1:j]$ shortly as $[i,j]_v$. Recall that a string u is a *period* of v if v is a prefix of u^k, with $k>1$. For each $i \leqq m$ let [4]:

$$reach_v[i] = \max\{j \leqq m / [0,1]_v \text{ is a period of } [0,j]_v\}$$
$$= i + \max\{j \leqq m-i / [0,j]_v = [i, i+j]_v\}.$$

Letting $v = v^R$, the *reverse* string of v, we associate with each position i in v the position $i' = m-i+1$ in w. We call i' the *conjugate of* i. Let now $revpat = pattern^R$.

LEMMA 1. $Q[i,k] = true$ *iff* $reach_{revpat}[i'-1] < \min(m, i'+k-1)$.

Proof. Assume that $Q[i,k] = \text{true}$. By definition, either (case 1) $0 \leqq k \leqq i$ and $pattern[i-k+1:i] \neq pattern[m-k+1, m]$ or (case 2) $i<k$ and $pattern[1,i] \neq pattern[m-i+1, m]$. Case 1 implies that $revpat[1:k] \neq revpat[m-i+1: m-i+k]$, that is to say $[0,k]_{revpat} \neq [i'-1, m-i+k]_{revpat}$. Thus the largest q such that $[0,q]_{revpat} = [m-i, m-i+q]_{revpat}$ must be less than k. It follows that $reach_{revpat}[i'-1] = i'-1+q < i'-1+k = m-i+k \leqq m$. Case 2 implies that $m < i+k-1$, whence we again need to prove that $reach_{revpat}[i'-1] < m$. This is easily accomplished by an argument analogous to that of case 1. Conversely, assume that $reach_{revpat}[i'-1] < \min(m, i'+k-1) = \min(m, m-i+k)$. Now $reach_{revpat}[m-1] = m-i+q$, where q is the largest integer such that $[0,q]_{revpat} = [m-i, m-i+q]_{revpat}$. Consider the case where $k>i$. Then $m < m-i+k$, whence $reach_{revpat}[m-i] < m$. Since $m = m-i+i$, it follows from the definition of *reach* that $[0,i]_{revpat} \neq [m-i, m]_{revpat}$, which implies that $Q[i,k] = \text{true}$. An analogous argument holds for the case where $0 \leqq k \leqq i$. □

The information needed for the table *reach* could be collected in linear time as a byproduct of the Knuth-Morris-Pratt algorithm [6]. A more explicit construction is the following. Let $d_1, d_2, \cdots, d_p$ be the sequence of all differences between consecutive occurrences of $w[1]$ in $w[1:m]$. We can put $reach_w[i] = i - prefix_w[i]$, where $prefix_w[i]$ is the longest prefix of w that starts at position $i+1$. This enables us to reason in terms of the more handy table $prefix_w$. To simplify matters even further, we extend $w[1:m]$ by appending one "sentinel" location to its right end. In other words, we have now an array $w[1:m+1]$ and we assume that $w[m+1]$ contains a symbol not appearing in $w[1:m]$. The array $prefix_w[1:m]$ is now filled in care of the following procedure.

Procedure *Prefix*

```
1.  for i := 1 to n do prefix_w[i] := 0 *initialize*
2.  i := d_1 - 1; k := 0;
3.  repeat k := k+1 until w[k] ≠ w[k+i] *compute first nontrivial entry*
4.  prefix_w[i] := k-1
5.  for f := 2 to p do *compute all other nontrivial entries*
6.      begin
7.      j := i
8.      i := i + d_f
9.      if prefix_w[d_f - 1] < k - d_f then prefix_w[i] := prefix_w[d_f]
10.     else begin k := max(0, k - d_f)
11.             repeat k := k+1 until w[k] ≠ w[i+1+k]
12.             prefix_w[i] := k
13.         end
14.     end
```

THEOREM 2. *The procedure Prefix correctly computes* $prefix_w[1:m]$.

Proof. $prefix_w[1: d_1 - 1]$ is filled with zero's by initialization, and it is easy to check that lines 2-4 compute $prefix_w[d_1 - 1]$. Assume now that $prefix_w$ has been correctly

computed up to a certain position i such that $w[i+1] = w[1]$ and let $prefix_w[i]$ be equal to some integer $p \geqq 1$. Let also d be the smallest integer such that $w[i+d+1] = w[1]$. Let $j = i+d$. The *repeat* loop of line 11 clearly computes $prefix_w[j]$ in the case $k - d_f \leqq 0$ (recall that all entries of $prefix_w$ are nonnegative). It remains to show that also the case $k - d_f > 0$ is dealt with consistently. This case splits in two subcases, both of which exploit the circumstance that position j falls within a replica of a prefix of w starting at i. The value of $prefix_w[i]$ has simply to be recopied from $prefix_w[d-1]$ if this latter is less than $k - d_f$ (line 9). Otherwise, $prefix_w[i]$ is at least $k - d_f$ and we need only check the following characters in an attempt to lengthen it. □

THEOREM 3. *The procedure Prefix takes $O(m)$ time.*

Proof. The total work for accessing positions i such that $w[i] = w[1]$ is obviously bounded by m. We can charge the work involved in comparing $w[k]$ and $w[i+1+k]$ to position $i+1+k$. Each such position cannot be charged more than one matching comparison. Mismatching comparisons cannot exceed $p \leqq m$, which concludes our proof. □

The procedure *Prefix*, once applied to $w = revpat$, makes readily available the desired table $reach_{revpat}$.

5. Concluding remarks. We have shown that the Boyer-Moore-Galil approach to pattern matching can be upgraded by keeping track of the segments of the pattern successfully matched with the text at each stage. Combining such recordings with a priori knowledge about the structure of the pattern yields an algorithm which accesses each text character at most twice.

From the standpoint of algorithmic combinatorics, this result is of some merit. Moreover, the increase in terms both of control structure and preprocessing overhead seems to be tolerable. Thus, the overall strategy compares rather favorably with other nontrivial ones, also in the practical perspective.

Acknowledgments. We are indebted to Z. Galil for many helpful comments and suggestions. We are also indebted to the referees for their excellent work in the revision of a preliminary version of this paper. In particular, we gratefully acknowledge the contribution conveyed by one of them, whose selfless profusion of punctual and thoroughly expert advice was of invaluable help in improving the presentation of our ideas.

REFERENCES

[1] A. V. AHO AND M. J. CORASICK, *Efficient string matching: an aid to bibliographic search*, Comm. ACM, 18 (1975), pp. 333-340.

[2] R. S. BOYER AND J. S. MOORE, *A fast string searching algorithm*, Comm. ACM, 20 (1977), pp. 262-272.

[3] Z. GALIL, *On improving the worst case running time of the Boyer-Moore string searching algorithm*, Comm. ACM, 22 (1979), pp. 505-508.

[4] Z. GALIL AND J. SEIFERAS, *Time space optimal string matching*, J. Comput. System Sci., 26 (1983), pp. 280-294.

[5] L. J. GUIBAS AND A. M. ODLYZKO, *A new proof of the linearity of the Boyer-Moore string searching algorithm*, Proc. 18th Annual IEEE Symposium on Foundations of Computer Science, 1977, pp. 189-195.

[6] D. E. KNUTH, J. H. MORRIS AND V. B. PRATT, *Fast pattern matching in strings*, this Journal, 6 (1977), pp. 189-195.

[7] W. RYTTER, *A correct preprocessing algorithm for Boyer-Moore string searching*, this Journal, 9 (1980), pp. 509-512.

[8] A. C. C. YAO, *The complexity of pattern matching for a random string*, Technical Report, Computer Science Dept., Stanford Univ., Stanford, CA, 1977.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
008

EFFICIENT SIMULATIONS AMONG SEVERAL MODELS OF PARALLEL COMPUTERS*

FRIEDHELM MEYER AUF DER HEIDE†

Abstract. A parallel computer (PC) with fixed communication network is called fair if the degree of this network is bounded, otherwise it is called unfair. In a PC with predictable communication each processor can precompute the addresses of the processors it wants to communicate with in the next t steps in $O(t)$ steps. For an arbitrary $\varepsilon > 0$ we define fair PC's M and M' with $O(n^{1+\varepsilon})$ processors each. $M(M')$ can simulate each unfair PC with predictable communication and $O(\log(n))$ storage locations per processor (each fair PC) with n processors with constant time loss. M' improves a result from [Acta Informatica, 19 (1983), pp. 269-296] where a time loss of $O(\log\log(n))$ was achieved. Assuming some reasonable properties of simulations we finally prove a lower bound $\Omega(\log(n))$ for the time loss of a fair PC which can simulate each unfair PC. Applying fast sorting or packet switching algorithms (Proc. 15th Annual ACM Symposiums on Theory of Computing, Boston, 1983, pp. 1-9; 10-16; Proc. ACM Symposiums on Principles of Distributed Computing, Ottawa, 1982) one sees easily that this bound is asymptotically tight.

Key words. parallel computers, general purpose machines, simulations

1. Introduction. In this paper we deal with the following question: How efficiently can one parallel computer (PC) with fixed communication network with bounded degree simulate all members of a certain class of PC's?

By a PC we mean a finite set of n processors which have the usual sequential capabilities. They are partially joined by wires. The graph defined by the processors and the wires is its communication network. In one step each processor is allowed to read a piece of information from a (relative to the communication network) neighboring processor. We allow several processors to read from the same processor at the same time. We assume the PC is synchronized.

Technological restrictions demand the degree of a PC, i.e. the degree of its communication network, to be bounded by a small constant.

We shall call such PC's fair. Those with large degree we call unfair. Later we shall always assume that their degree is $n-1$, i.e. that their communication network is the complete graph. We furthermore assume that each processor only has $O(\log(n))$ storage locations, each able to store one integer.

An important class of unfair PC's are those with predictable communication (unfair PC's with pred. com.).

Such a PC has the additional property that for each integer t, each processor can compute for itself the sequence of addresses of processors it wants to read from during the next t steps in $O(t)$ steps.

Famous examples of unfair PC's with pred.com. are the ascend- and descend-programs for cubes defined by Preparata and Vuillemin in [5]. Such a cube is an unfair PC with $N = 2^k$ processors, and its communication network is a k-dimensional cube. The prediction of the communication is very easy for each processor: Neighbour in direction of the first dimension, neighbour in the direction of the second dimension, and so on.

Preparata and Vuillemin could simulate this special, very regular PC with pred.com. by a fair PC with N processors, the Cube-Connected Cycles, and constant time loss.

* Received by the editors May 19, 1983, and in revised form October 1, 1984.

† Fachbereich Informatik, Johann Wolfgang Goethe-Universität, 6000 Frankfurt a.M., Federal Republic of Germany.

The aim of this paper is to determine the efficiency of the following types of fair PC's:

- n-universal PC's. They can simulate each fair PC with n processors and fixed degree.
- n-simulators. They can simulate each unfair PC with pred.com. and n processors.
- general n-simulators. They can simulate each unfair PC with n processors.

In [3] an n-universal PC with $O(n)$ processors and time loss $O(\log(n))$ is constructed. In [4] an n-universal PC with $O(n^{1+\varepsilon})$ processors for some arbitrary $\varepsilon > 0$ is constructed which has only time loss $O(\log\log(n))$.

In the third chapter of this paper this last result is improved by presenting an n-universal PC with the same number of processors as above but with constant time loss only.

In the fourth chapter we refine the ideas of the simulation of the third chapter to obtain an n-simulator. It also has $O(n^{1+\varepsilon})$ processors for some arbitrary $\varepsilon > 0$ and constant time loss.

For the above constructions we need a PC which can execute packet switching in a much more general way than for example permutation networks. In the second chapter, such PC's, so-called (a, b)-distributors, are introduced.

The last section of this paper shows that we cannot hope for fast simulations if we want to construct a general n-simulator. We prove a lower bound $\Omega(\log(n))$ for the time loss of a general n-simulator, independent on the number of its processors. This result holds when assuming some reasonable properties of the design of simulations as they are already defined in [4] for proving a time-processor trade-off for n-universal PC's.

By the results of Ajtai, Komlos and Szemeredi [1] or Reif and Valiant [6] one can show that this lower bound is tight within a constant factor.

The above authors have developed fair sorting PC's which can sort n numbers using only $O(\log(n))$ steps (with overwhelming probability in the case of Reif and Valiant).

One can easily see that with such a fair PC one can simulate one step of an unfair PC in $O(\log(n))$ steps, using such a fair sorting PC as "post-office" for transporting the respective informations between the processors. The same result can be achieved by using the parallel packet switching algorithm for Cube-Connected Cycles presented by Upfal in [7] who generalizes the corresponding algorithm for cubes due to Valiant and Brebner [8].

2. The construction of (a, b)-distributors. In this section we construct PC's which are able, in a more general way than permutation networks, to broadcast information. They are constructed similarly to the distributors shown in [4]. These PC's are needed for the constructions of the n-universal PC and the n-simulator in the next sections.

Let a, b be integers, $a \leqq b$. An (a, b)-*distributor* $D_{a,b}$ is a fair PC which has $a+b$ distinguished processors, a input processors $A_1, \cdots, A_a$ and b output processors $B_1, \cdots, B_b$. It has the following property:

If each B_i, $i \in \{1, \cdots, b\}$, has stored an integer $c_i \in \{1, \cdots, a+1\}$, then $D_{a,b}$ can initialize itself such that afterwards the following holds:

If each A_j, $j \in \{1, \cdots, a\}$, contains an integer string x_j of length $O(\log(n))$, then $D_{a,b}$ can distribute $(x_1, \cdots, x_a)$ according to $(c_1, \cdots, c_b)$, i.e. can transport each x_j, $j \in \{1, \cdots, a\}$, to each B_i with $c_i = j$, $i \in \{1, \cdots, a\}$, in $O(\log(b)+\log(n))$ steps. The above initialization is called the initialization of $D_{a,b}$ for $(c_1, \cdots, c_b)$, and the time it needs is the initialization time of $D_{a,b}$.

We now present an (a, b)-distributor based on the well-known Waksman permutation network (see [9]).

Let $b' := 2^{\lceil \log(b) \rceil}$. The b'-Waksman permutation network $W_{b'}$ is a fair PC with b' input processors $A_1, \cdots, A_{b'}$ and b' output processors $B_1, \cdots, B_{b'}$. For each partial permutation Π on $\{1, \cdots, b'\}$, $W_{b'}$ can initialize itself for Π such that afterwards there are marked pairwise disjoint paths of length $O(\log(b'))$ in $W_{b'}$ form A_i to $B_{\Pi(i)}$, $i \in \{1, \cdots, b'\}$.

In [3], Galil and Paul have shown that this initialization of $W_{b'}$ for Π can be executed in $O(\log(b')^4)$ steps. Furthermore it is well known that Batcher's sorting algorithm [2] can be implemented on $W_{b'}$ and sorts b' numbers in $O(\log(b')^2)$ steps. $W_{b'}$ has $2b' \log(b')$ processors. W_8 is shown in Fig. 1.

We now shall insert some additional wires in $W_{b'}$ as illustrated for W_8 in Fig. 2. The resulting network we call $\bar{W}_{b'}$.

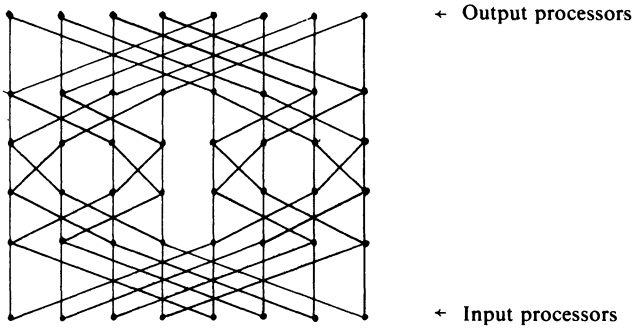


FIG. 1. *The fair* PC W_8.

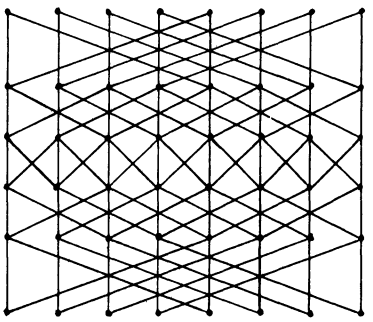


FIG. 2. *The fair* PC $\bar{W}_8$.

In addition to the capabilities of $W_{b'}$ this network can do the following: If for $i \in \{1, \cdots, b'\}$, A_i contains a number c_i, $c_1 \leqq \cdots \leqq c_{b'}$, $\bar{W}_{b'}$ can mark pairwise disjoint trees of depth $2 \cdot \log(b') - 1$ in $\bar{W}_{b'}$, one for each $x \in \{c_1, \cdots, c_b\}$. The leaves of a tree belonging to some such x are those A_i with $c_i = x$, its root is that B_i with $i = \min\{i', c_{i'} = x\}$. The existence and construction of such trees is obvious; an example is shown in Fig. 3.

We shall use these trees for transporting data from one input to many output processors along the paths in the trees. For $i \in \{1, \cdots, b\}$ let $c_i \in \{1, \cdots, a+1\}$ be stored in B_i. The following algorithm will mark a trees $G_1, \cdots, G_a$ in $\bar{W}_{b'}$. For $i \in \{1, \cdots, a\}$, the root of G_i is A_i and its leaves are all B_j's with $c_j = i$. G_i has depth

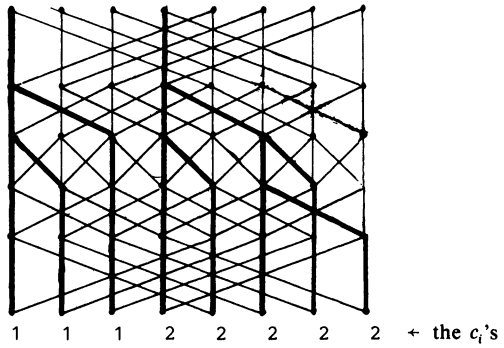


FIG. 3. $\bar{W}_8$ *with marked trees for* $(c_1, \cdots, c_8) = (1, 1, 1, 2, 2, 2, 2, 2)$.

$O(\log(b'))$. Each processor of $\bar{W}_{b'}$ lies on at most three such trees. Clearly after such a marking $\bar{W}_{b'}$ can distribute a strings each of length $O(\log(n))$ according to $(c_1, \cdots, c_b)$ in $O(\log(b') + \log(n))$ steps by sending them along the paths of the respective trees. Thus $\bar{W}_{b'}$ is initialized for $(c_1, \cdots, c_b)$.

The algorithm works as follows.

Part 1. $\bar{W}_{b'}$ sorts $c_1, \cdots, c_b$ to the sequence $d_1, \cdots, d_b$.

Part 2. $\bar{W}_{b'}$ initializes itself for the partial permutation which maps i to j, if $d_i = c_j$, $i \in \{1, \cdots, a\}$.

Remark 1. Now there are pairwise disjoint paths marked for each $i \in \{1, \cdots, b\}$ from B_i to the A_j where c_i is transported to in Part 1. For $i \in \{1, \cdots, a+1\}$ let s_i denote the smallest $j \in \{1, \cdots, b\}$ with $d_j = i$. Then for each $i \in \{1, \cdots, a\}$, $d_j = i$ for each $j \in \{s_i, \cdots, s_{i+1} - 1\}$.

Part 3. Mark a pairwise disjoint trees in $\bar{W}_{b'}$. For $i \in \{1, \cdots, a\}$, the ith tree has the root B_{s_i} and the leaves $A_j, j \in \{s_i, \cdots, s_{i+1} - 1\}$.

Remark 2. As shown above such trees can be marked in $O(\log(b'))$ steps. Each tree has depth $2 \cdot \log(b') - 1$.

Part 4. $\bar{W}_{b'}$ initializes itself for the partial permutation which maps s_i to i, $i \in \{1, \cdots, a\}$.

For $i \in \{1, \cdots, a\}$, G_i now is the tree which consists of the ith path from Part 4, the ith tree from Part 3 and the jth paths from Part 2, $j \in \{s_i, \cdots, s_{i+1} - 1\}$.

By the explanation above we know that we hereby have initialized $\bar{W}_{b'}$ for $(c_1, \cdots, c_b)$. Furthermore we know that Part 1 can be executed in $O(\log(b')^2)$ steps, Part 3 in $O(\log(b'))$ steps and Part 2 and 4 in $O(\log(b')^4)$ steps each. Thus the initialization time of $W_{b'}$ is $O(\log(b')^4)$.

THEOREM 1. *Let* a, b *be integers,* $a \leqq b$, $b' = 2^{\lceil \log(b) \rceil}$. $\bar{W}_{b'}$ *is an* (a, b)*-distributor with* $O(b \log(b))$ *processors and initialization time* $O(\log(b)^4)$.

Without proof we will point out two possible improvements of this theorem. We can construct (a, b)-distributors with initialization time $O(\log(b))$ if we are able to sort b numbers in $O(\log(b))$ steps. Ajtai, Komlos and Szemeredi [1] have done so with the help of a fair PC with $O(b \log(b))$ processors. This fair PC can also sort packets of length s according to some keys in $O(\log(b) + s)$ steps. With this result we can construct an (a, b)-distributor with $O(b \log(b))$ processors and initialization time $O(\log(b))$. A similar construction can be found in [10].

A similar result can be achieved when using the sorting algorithm from [6] due to Reif and Valiant. They have sorted b numbers on Cube-Connected Cycles using $O(\log(b))$ steps with overwhelming probability. In order to sort packets of length

$O(\log(n))$ we here need $O(\log(n))$ such fair PC's in order to do so in $O(\log(b)+\log(n))$ steps. Thus we obtain an (a, b)-distributor with $O(b\log((n))$ processors and initialization time $O(\log(b))$ which allows distributions using $O(\log(b)+\log(n))$ steps with overwhelming probability.

3. The construction of an n-universal PC. In this section we will construct an n-universal PC M_0, that means a fair PC which can simulate each fair PC with n processors and fixed degree c.

Let H be a fair PC with n processors $R_1, \cdots, R_n$ and communication network G.

The idea of our simulation is as follows. We construct a fair PC D_t^* which can simulate H for t steps, if it is prepared suitably. Furthermore we shall see that we can apply an (a, b)-distributor to prepare D_t^* before each phase of t simulated steps as demanded above. We shall choose t such that a preparation needs $O(t)$ steps. Thus t steps of H are simulated in $O(t)$ steps which yields constant time loss.

We shall now construct D_t^*. It consists of n copies $D_t^1, \cdots, D_t^n$ of a fair PC D_t, whose communication network is a complete c-ary tree of depth t. We now show how to initialize D_t^* such that it can simulate t steps of H. First we attach to each processor P of D_t^* an address $l(P)$ of the processor of H, P has to simulate.

For $i\in\{1, \cdots, n\}$ the root P of D_t^i gets $l(P)=i$. If R_i has $c'\leqq c$ neighbours $R_{i_1}, \cdots, R_{i_{c'}}$ in H, the first c' neighbours $P_1, \cdots, P_{c'}$ of P get $l(P_j)=i_j$, $j\in\{1, \cdots, c'\}$. This attachment is completed in the obvious way.

Let K be a configuration of H represented by the tuple $(K_1, \cdots, K_n)$ of configurations of the processors of H. We say, D_t^* is prepared for K, if each processor P of D_t^* for which $l(P)$ is defined contains $K_{l(P)}$. We say D_t^* simulates t steps of H started with K, if it computes K_i' in the root of D_t^i, $i\in\{1, \cdots, n\}$, where $K'=(K_1', \cdots, K_n')$ is the tth successor-configuration of K.

LEMMA 2. *If D_t^* is prepared for K, it can simulate t steps of H started with K in $O(t)$ steps.* □

The proof can be done by induction on t and is obvious. Now suppose that D_t^* has simulated t steps of H started with some configuration K. We now have to prepare D_t^* for K', the tth successor configuration of K. We know that for each $i\in\{1, \cdots, n\}$, the root processor of D_t^i has computed K_i', but the other processors P of D_t^* with $l(P)=i$ have not computed this configuration because they have earlier stopped simulating. Therefore the root processor of D_t^i has to inform these processors about K_i'. Applying an idea from [4] it suffices to transport the string Info(K, R_i, t) of numbers R_i reads from neighbours during t steps of H started with K. As a processor P of D_t^* with $l(P)=i$ knows K_i, it can, with the help of Info(K, R_i, t), compute K_i' in $O(t)$ steps. Info(K, R_i, t) has length at most t. Thus we need a network which transports Info(K, R_i, t) from the root processor of D_t^i to each processor P with $l(P)=i$, $i=1, \cdots, n$. Let m be the number of processors of D_t^*, then the above is exactly what an $(n, m-n)$-distributor as defined and constructed in the last section can do, if we identify its input processors with the root processors of $D_t^1, \cdots, D_t^n$ and its output processors with the other processors of D_t^*. This fair PC we call M_0.

Thus the exemplary of D_t^* in M_0 can be prepared for K' needing $O(\log(m-n)+t)$ steps for distributing Info(K, R_i, t)'s and $O(t)$ steps for computing K_i' in each processor P with $l(P)=i$, $i=1, \cdots, n$. Therefore, the preparation needs $O(\log(m-n)+t)+O(t)$ steps. Thus $O(\log(m-n)+t)$ steps are necessary to simulate t steps of H.

We now choose $t=\lfloor\varepsilon\log_c(n)\rfloor$ for some arbitrary $\varepsilon>0$. Then M_0 has $O(n^{1+\varepsilon}\log(n))$ processors and needs $O((1+\varepsilon)\log(n)+\varepsilon\log(n))=$

$O((1+2\varepsilon)\log(n))$ steps for simulating $\lfloor \varepsilon \log_c(n) \rfloor$ steps of H. Thus its time loss is $O((1+2\varepsilon)/\varepsilon)$.

THEOREM 3. *M_0 is an n-universal* PC *with $O(n^{1+\varepsilon}\log(n))$ processors and time loss $O((1+2\varepsilon)/\varepsilon)$ (which is a constant for fixed $\varepsilon > 0$).*

4. The construction of an n-simulator. In this section we shall construct an n-simulator. The basic idea of the construction is similar to the one in the last section. Let H be an unfair PC with pred.com. and n processors $R_1, \cdots, R_n$. We again describe a fair PC T_t^* which can simulate t steps of H in $O(t)$ steps if it is prepared in a suitable way. But in this case we see that we are not able to prepare T_t^* in $O(\log(n))$ steps, if we chose $t = O(\log(n))$. The reason is that in the case of an n-simulator, we may not demand that a processor of T_t^* simulates the same processor of H all over the simulation. Therefore we have to inform the processors of T_t^* after each phase of t simulated steps which processors of H they have to simulate now. Although it is possible to compute fast which processor of H has to be simulated by which processor of T_t^*, it turns out that broadcasting this information needs $\Omega(\log(n)^2)$ steps if $t = O(\log(n))$. The idea of how to solve this problem is the following. We shall execute an initialization of T_t^* each time before d phases of simulating t steps of H. This initialization will not be much slower than one preparation and will guarantee that afterwards d preparations can be done fast. Thus we obtain constant time loss by choosing d in an appropriate way. For simplifying the description of the simulation we shall use d copies of T_t^* in our simulation, each responsible for one of the d phases between two initializations. Each of these exemplaries, together with some (a, b)-distributors, will be called a weak n-simulator.

First we describe the fair PC T_t^* for some fixed integer t. T_t^* can simulate t steps of each unfair PC with pred.com. in $O(t)$ steps if it is prepared in an appropriate way. T_t^* consists of n exemplaries $T_t^1, \cdots, T_t^n$ of a fair PC T_t which we will define now. Its communication network is a tree whose vertices are replaced by cycles. The cycle corresponding to its root is called the root cycle, each processor on it is a root processor and one of them is the main root.

T_t is inductively defined as follows: T_0 consists of one processor, it is its main root and forms its root cycle. For $t > 0$, T_t consists of exemplaries of $T_0, \cdots, T_{t-1}$ and t new processors $P_0, \cdots, P_{t-1}$. These processors form the root cycle of T_t by wires between P_p and $P_{(p+1)\bmod(t)}$ for $p \in \{0, \cdots, t-1\}$. P_0 is the main root. Furthermore, for each $p \in \{0, \cdots, t-1\}$, P_p is joined to the main root of T_p.

An example of this fair PC is shown in Fig. 4. The following lemma can easily be proved by evaluating the obvious recursion for the number of processors of T_t and by the above definition.

LEMMA 4. *For $t \geqq 1$, T_t has $3 \cdot 2^{t-1} - 1$ processors and degree 3.*

Now let H be an unfair PC with pred. com. and n processors $R_1, \cdots, R_n$. A configuration $K = (K_1, \cdots, K_n)$ of H consists of configurations K_i for each processor R_i of H, $i \in \{1, \cdots, n\}$. Recall that each processor has only $O(\log(n))$ storage locations. Thus each K_i can be represented by a coding of its program and a list of the contents of its storage locations. This representation is an integer string of length $O(\log(n))$. In the sequel we shall identify this string with the configuration.

Let $\bar{K} = (\bar{K}_1, \cdots, \bar{K}_n)$ be the pth successor-configuration of K. Then for $i \in \{1, \cdots, n\}$, $\bar{K}_i$ is the pth successor-configuration of K for R_i.

For an integer p and $i \in \{1, \cdots, n\}$, Com (K, R_i, p) denotes the string of addresses of processors R_i reads from during p steps of H started with K. For $q \in \{1, \cdots, p\}$, Com $(K, R_i, p)_q$ denotes the qth element of Com (K, R_i, p). If for some such q, H does

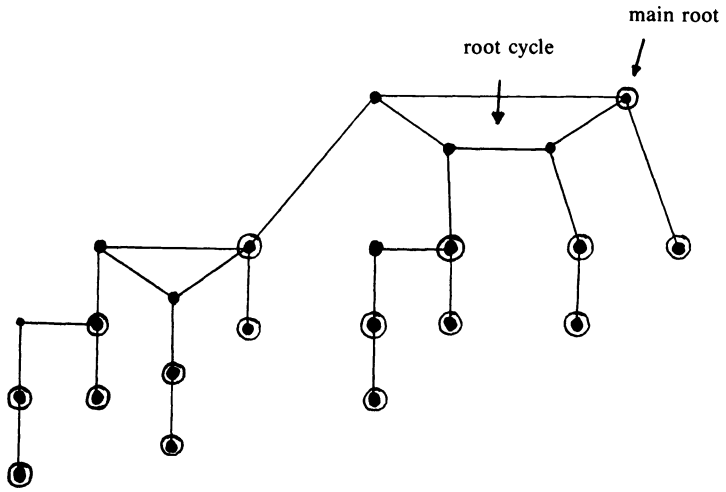


FIG. 4. *The fair* PC T_4.

not read from another processor in the qth step started with K, we assume that Com $(K, R_i, p) = i$.

Let $i \in \{1, \cdots, n\}$. We say T_t is prepared for K and R_i for t steps, if the following holds:

If $t = 0$ then T_t contains K_i.

Let $t > 0$. Then each root processor contains K_i, and for each $p \in \{0, \cdots, t-1\}$ the exemplary of T_p joint to the pth root processor is prepared for K and R_j for p steps, if $j = \text{Com}\,(K, R_i, t)_{p+1}$ and $j \neq i$. If $j = i$, there is no condition on T_p.

T_t^* is prepared for K if for each $i \in \{1, \cdots, n\}$ T_t^i is prepared for K and R_i for t steps.

The processor of H being attached by the above preparation to some processor P of T_t^* is said to be represented by P relative to K.

We say T_t^* simulates t steps of H started with K, if T_t^* executes a computation which finishes with the tth successor-configuration of K for R_i in each root processor of T_t^i, $i \in \{1, \cdots, n\}$.

LEMMA 5. *If T_t^* is prepared for K, it can simulate t steps of H started with K in $O(t)$ steps.*

Proof. Let $i \in \{1, \cdots, n\}$ be fixed, $P_0, \cdots, P_{t-1}$ be the root processors of T_t^i. Suppose that T_t^* is prepared for K.

For $p \in \{1, \cdots, t\}$ we say that the root cycle of T_t^i is p-prepared if P_{p-1} and $P_{p \bmod (t)}$ contain the pth successor-configuration of K for R_i and for each $q \in \{1, \cdots, p-1\}$, $P_{(p+q) \bmod (t)}$ contains the $(p-q)$th successor configuration of K for R_i. The root cycle of T_t^i is 0-prepared, if P_0 contains K_i. (This is fulfilled for example, when T_t^* is prepared for K.)

We now want to find an algorithm which transfers a p-prepared root cycle to a $(p+1)$-prepared one. For this purpose we first assume that for each $q \in \{0, \cdots, t-1\}$ the main root Q of the exemplary of T_q joint to P_q contains the qth successor-configuration of K for the processor R_j being represented by Q. Thus Q contains the message R_i wants to read from R_j in the $(q+1)$th step of H started with K.

Now if the root cycle is p-prepared for $p \in \{0, \cdots, t-2\}$, it becomes $(p+1)$-prepared by the following algorithm.

Part 1. For each $q \in \{1, \cdots, p\}$, $P_{(p+q) \bmod (t)}$ simulates the $(p-q+1)$th step of R_i with the help of $P_{(p+q-1) \bmod (t)}$.

Remark 1. As $P_{(p+q-1) \bmod (t)}$ has already executed this step by the definition of "p-prepared", Part 1 can be done in constant time.

Part 2. P_p simulates the $(p+1)$th step of R_i.

Remark 2. This can be done in constant time because we have assumed that the message R_i perhaps wants to read from another processor is stored in the main root of the T_p joint to P_p.

Part 3. For each $q \in \{1, \cdots, p+1\}$, $P_{(p+q) \bmod (t)}$ simulates the $(p-q+2)$th step of R_i with the help of $P_{(p+q-1) \bmod (t)}$.

Remark 3. This works in constant time, because in step 1 (resp. step 2) $P_{(p+q-1) \bmod (t)}$ just has simulated this step.

Thus T_t^* is $(p+1)$-prepared in a constant number s' of steps. Now we may inductively assume that after $s' \cdot p$ steps the root cycle of the exemplary of T_p joint to P_p is p-prepared. But this means that its main root contains the message R_i needs to execute its $(p+1)$th step after $s' \cdot p$ steps.

By our algorithm this message is needed after $s' \cdot p +$(time for step 1) many steps that means it is available when it is required by P_p. Thus P_0 contains the tth successor-configuration of K for R_i after $s' \cdot t$ steps. Clearly in further $s'' \cdot t$ steps each root processor can have stored this configuration.

Executing this algorithm in parallel for each $i \in \{1, \cdots, n\}$ we have simulated t steps of M started with K in $(s'+s'') \cdot t$ steps. □

Figure 5 shows the states of the p-prepared root cycle of T_8^i for some $i \in \{1, \cdots, n\}$ and each $p \in \{1, \cdots, 8\}$. A number l in the qth column and pth row, $q \in \{0, \cdots, 7\}$, $p \in \{1, \cdots, 8\}$ means: If T_8 is p-prepared, P_q contains the lth successor configuration of K for R_i.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
1	1	1	0	0	0	0	0	0
2	0	2	2	1	0	0	0	0
3	0	0	3	3	2	1	0	0
4	0	0	0	4	4	3	2	1
5	2	1	0	0	5	5	4	3
6	4	3	2	1	0	6	6	5
7	6	5	4	3	2	1	7	7
8	8	7	6	5	4	3	2	8

FIG. 5. *The design of a p-prepared root cycle.*

In order to obtain a fast simulation of arbitrarily many steps of H we have to prepare T_t^* before each phase of t steps for the appropriate configuration of H. In order to obtain an n-simulator of at most polynomial size we have to choose $t = O(\log(n))$ because of Lemma 4. But then we have to prepare T_t^* before each phase of t steps in $O(\log(n))$ time in order to obtain a constant time loss. We shall see later that such algorithms would need $\Omega(\log(n)^2)$ steps. They have to execute $\Omega(\log(n))$ initializations of (a, b)-distributors sequentially each of which needs $\Omega(\log(n))$ steps (see § 2). Therefore we will execute an initialization each time before d such phases of t steps, where d is chosen suitably. It turns out that this initialization for d preparations can be done in parallel and does not need much more time than one preparation.

The effect of this initialization is that afterwards d preparations can be executed, each in $O(\log(n))$ steps. This trick will guarantee constant time loss. Let in the sequel $\varepsilon > 0$ be fixed and $t := \lfloor \varepsilon \log(n) \rfloor$. Then by Lemma 4, T_t^* has at most $3n^{1+\varepsilon}$ processors.

Now we shall first define a type of fair PC's, so-called weak n-simulators, which will be used for constructing n-simulators. An explicit construction of a weak n-simulator will be given later.

A weak n-simulator M is a fair PC with the following properties.

- M contains an exemplary of T_t^*.
- If $K = (K_1, \cdots, K_n)$ is some configuration of H and for each $i \in \{1, \cdots, n\}$, each root processor of T_t^i contains $\mathrm{Com}(K, R_i, t)$, then M can initialize itself such that afterwards the following holds: If for each $i \in \{1, \cdots, n\}$, each root processor of T_t^i contains K_i, then M can prepare T_t^* for K in $O(\log(n))$ steps.

The above initialization we call the initialization of M for K and the time it needs the initialization time of M. Note that the initialization does not include the preparation of T_t^* but only guarantees that this preparation can be done fast.

We now shall construct n-simulators. Let M be some weak n-simulator with initialization time d. Then the fair PC M^* consists of $r := \lceil d/t \rceil$ exemplaries of M called $M^0, \cdots, M^{r-1}$. For each $l \in \{0, \cdots, r-1\}$, $i \in \{1, \cdots, n\}$, each root processor of T_t^i in M^l is joint to the corresponding processor in $M^{(l+1) \bmod (r)}$.

THEOREM 6. *Let M be a weak n-simulator with initialization time d. Then M^* is an n-simulator which can simulate l steps of some arbitrary unfair* PC *with pred. com. and n processors in $O(d + l/\varepsilon)$ steps. If M has m processors, M^* has $\lceil d/t \rceil \cdot m$ processors. (For fixed $\varepsilon > 0$, we thus have achieved constant time loss, if $l = \Omega(d)$.)*

Proof. The computation of the number of processors of M^* is clear. We shall construct an algorithm which simulates $d' := t \cdot r$ steps of H started with $K^0 = (K_1^0, \cdots, K_n^0)$. For $j \in \{1, \cdots, r\}$ let $K^j = (K_1^j, \cdots, K_n^j)$ be the $(t \cdot j)$th successor-configuration of K^0.

Assume that for each $i \in \{1, \cdots, n\}$, $q \in \{0, \cdots, r-1\}$, K_i^0 is stored in each root processor of T_t^i in M^q.

Now d' steps of H started with K^0 can be simulated as follows.

Part 1. For each $q \in \{0, \cdots, r-1\}$, $i \in \{1, \cdots, n\}$, each root processor of T_t^i in M^q computes $\mathrm{Com}(K^q, R_i, t)$.

Remark 1. This can be done in $O(r \cdot t) = O(d')$ steps because of the definition of predictable communication.

Part 2. For each $q \in \{0, \cdots, r-1\}$, M^q initializes itself for K^q.

Remark 2. This can (after having executed Part 1) be done in d steps as d is the initialization time of the M^q's.

Part 3. For $q = 0, \cdots, r-1$ do (sequentially)

Begin

a) M^q prepares the exemplary T' of T_t^* in M^q for K^q.

b) T' simulates t steps of H started with K^q.
Comment. Now for each $i \in \{1, \cdots, n\}$ each root processor of T_t^i in T' contains $K_i^{(q+1)}$.

c) For each $i \in \{1, \cdots, n\}$, each root processor of T_t^i in T' transports $K_i^{(q+1)}$ to the corresponding processor in $M^{(q+1) \bmod (r)}$.

End

Remark 3. Now for each $i \in \{1, \cdots, n\}$, each root processor of T_t^i in M^0 has stored K_i^r, the d'th successor-configuration of K^0 for R_i.

Remark 4. Each pass of the loop of Part 3 needs $O(\log(n))$ steps: $O(\log(n))$ for a) because of the definition of a weak n-simulator, $O(t)$ for b) because of Lemma 2, $O(\log(n))$ for c) because we have assumed that each configuration of a processor is represented by an integer string of length $O(\log(n))$. Thus Part 3 needs $O(r \cdot \log(n)) = O(d/\varepsilon)$ steps.

Part 4. For each $q \in \{0, \cdots, r-1\}$, $i \in \{1, \cdots, n\}$, K_i^r is transported to each root processor of T_t^i in M^q.

Remark 5. This can be done in $O(r \cdot \log(n)) = O(d')$ steps because of the above bound for the lengths of the representations of configurations.

Now we have achieved all preconditions for starting this algorithm again with $K^0 \leftarrow K^r$. Remarks 1, 2, 4 and 5 guarantee that we have only needed $O(d/\varepsilon)$ steps for simulating d' steps of H. Repeating this algorithm we obtain that we need $O(l/\varepsilon)$ steps for simulating l steps of H, if $l = \Omega(d)$. If l is smaller, we still have to execute Parts 1 and 2 once. Thus we need $O(d)$ steps also in this case. Therefore in general we need $O(d + l/\varepsilon)$ steps for simulating l steps of H. □

Now the problem of constructing n-simulators is reduced to constructing weak n-simulators.

This will be done with the help of (a, b)-distributors.

For $j \in \{0, \cdots, t-1\}$ let L_j be the following subset of the set of processors of T_t^*.

L_0 is the set of root processors of $T_t^1, \cdots, T_t^n$.

For $j > 0$, L_j is the set of all processors which belong to cycles which are joint to processors of L_{j-1} and which do not belong to L_{j-2} or L_{j-1}.

Informally, L_j consists of those processors which belong to a cycle in depth j of some T_t^i in T_t^*. Let $\# L_j =: m_j$, $j \in \{0, \cdots, t-1\}$.

For $j \in \{0, \cdots, t-1\}$ let D_j be an (n, m_j)-distributor with initialization time d_j.

Then the fair PC M based on $D_0, \cdots, D_{t-1}$ is defined as follows: M consists of T_t^* and $D_1, \cdots, D_{t-1}$ where for $j \in \{0, \cdots, t-1\}$ L_j is the set of input processors of D_j and the jth root processors of $T_t^1, \cdots, T_t^n$ are its output processors.

LEMMA 7. *M is a weak n-simulator with initialization time $O(\log(n)^2 + \sum_{j=0}^{t-1} d_j)$.*

Proof. Let $K = (K_1, \cdots, K_n)$ be a configuration of H, and suppose that for each $i \in \{1, \cdots, n\}$, each root processor of T_t^i has stored $\mathrm{Com}(K, R_i, t)$. Let for each processor P of T_t^* $l(P)$ be the number $i \in \{1, \cdots, n\}$ such that R_i is represented by P relative to K. Clearly, for each $i \in \{1, \cdots, n\}$, $l(P) = i$ for each root processor P of T_t^i. The following algorithm initializes M for K.

For $j = 0, \cdots, t-1$ do (sequentially)

Begin

a) D_j initializes itself for $(l(P), P \in L_j)$.

b) D_j distributes $(\mathrm{Com}(K, R_i, t), i \in \{1, \cdots, n\})$ according to $(l(P), P \in L_j)$.

c) For each $P \in L_j$; If for $q \in \{1, \cdots, t-1\}$, $p \in \{0, \cdots, q-1\}$, P is the pth root processor of an exemplary of T_q in T_t^*, then P sends $z := \mathrm{Com}(K, R_{l(P)}, t)_{p+1}$ to its neighbour Q in L_{j+1} and $l(Q) := z$.
Comment: Now for each cycle whose processors belong to L_j, one of its processors Q knows $l(Q)$.

d) For each $Q \in L_{j+1}$ which knows $L(Q)$: Q transports $l(Q)$ to each processor Q' of the cycle it belongs to, and $l(Q') := l(Q)$.

End

Obviously this algorithm attaches the correct address $l(P)$ to each processor P of T_t^*. Because of the initializations of $D_0, \cdots, D_{t-1}$ in step a) of the passes of the loop, finally M is initialized for K.

For $j \in \{0, \cdots, t-1\}$, the jth pass of the loop needs $O(d_j)+O(\log(n))+O(1)+O(t)=O(d_j+\log(n))$ steps. Thus the initialization time of M is $O(\log(n)^2+\sum_{j=0}^{t-1} d_j)$. Now a preparation of T_t^* in M can be executed in $O(\max_j\{\log(d_j)+\log(n)\})=O(\log(n))$ steps. □

We now apply Theorem 1 to Lemma 7 and obtain a weak n-simulator M_1. Because of the choice of t the number of processors of T_t^* is at most $3n^{1+\varepsilon}$. The distributors $D_0, \cdots, D_{t-1}$ all together have $\sum_{j=0}^{t-1} O(m_j \log(m_j)) = O(\log(n)\sum_{j=0}^{t-1} m_j) = O(n^{1+\varepsilon}\log(n))$ processors, as $\sum_{j=1}^{t-1} m_j$ is the number of processors of T_t^*.

Thus M_1 has $O(n^{1+\varepsilon}\log(n))$ processors.

Inserting the bound for the initialization time of $W_{b'}$ from Theorem 1 in Lemma 7, we obtain that the initialization time of M_1 is $O(\log(n)^5)$.

Inserting these results in Theorem 6 we obtain

THEOREM 8. *M_1^* is an n-simulator with $O(n^{1+\varepsilon}\log(n)^5)$ processors. M_1^* can simulate l steps of some arbitrary unfair* PC *with pred.com. and n processors in $O(\log(n)^5+l/\varepsilon)$ steps.*

With the help of the two distributors mentioned at the end of § 2, one can construct weak n-simulators M_2 and M_3.

THEOREM 9. *M_2^* (M_3^*) is an n-simulator with $O(n^{1+\varepsilon}\log(n)^2)$ processors. M_2^* (M_3^*) can simulate l steps of some arbitrary unfair* PC *with pred.com. and n processors in $O(\log(n)^2+l/\varepsilon)$ steps (with overwhelming probability).*

5. A lower bound for the time loss of general n-simulators. In this section we show that we may not hope for such fast simulations as described in the previous sections, if we want to construct general n-simulators, i.e. if we remove the restriction "predictable communication" from the unfair PC's being simulated.

We now shall define a graph-theoretical model of simulations of unfair PC's by a fair PC. For this purpose we first note that the problem arising during a simulation is that of realizing the communications between processors. Therefore we represent a step of a computation by the directed graph F with n vertices $R_1, \cdots, R_n$, in which (R_i, R_j) is an edge for some $i, j \in \{1, \cdots, n\}$, $i \neq j$, if R_i reads a piece of information from R_j in this step. As each processor can read information from at most one other processor in one step, F has outdegree one. We now call such a directed graph F with n vertices and outdegree one a computation step. A sequence $F_1, \cdots, F_l$ of computation steps is called a computation (of length l).

We now define a graph theoretical model of a simulation of a computation of length l by a fair PC M with degree c and m processors $Q_1, \cdots, Q_m$ for some integer $m \geqq n$. We shall identify M with its communication network.

The following model of simulation is that of a simulation of type 3 as defined in [4]. This model is very general: for example, each simulation developed so far in the articles quoted in this paper or in previous sections is of this type.

Let $F_1, \cdots, F_l$ be a computation. Then a simulation of $F_1, \cdots, F_l$ by M consists of pairwise disjoint, nonempty subsets $A_1^t, \cdots, A_n^t$ of $\{Q_1, \cdots, Q_m\}$ for each $t \in \{0, \cdots, l\}$. For $t \in \{0, \cdots, l\}$, $i \in \{1, \cdots, n\}$, A_i^t contains those processors which simulate R_i when t steps of the computation are simulated. The members of A_i^t are called the representatives of R_i at time t. $A_1^1, \cdots, A_n^1$ have one element, each. In order to simulate the t-th step, $t \in \{1, \cdots, l\}$, each $Q \in A_i^t$, $i \in \{1, \cdots, n\}$, must be joint by a path in M to some $Q' \in A_i^{t-1}$ and some $Q'' \in A_j^{t-1}$, if (i, j) is an edge in F_t. These paths are called the t-transport paths and their maximal length is k_t, the t-time loss of simulation. The time loss of the simulation is

$$k = \frac{1}{l}\sum_{j=1}^{l} k_j.$$

We now describe the difference between a simulation of an unfair PC with pred.com. and an unfair PC whose communication can not become predicted fast. In the first case we may assume that M knows the whole computation $F_1, \cdots, F_l$ already in the beginning of the simulation, because it can precompute it without loss of time. In the second case this is impossible; we only may assume that a general n-simulator simulates "on-line" i.e. it gets to know F_t after having simulated $F_1, \cdots, F_{t-1}$ for each $t \in \{2, \cdots, l\}$.

Now let the time loss of a general n-simulator M be the maximal time loss of the simulation of some computation.

We shall prove:

THEOREM 10. *Each general n-simulator has time loss* $\Omega(\log(n))$.

As pointed out in the introduction, this bound can be achieved with the help of the fair PC's from [1], [6], or [7].

In order to prove this theorem let M be a fair PC with m processors. We shall define a computation of infinite length for which each finite initial sequence is simulated by M with time loss $\Omega(\log(n))$.

If in some step t of a simulation $A_1, \cdots, A_n$ are the sets of representatives at time t, we say M is initialized with $A_1, \cdots, A_n$. The key observation of this proof is as follows:

There is a computation F such that either M needs at least $\gamma \log(n)$ steps to simulate it for some suitable $\gamma > 0$ or the number of representatives at time $t+1$ decreases considerably relative to the number of representatives at time t. As this number may not become smaller than n, we may not too often simulate fast. This will prove the theorem.

LEMMA 11. *For each* $\varepsilon \in (0, \frac{1}{2})$ *there is* $\gamma > 0$ *such that for each initialization of M with some* $A_1, \cdots, A_n$ *there is a computation step F with the property: M needs at least* $\gamma \log(n)$ *steps to simulate F or the sets of representatives* $A_1', \cdots, A_n'$ *after the simulation of this step fulfills*

$$\sum_{i=1}^{n} \# A_i' \leqq \frac{1}{n^{\varepsilon}} \sum_{i=1}^{n} \# A_i.$$

Proof. Let $i \in \{1, \cdots, n\}$ be fixed, and for $j \in \{1, \cdots, n\}$, $j \neq i$, let F^j be the computation step with one edge, namely (i, j). Let M be initialized with $A_1, \cdots, A_n$.

Let $\varepsilon \in (0, \frac{1}{2})$ be fixed.

Claim. There is $\gamma > 0$ such that the following holds: if for each $j \in \{1, \cdots, n\}$, $j \neq i$, M can simulate F^j in less than $\gamma \log(n)$ steps, then there is $j' \in \{1, \cdots, n\}$, $j' \neq i$, such that M simulates $F^{j'}$ in at least $\gamma \log(n)$ steps, or such that the set A_i' of representatives for R_i after the simulation of F^j fulfills $\# A_i' \leqq \# A_i / n^{\varepsilon}$.

Proof. Let k be the maximal time loss of M when simulating some F^q, $q \in \{1, \cdots, n\}$, $q \neq i$. Let $j \in \{1, \cdots, n\}$, $j \neq i$. Let $A_1', \cdots, A_n'$ be the sets of representatives after the simulation of F^j. We say, a processor $P \in A_i$ survives, if there is a transport path from P to some processor Q of A_i'. In this case we say Q is created by P. We denote the set of surviving processors form A_i by B^j. For each $P \in B^j$ we fix some $C(P) \in A_i'$ which is created by P. The element of A_j which is joint to $C(P)$ by a transport path is called the partner of P. (If there are more than one such element in A_j, pick one of them.) We now may conclude the following:

1) The partner of each $P \in B^j$ belongs to $U_{2k}(P)$[1].

2) Each $Q \in A_j$ is the partner of at most $\# U_{2k}(P)$ processors of B^j.

---

[1] For a processor P of M, $U_r(P)$ denotes the set of all processors of M which can be reached from P by a path of length at most r. For a subset A of the processors of M, $U_r(A) := \bigcup_{P \in A} U_r(P)$.

1) and 2) hold because P and its partner are joined by a path of length at most $2k$ via $C(P)$.

As for each processor P of M, $\#U_{2k}(P) \leqq c^{2k+1}$ we may conclude by 2) that there are at least $(\#B^j) \cdot c^{-2k-1}$ different partners of processors from B^j which all belong to $U_{2k}(B^j)$ because of 1).

As this holds for each $j \in \{1, \cdots, n\}$, $j \neq i$, and as these sets of partners are pairwise disjoint we obtain:

$$\#U_{2k}(A_i) \geqq \frac{1}{c^{2k+1}} \cdot \sum_{\substack{j=1 \\ j \neq i}}^{n} \#B^j.$$

As on the other hand $\#U_{2k}(A_i) \leqq c^{2k+1} \cdot \#A_i$, we obtain that

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} \#B^j \leqq \#A_i \cdot c^{4k+2}.$$

Now let $j' \in \{1, \cdots, n\}$, $j \neq i$, be chosen such that $\#B^{j'}$ is minimal.
Then

$$\#B^{j'} \leqq \frac{1}{n-1} \cdot c^{4k+2} \cdot \#A_i \leqq \frac{1}{n^{2\varepsilon}} \cdot \#A_i,$$

if we choose $k \leqq \gamma' \log(n)$ for some suitably chosen $\gamma' > 0$. Note that $\varepsilon < \frac{1}{2}$, thus $1 - 2\varepsilon > 0$ and we may choose

$$\gamma' \approx \tfrac{1}{4}(1-2\varepsilon) \log(c) > 0.$$

Here we assume that each F^j can be simulated with less then $\gamma' \log(n)$ steps.

As we know that only surviving processors from A_i can create members of A_i', we know that only processors of $B^{j'}$ can do so if $F^{j'}$ is simulated. We may choose $\gamma'' > 0$ such that $\#U_{\lfloor \gamma'' \log(n) \rfloor}(P) \leqq n^{\varepsilon}$ for each processor P of M. Let $\gamma := \min\{\gamma', \gamma''\}$. Now assume that $F^{j'}$ is simulated with less than $\gamma \log(n)$ steps. As $\gamma \leqq \gamma'$ we know from above that $\#B^{j'} \leqq \#A_i / n^{2\varepsilon}$. As $\gamma \leqq \gamma''$ we know that each member of $B^{j'}$ can only create n^{ε} elements of A_i'. Thus $\#A_i' \leqq \#B^{j'} \cdot n^{\varepsilon} \leqq \#A_i / n^{\varepsilon}$ which proves the claim. □

We now can define the computation step F demanded in Lemma 11. If there are $i, j \in \{1, \cdots, n\}$, $i \neq j$, such that the simulation of the computation step which only has the edge (i, j) needs at least $\gamma \log(n)$ steps, then this is F. Otherwise, let F contain all those edges (i, j'), $i \in \{1, \cdots, n\}$, where j' is defined for i in the claim. Now if M simulates F faster than $\gamma \log(n)$, we know that for the sets of representatives $A_1', \cdots, A_n'$ after this simulation, $\#A_i' \leqq \#A_i / n^{\varepsilon}$ for each $i \in \{1, \cdots, n\}$. Thus $\sum_{i=1}^{n} \#A_i' \leqq 1/n^{\varepsilon} \cdot \sum_{i=1}^{n} \#A_i$, which proves the lemma. □

Proof of Theorem 10. Consider the computation $F_1, F_2, \cdots$ which is defined step by step by Lemma 11. Let l be an integer and for $t \in \{0, \cdots, l\}$ let h_t denote the number of representatives at time t. Let $k_1, \cdots, k_l$ be the t-time losses of the simulation of $F_1, \cdots, F_l$ and let $S \subset \{1, \cdots, l\}$ be the set of those indices t for which k_t is smaller than $\gamma \log(n)$, $s := \#S$. Then

$$(*) \qquad \sum_{t=1}^{l} k_t \geqq \sum_{t \in S}^{l} k_t \geqq (l-s)\gamma \log(n).$$

It remains now to bound s. We know that $h_0 = n$, $h_t \geqq n$ for each $t \in \{1, \cdots, l\}$. Furthermore we know by Lemma 11 that during the simulation the number of representatives is decreased s times by a factor of at least $1/n^{\varepsilon}$, namely during the simulation of each

tth step with $t \in S$. On the other hand it is at most $(l-s)$ times increased by a factor of at most c^{k_t+1} for each $t \in \{1, \cdots, l\} \setminus S$. Thus we may conclude

$$n \leqq h_l \leqq h_0 \cdot \left(\frac{1}{n^{\varepsilon}}\right)^{s} \cdot \prod_{\substack{t=1 \\ t \notin S}}^{l} c^{k_t+1} \leqq \frac{1}{n^{\varepsilon \cdot s-1}} \cdot c^{l+\sum_{t=1}^{l} k_t}.$$

If now $\sum_{t=1}^{l} k_t > l \cdot (\varepsilon \cdot \log_c (n)/2 - 1)$ the theorem is proved.

Otherwise $n \leqq n^{(\varepsilon \cdot l)/2}/n^{\varepsilon \cdot s-1}$ which implies $n^{\varepsilon \cdot s} \leqq n^{(\varepsilon \cdot l)/2}$. Thus $s \leqq l/2$.

Inserting this bound in the inequality (*) from above, we obtain $\sum_{t=1}^{t} k_t \geqq l \cdot (\gamma/2) \log (n)$, which proves the theorem. □

REFERENCES

[1] M. AJTAI, J. KOMLOS AND E. SZEMEREDI, *An $O(n \log (n))$ sorting network*, Proc. 15th Annual ACM Symposium on Theory of Computing, Boston, 1983, pp. 1-9.

[2] K. BATCHER, *Sorting networks and their applications*, AFIPS Spring Joint Computing Conference, 32 1968, pp. 307-314.

[3] Z. GALIL AND W. J. PAUL, *A general purpose parallel computer*, J. Assoc. Comput. Mach., 30 (1983), pp. 360-387.

[4] F. MEYER AUF DER HEIDE, *Efficiency of universal parallel computers*, Acta Informatica, 19 (1983), pp. 269-296.

[5] F. P. PREPARATA AND J. VUILLEMIN, *The cube-connected cycles: a versatile network for parallel computation*, Comm. ACM, 24 (1981), pp. 300-310.

[6] J. H. REIF AND L. G. VALIANT, *A logarithmic time sort for linear size networks*, 15th Annual ACM Symposium on Theory of Computing, Boston, 1983, pp. 10-16.

[7] E. UPFAL, *Efficient schemes for parallel communication*, Proc. ACM Symposium on Principles of Distributed Computing, Ottawa, 1982.

[8] L. G. VALIANT AND G. J. BREBNER, *Universal schemes for parallel communication*, Proc. 13th Annual ACM Symposium on Theory of Computing, Milwaukee, WI, 1981, pp. 263-267.

[9] A. WAKSMAN, *A permutation network*, J. Assoc Comput. Mach., 15 (1968), pp. 159-163.

[10] U. VISHKIN, *A parallel-design-distributed-implementation* (PDDI) *general purpose computer*, Technical Report No. 96, Dept. Computer Science, New York Univ., New York, 1983.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
009

AN $O(EV \log V)$ ALGORITHM FOR FINDING A MAXIMAL WEIGHTED MATCHING IN GENERAL GRAPHS*

ZVI GALIL†, SILVIO MICALI‡ AND HAROLD GABOW§

Abstract. We define two generalized types of a priority queue by allowing some forms of changing the priorities of the elements in the queue. We show that they can be implemented efficiently. Consequently, each operation takes $O(\log n)$ time. We use these generalized priority queues to construct an $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs.

Key words. matching, augmenting path, blossoms, generalized priority queues, primal dual algorithm, time complexity

Introduction. We are given a graph $G=(V, E)$ with vertex set V and edge set E. Each edge $(i, j) \in E$ has a weight w_{ij} associated with it. A *matching* is a subset of the edges, no two of which have a common vertex. We want to find a matching with the maximal total weight.

In this paper we deal with the general problem. There are three restricted versions of the problem: we can restrict attention to bipartite graphs, or to maximizing cardinality (unit weights) or both. For a survey on the status of the four versions of the problem see [5]. In the time bounds mentioned below we use V and E for the size of the corresponding sets. No confusion will arise.

Edmonds [3] gave the first polynomial time algorithm to the problem, whose time bound is $O(V^4)$. Lawler [8] and independently Gabow [4] improved Edmonds' algorithm by finding a way to implement it in $O(V^3)$..

We develop an $O(EV \log V)$ algorithm, which is much better for sparse graphs. We note that for the problem of finding a maximal flow in networks, a number of efficient algorithms for sparse graphs have been developed in recent years ([6], [9]), while an $O(V^3)$ algorithm has been known for some time [7]. Our algorithm is also an implementation of Edmonds' algorithm.

Our improvement is derived from some simple observations on data structures. We design two generalized types of a priority queue by allowing some forms of changing the priorities of the elements in the queue. We show that each operation on these priority queues can still be implemented in time $O(\log n)$, where n is the total number of elements.

In § 1 we define the two types of priority queues. In § 2 we show how to implement each operation on these priority queues in time $O(\log n)$. In § 3 we review the notions of augmenting paths and blossoms. In § 4 we describe our version of Edmonds' algorithm. We leave out some details of the implementation. In § 5 we show how a straightforward implementation yields an $O(EV^2)$ algorithm. (Edmonds' bound was

* Received by the editors June 16, 1983, and in revised form May 15, 1984.

† Computer Science Department Tel-Aviv University, Ramat Aviv, Tel-Aviv, Israel, and Computer Science Department, Columbia University, New York, New York 10027. The research of this author was supported in part by the National Science Foundation under grant MCS78-25301 at the University of California at Berkeley, by the Israel Commission of Basic Research, and by the National Science Foundation under grant MCS-8303139 at Columbia University.

‡ Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02138. The research of this author was supported in part by DARPA under grant N00039-82-C-0235 and by the National Science Foundation under grant MCS82-04506 at the University of California at Berkeley.

§ Computer Science Department, University of Colorado, Boulder, Colorado 80309. The research of this author was supported by the National Science Foundation under grant MCS-8302648.

$O(V^4)$.) Then we show the changes needed to obtain an $O(V^3)$ bound, yielding (a more complete version of) the algorithm by Lawler [8]. In § 6 we show how to use the generalized priority queues to obtain the $O(EV \log V)$ algorithm.

1. Generalized priority queues. A *priority queue* [1] or a p.q. in short is an abstract data structure consisting of a collection of *elements*, each with an associated real valued *priority*. Three operations are possible on a p.q.:

(1) insert an element i with priority p_i;

(2) delete an element; and

(3) find an element with the minimal priority.

An implementation of a p.q. is said to be *efficient* if each operation takes $O(\log n)$ time where n is the number of elements. Many efficient implementations of p.q.'s are known, e.g., 2-3 trees [1].

In p.q.'s elements have *fixed* priorities. We consider here the following question. What happens if we allow the priority of the elements to change? Obviously, an additional operation which changes the priority of one element can be easily implemented in time $O(\log n)$. On the other hand, it is not natural to allow arbitrary changes in an arbitrary subset of the elements in one operation simply because one has to specify all these changes.

We introduce two generalized types of p.q.'s which we denote by p.q.$_1$ and p.q.$_2$. The first simply allows a uniform change in the priorities of *all* the elements currently in it. The second allows a uniform change in the priorities of an easily specified subset of the elements.

More precisely, p.q.$_1$ enables the following additional operation:

(4) subtract from the priorities of all the *current* elements some real number δ.

This type of p.q. is not new. A version of p.q.$_1$ was used by Tarjan [10]. Note that in (4) we can add instead of subtract. In our case we will mostly subtract $\delta > 0$.

To define p.q.$_2$ we first need some assumptions. We assume that the elements are partitioned into groups. Every group is either *active* or *nonactive*. An element is active (or not) if its group is. We assume that the elements are totally ordered. By splitting a group according to an element i we mean to create two groups from all the elements in the group greater (not greater) than i. Note that unlike the usual split operation we split a group according to an element and not according to its priority.

The operations possible for p.q.$_2$ are:

(1)′ insert an element i with priority p_i to one of the groups;

(2)′ delete an element;

(3)′ find an *active* element with the minimal priority;

(4)′ decrease the priorities of all the *active* elements by some real number δ;

(5)′ generate a new empty group (active or not);

(6)′ delete a group (active or not);

(7)′ change the status of a group from active to nonactive or vice versa; and

(8)′ split a group according to an element in it.

In § 6 we use p.q.$_1$ and p.q.$_2$ to obtain an improved algorithm for finding a maximal weighted matching in general graphs.

2. An efficient implementation for p.q.$_1$ and p.q.$_2$. It may look at first that one may need up to n steps to update all the priorities as a result of one change. However, it is possible to implement efficiently p.q.$_1$ and p.q.$_2$. In particular, the change of priorities will be achieved implicitly by *one* operation.

p.q.$_1$ can be easily simulated by a conventional p.q. We maintain $\Delta = \sum \delta$, where the sum is over all changes δ so far. In the p.q. we use *modified priorities* which are

computed when elements are inserted into the p.q. The modified priority of i is $p_i+\Delta$. So when an element is inserted we add Δ to its priority. The nice property of the modified priority is that, unlike the original priority, it *does not* change. Tarjan's implementation [10] is more complicated because he also allows merging of p.q.'s. Instead of storing priorities he maintains differences of priorities.

The efficient implementation of p.q.$_2$ is less straightforward. Each group g has a p.q. A_g corresponding to it, and each element has its modified priority. However, the modification is not the same for all the elements. If i is inserted into group g, then its modified priority is set to $p_i+\Delta_g$, where $\Delta_g=\sum\delta$, and the sum is over the changes made when g was a part of an active group (possibly g itself). As for p.q.$_1$, these modified priorities do not change. To update Δ_g we maintain Δ_g^{last}, which is the value of Δ when g was last considered (in operations (1)′, (2)′, (7)′, or (8)′). Whenever we consider an active group g, before resetting its Δ_g^{last} we update Δ_g as follows: $\Delta_g \leftarrow \Delta_g+\Delta-\Delta_g^{\text{last}}$. When we split a group g to groups g_1 and g_2 we set $\Delta_{g_i}, \Delta_{g_i}^{\text{last}} \leftarrow \Delta_g$ for $i=1,2$. We also maintain a p.q.$_1$ B which contains one element with the minimal priority from every active group.

Implementing the first seven operations is quite easy. Note that an insert to or a delete from A_g may require an insert to or a delete from B (or both). Note also that if i is in group g and its modified priority which is stored in A_g is q_i $(=p_i+\Delta_g)$, then if it is inserted to B, the modified priority in the p.q. that implements B should be $q_i-\Delta_g+\Delta$. To efficiently implement a split one needs to make a key observation on 2-3 trees. We need the observation because, unlike conventional p.q.'s, we split according to an element and not its priority.

In [1] two kinds of priority queues are described. In the first kind the elements are stored in the leaves and each internal node contains the smallest element of the two (or three) subtrees rooted at his sons. In the second kind the elements are stored in the leaves, and in addition the order is preserved; i.e. the smallest element is stored in the leftmost leaf, etc. This kind supports the operations of concatenate and split. Such priorities queues are called *concatenable queues.*

In our case we have two order relations: the priorities and the order of the elements. Fortunatey, the same 2-3 tree can support both. It contains the information of the first kind for handling the priorities, and of the second kind to handle the order of the elements. The ability to handle both simultaneously is the result of the following observation: assume we treat our 2-3 trees as being of the second type; we split them or concatenate them. If we visit and possibly make changes in a node, we also visit all its ancestors in the tree up to the root. These are exactly all the nodes that may be affected and have to be updated if the tree is of the first kind. For more details on the various operations see [1].

3. Blossoms and their representation. We assume that we are given a graph referred to as the *original graph,* and a matching M. The algorithm discovers certain sets of vertices (of odd size) called *blossoms* and shrinks them. It is convenient to consider also the vertices of the graph as (trivial) blossoms of size one. Consequently, at any moment the blossoms constitute the vertices of the *current graph.*

An *alternating path* from a vertex u_0 to a vertex u_r in the original graph is a sequence of edges $\{e_i=(u_{i-1},u_i)\}_{i=1}^r$ such that $u_1,\cdots,u_r$ are distinct and for $i=1,\cdots,r-1$, $e_i\in M$ iff $e_{i+1}\notin M$. An *alternating path* from a blossom B_0 to a blossom B_r (possibly $B_0=B_r$) is a sequence of edges $\{e_i=(u_{i-1},v_i)\}_{i=1}^r$ such that for $i=0,1,\cdots,r$ u_i, $v_i\in B_i$ where $B_1,\cdots,B_r$ are distinct blossoms and for $i=1,\cdots,r-1$, $e_i\in M$ iff $e_{i+1}\notin M$. When the algorithm discovers an alternating path of odd length

$\{e_i=(u_{i-1}, v_i)\}_{i=1}^{r}$ (r odd) from a blossom B_0 to itself ($B_0=B_r$; $e_1, e_r \notin M$), a new blossom B is formed. The blossoms $B_1, \cdots, B_r$ stop being blossoms and are referred to as the subblossoms of B. Consequently, at any time each vertex is in a unique blossom.

Each blossom has a *base* vertex. The base of a trivial blossom is the unique vertex in it. The base of the blossom B defined above is the base of B_r. Note that if b is the base of B and c is a vertex in B then $(b, c) \notin M$. Also if u is in B and is not the base of B, then there is a v in B such that $(u, v) \in M$ and for every w not in B $(u, w) \notin M$.

A nontrivial blossom is represented by the doubly linked list $\{(B_i, e_i)\}_{i=1}^{r}$ and by its base. Note that

Fact 1. For every $1 \leqq i \leqq r-1$, $(e_1, e_2, \cdots, e_i)$ and $(e_r, e_{r-1}, \cdots, e_{i+1})$ are alternating paths from B_0 to B_i. One is of odd length and one of even length. The one of even length is the one whose last edge is in M.

An easy induction on the structure of the blossom implies

Fact 2. In the original graph, there is an even length alternating path from the base of the blossom to any vertex in it.

A vertex i is *matched* if there is an edge (i, j) in M, and is *exposed* otherwise. A blossom is matched (exposed) if its base is. Edges in M are said to be matched. An *augmenting path* is an alternating path between two exposed vertices (blossoms). By Fact 2, any augmenting path between two exposed blossoms can be expanded to an augmenting path in the original graph between the two (exposed) bases of these blossoms.

One can define a tree that represents the structure of a blossom. In this tree $B_1, \cdots, B_r$ are the sons of B, and the leaves are the vertices of the blossom. We call it the *structure tree*. This tree is implicitly represented by the lists $\{(B_i, e_i)\}_{i=1}^{r}$. The tree implies a total order on the vertices of the blossom: $u < v$ if u is to the left of v in the tree. Note that the base of a blossom is its largest vertex.

Although we conceptually consider the blossoms shrunk, we do not actually shrink them. Edges (u, v) retain their identity. So u and v may belong to blossoms but the edge remains (u, v). If we use such an edge and reach a vertex v we will need to find the blossom of v. So in addition we represent blossoms as ordered sets of vertices. The operations that we need are find, concatenate and split [1].

4. The algorithm.

4.1. A sketch of the algorithm. The algorithm applies the primal-dual method [8]. At any moment we have a matching M and an assignment of values to the dual variables: u_i for every vertex i, and z_k for every odd subset B_k of vertices, $|B_k| = 2r_k + 1$, $r_k > 0$. As will be explained below, it is not important to know what is the meaning of the dual variables.

For every edge (i, j) we define

$$\pi_{ij} = u_i + u_j - w_{ij} + \sum_{k:\, i,\, j \in B_k} z_k.$$

By duality theory (see [8]), the matching has maximum weight if (0)-(3) hold for every vertex i, edge (i, j), and odd subset B_k:

(0) $u_i, \pi_{ij}, z_k \geqq 0$;

(1) $(i, j) \in M \Rightarrow \pi_{ij} = 0$;

(2) i exposed $\Rightarrow u_i = 0$; and

(3) $z_k > 0 \Rightarrow B_k$ is full ($|\{(i, j) | i, j \in B_k, (i, j) \in M\}| = r_k$).

In fact, we need duality theory for motivation only. The following short proof implies that if (0)-(3) hold, then the matching M has maximal weight: let u_i, z_k and

π_{ij} be the values associated with M, and let N be any other matching. Then

$$\sum_{(i,j)\in N} w_{ij} \leqq \sum_{(i,j)\in N} (u_i + u_j) + \sum_{(i,j)\in N} \sum_{k:i,j\in B_k} z_k \leqq \sum_i u_i + \sum_k r_k z_k = \sum_{(i,j)\in M} w_{ij}.$$

The first inequality follows from $\pi_{ij} \geqq 0$; the second from u_i, $z_k \geqq 0$ and the fact that N is a matching; and the equality follows from (2), (3) and the fact that M is a matching.

The algorithm will have $z_k > 0$ only for blossoms B_k. Consequently the number of positive z_k's will be small ($O(V)$). Moreover, (3) will hold automatically.

We start with $M = \Phi$ and $u_i = (\max_{k,l} w_{k,l})/2$ for all i and no blossoms (and no z_k's). So except for (2) all other conditions for optimality hold. The algorithm makes changes that preserve (0), (1), (3) and eventually reduce the number of violations of (2) to zero. The resulting matching therefore has maximal weight.

4.2. The search. The main part of the algorithm consists of a search for an augmenting path between two exposed blossoms. The search uses only edges (i, j) with $\pi_{ij} = 0$. During the search, blossoms are labeled by S and T, where an S (T) label denotes an even (odd) length alternating path from an exposed blossom. (Other papers use outer and inner for S and T.) A blossom labeled by S (T) is referred to as an *S-blossom* (a *T-blossom*). A vertex in an S-blossom (a T-blossom) is an *S-vertex* (a *T-vertex*). We also have free blossoms—those without a label, and free vertices—those in free blossoms. During the search new (S) blossoms can be generated. The search may lead to the discovery of an augmenting path. In this case the matching is augmented and we have two less exposed vertices and consequently two less violations of (2). After an augmentation all the labels are erased. So, all blossoms become free. Each augmentation terminates a *stage.*

Initially all exposed blossoms are labeled S. Then the search uses *useful* edges to label more blossoms. A useful edge is an unmatched edge (i, j) with $\pi_{ij} = 0$, i an S-vertex and j is either a free vertex (Case 1) or an S-vertex in a blossom different from the blossom of i (Case 2).

Case 1. j is in a free blossom B with base b. In this case B is labeled with $[T, (i, j)]$. There must be an edge in M of the form (b, c) (otherwise B would be labeled by S). Assume c is in a blossom C. C must be free because we always use immediately the edge in the matching. (It cannot be labeled S because an S label arrives always through a matched edge, so it could arrive only through (b, c). It cannot be labeled by T because if C were labeled by T, we would have immediately labeled B by S.) We label C by $[S, (b, c)]$. The second part of the label records the edge through which it has arrived. In the case of an S label, this part is redundant because c is the base of C and (b, c) is the unique edge in M that is incident with c.

Case 2. j is in an S-blossom B, i is in an S-blossom $C \neq B$.

Using the second part of the labels, we backtrack along the two paths from exposed blossoms to B and to C. If the exposed blossoms are different, an augmenting path has been found. If they are the same, a new blossom is discovered.

If we discover an augmenting path between two exposed blossoms, we first change the status of the edges on the path (from matched to unmatched and vice versa). Consider a blossom B on this path and the two edges $e \in M$ and $e' \notin M$ incident with it. The first enters b, the base of B, and the second leaves through some vertex c that is in some subblossom B_i of B. (See Fig. 1.) We recursively find the even length alternating path in B from b to c (guaranteed by Fact 2) and change the status of its edges: Using the list of subblossoms of B and Fact 1, we find the alternating path through the subblossoms of B ($e_1, \cdots, e_i$ or $e_{i+1}, \cdots, e_k$) of even length. We change the status of the edges on this even length path. We also change the base of B to c

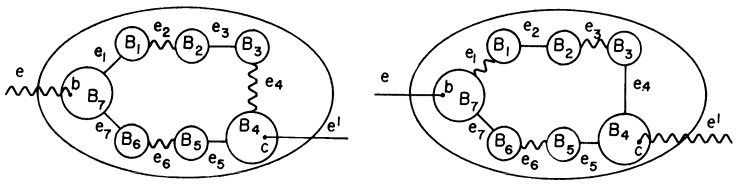


FIG. 1. *Recursively finding the augmenting path. Matched edges are drawn wiggly.* (a) *Before the augmentation in a blossom B. The base is b and the list of subblossoms* $\{(B_1, e_1), \cdots, (B_7, e_7)\}$. (b) *After the augmentation in B. The base is c and the subblossom list is* $\{(B_3, e_4), (B_2, e_3), (B_1, e_2)(B_7, e_1), (B_6, e_7), (B_5, e_6), (B_4, e_5)\}$.

and cyclically permute the list of subblossoms of B (so B_i is now last). We continue recursively with the subblossoms along this even length path. The parts of the alternating paths inside the two exposed blossoms are found similarly.

In case the backtracking leads to the same exposed blossom, we find the first common blossom D on the two paths. We use the parts of the paths from D to B and to C to generate the list $\{(B_i, e_i)\}_{i=1}^{r}$ for the new blossom. $B_r = D$ and e_i are taken from the two paths. We initialize the dual variable associated with the new blossom to 0, and label the new blossom by S.

During the search we choose any useful edge and act according to the case we are in. As a result, some useful edges may stop being useful and some edges may become useful. The search may succeed (if we find an alternating path) or fail (if there are no more useful edges).

4.3. A change in the dual variables. If the search fails, we make the following changes in the dual variables. We choose $\delta > 0$ and execute:

(a) $u_i \leftarrow u_i - \delta$ for every S-vertex i;

(b) $u_i \leftarrow u_i + \delta$ for every T-vertex i;

(c) $z_k \leftarrow z_k + 2\delta$ for every S-blossom B_k; and

(d) $z_k \leftarrow z_k - 2\delta$ for every T-blossom B_k.

Such a choice of δ preserves (1) and (3). To preserve (0) we choose $\delta = \min(\delta_1, \delta_2, \delta_3, \delta_4)$, where

$$\delta_1 = \min_{i:\ S\text{-vertex}} u_i$$

$$\delta_2 = \min_{\substack{(i,j)\in E \\ i:\ S\text{-vertex} \\ j:\ \text{free vertex}}} \pi_{ij}$$

$$\delta_3 = \min_{\substack{(i,j)\in E \\ i,j:\ S\text{-vertices not in the same blossom}}} (\pi_{ij}/2)$$

$$\delta_4 = \min_{B_k \text{ a } T\text{-blossom}} (z_k/2)$$

Note that $\delta_1 = u_{i_0} = (\max_{k,l} w_{k,l})/2 - \Delta$, where i_0 is any exposed vertex and Δ is the sum of the changes δ so far. This is because initially $u_i = (\max_{k,l} w_{k,l})/2$ for every S-vertex i, and the fact that the exposed vertices were always S-vertices and their u_i's were always decreased by δ. Consequently, if $\delta = \delta_1$, then after the change (2) is satisfied and we have a matching with maximal weight.

If $\delta = \delta_4$, we expand all T-blossoms B_k on which the minimum was attained. (Their z_k becomes 0.) Expanding a blossom B is described in Fig. 2. B stops being a blossom and its subblossoms become blossoms. The label of B is $[T, (p, q)]$ where (p, q) is the edge through which B received its T label. Assume $q \in B_i$, where $B_1, \cdots, B_r$ are the subblossoms of B. The subblossoms on the odd length path from $B_0 = B_r$ to B_i (see Fact 1) except for B_0 and B_i become free. The ones on the even length path get alternating labels starting and ending with T. It is here where we need the split operation. For $i = 1, \cdots, r-1$, we split each B_i from B according to its base which is its largest element. As a result of expanding a T-blossom some edges may become useful. If that is the case we resume the search. Otherwise we make another change of the dual variables.

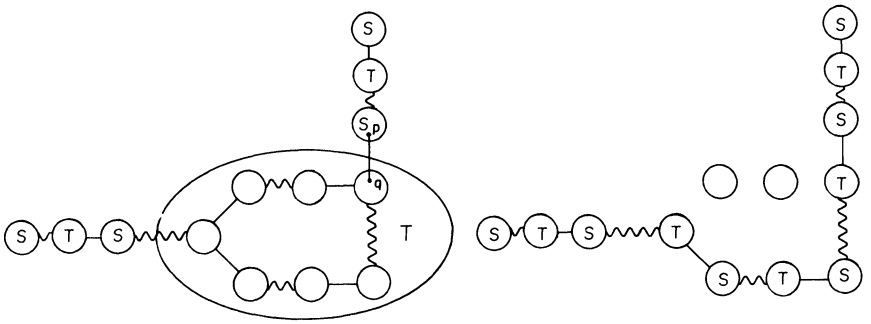


FIG. 2. *Expanding a T-blossom*: (a) *before, and* (b) *after the expansion.*

If $\delta = \delta_2$ ($\delta = \delta_3$), all edges (i, j) with i an S-vertex and j a free vertex (an S-vertex not in the same blossom) on which the minimum was attained become useful (their π_{ij} becomes 0) and we resume the search. The two cases correspond to the two cases in § 4.2.

At the end of each stage we also expand all S-blossoms B_k with $z_k = 0$. Note that finding the alternating path within a blossom can be deferred to the time we expand it. This way we save the repeated changes within the same blossom.

Keeping the blossoms with positive dual variables to the next stage is important. This makes sure that (3) always holds. This explains why T-blossoms can be generated. The latter are expanded whenever their dual variables become 0.

5. The known algorithms. Let us call a *substage* each change in the dual variables. Obviously, there are $O(V)$ stages. There are $O(V)$ different blossoms per stage: each S-blossom corresponds to a unique node in one of the structure trees at the end of a stage. Each T-blossom (free blossom) corresponds to a unique node in one of the structure trees at the beginning of the stage. But, whenever $\delta = \delta_2$ ($\delta = \delta_3$) a new T-blossom (S-blossom) is generated, and whenever $\delta = \delta_4$ a T-blossom is expanded. Hence, $\delta = \delta_i$, $i = 2, 3, 4$, at most $O(V)$ times per stage. Finally, $\delta = \delta_1$ at most once. Consequently, there are $O(V)$ substages per stage.

The most costly part in a substage is finding useful edges and computing δ. The obvious way to do it takes $O(E)$ steps (in each substage we consider all the edges) and yields an $O(EV^2)$ algorithm. To maintain the sets one uses ordered lists for concatenate and split and an array for the find. The naive implementation costs $O(V^3)$. (There are $O(V)$ concatenates and splits per stage, each costs $O(V)$.) The cost of maintaining the dual variables is also $O(V^3)$ ($O(V)$ per substage). The resulting

algorithm is essentially Edmonds' algorithm. The time bound that was given for it was $O(V^4)$ because E was bounded above by V^2.

The only parts which require more than $O(V^3)$ are maintaining δ_2 and δ_3 and finding useful edges. The latter is handled automatically because $\delta_2 = 0$ ($\delta_3 = 0$) iff there are useful edges of Case 1 (Case 2) and these are the edges on which the minimum (0) is attained. We show first how to take care of δ_2. For every free vertex (T-vertex) j let

$$\pi_j = \min_{\substack{(i,j)\in E \\ i:\ S\text{-vertex}}} \pi_{ij}.$$

Then

$$\delta_2 = \min_{j:\ \text{free vertex}} \pi_j.$$

Together with π_j we record an edge (i, j), i an S-vertex, such that $\pi_j = \pi_{ij}$. For each change of δ, we only change π_j for free vertices j. Consequently, the changes of $\{\pi_j\}$ and computing δ_2 cost $O(V^3)$. Recall that free vertices may become T-vertices (when a blossom is labeled by T) and T-vertices may become free (when we expand a T-blossom). That is why we need π_j's for T-vertices as well.

To take care of δ_3, we define for every pair of S-blossoms B_k, B_l,

$$\varphi_{k,l} = \min_{\substack{(i,j)\in E \\ i\in B_k,\, j\in B_l}} (\pi_{ij}/2).$$

We record the edge $e_{k,l}$ on which the minimum is attained and maintain $\varphi_k = \min_l \varphi_{k,l}$. We do not maintain $\varphi_{k,l}$, but any time we need it we compute it by using $e_{k,l}$. Obviously $\delta_3 = \min_k \varphi_k$. The changes in the dual variables and computing δ_3 cost $O(V^3)$ as for δ_2. We have to update $\{\varphi_k\}$ and $\{e_{k,l}\}$ any time an S-blossom B_k is constructed from $B_{i_1}, \cdots, B_{i_r}$. Recall that $(r+1)/2$ of them are S-blossoms and $(r-1)/2$ of them are T-blossoms. We first "make" each T-blossom B_m an S-blossom by scanning all its edges and computing for it $\{\varphi_{m,l}\}$ and $\{e_{m,l}\}$. Then we use the $\varphi_{m,l}$'s of $B_{i_1}, \cdots, B_{i_r}$ to compute φ_k, $\{e_{k,l}\}$ for the new blossom B_k, and to update $\{\varphi_j\}$ for $j \neq k$.

The total cost (per stage) to make T-blossoms S-blossoms is $O(E)$. We now compute $T(n)$, the rest of the cost of maintaining δ_3, where n is the number of S-blossoms plus the number of non S-vertices in the graph. As above, assume that a new S-blossom is constructed from r subblossoms. It follows that $T(n) \leqq crn + T(n-r+1)$ because rn is a bound on the number of $\varphi_{k,l}$'s considered after making the T-blossoms S-blossoms. $T(n) = O(n^2)$ (by induction on n), and the cost of computing δ_3 is $O(V^3)$. The resulting $O(V^3)$ algorithm is essentially a (more complete version of) Lawler's algorithm [8].

6. The $O(EV \log V)$ algorithm. The most costly part of Edmonds' algorithm is the frequent updates of the dual variables, which cause changes in $\{\pi_{i,j}\}$. Note that all the elements that determine each δ_i are decreased by δ for each change in the dual variables.

The new algorithm is also an implementation of Edmonds' algorithm. The high level description of § 4 (including the search, augmenting the matching, the change of dual variables and the resulting changes in the blossoms) is identical. The main difference is in maintaining the δ_i's by generalized priority queues that we describe next.

We maintain δ_1 by a p.q.$_1$. In this p.q. the elements are the S-vertices i and their priorities u_i. We do not need this p.q.$_1$ for computing δ_1, since $\delta_1 = u_{i_0} = (\max_{k,l} w_{k,l})/2 - \Delta$ where i_0 is any exposed vertex and Δ is the sum of the δ's so far. We use a p.q.$_1$ because we need to maintain the u_i's for computing π_{ij} when the edge

(i, j) is considered. For the same reason we maintain another p.q.$_1$ for the u_i's of the T-vertices.

We maintain δ_3 by a p.q.$_1$. The p.q. contains all *good edges* (i, j) with i and j in different S-blossoms as well as some *superfluous edges* (i, j) with i and j in the same S-blossom. The reason for having superfluous edges is that we do not have time to locate them and delete them any time a new S-blossom is constructed. The priority of a good edge (i, j) is $\pi_{ij}/2$.

We maintain δ_4 by a p.q.$_1$. The elements in the p.q. are the T-blossoms B_k and their priority $z_k/2$. We have a similar p.q.$_1$ for the S-blossoms, because we need to maintain their z_k. (At the end of a stage they become free and in the next stage they may become T-blossoms.)

If we try to maintain δ_2 by a p.q.$_1$, we have a difficulty. Consider Fig. 3. Initially there may be a large free blossom B_1. At that time all edges in Fig. 3 should be considered for finding the value of δ_2. B_1 may become a T-blossom. Then these edges are not among those edges that determine δ_2. Later on B_1 may be expanded and one of its subblossoms, B_2, may become free. The latter may later become a T-blossom and so on. A simple implementation requires the consideration of each such edge an unbounded number of times (up to k in Fig. 3).

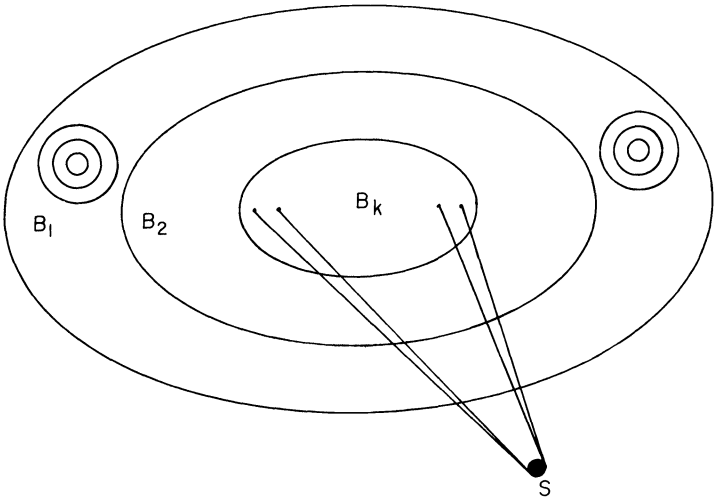


FIG. 3. *Edges from an exposed vertex to the innermost blossom that we may have to consider again and again if the blossoms* $B_1, \cdots, B_k$ *are eventually expanded.*

To maintain δ_2 we have a p.q.$_2$. For every free blossom (T-blossom) B_k we have an active (a nonactive) group of all the edges from S-vertices to vertices in B_k. The priority of an edge (i, j) is π_{ij}. Note that if (i, j) is in a nonactive group (i is an S-vertex and j is a T-vertex), then π_{ij} does not change when we make a change in the dual variables. It is now easy to verify that the eight operations of p.q.$_2$ suffice for our purposes.

Consider a group g which corresponds to a blossom B. The elements of the group are the edges $\{(i, j) | i \text{ an } S\text{-vertex}, j \in B\}$. The order on the elements is derived from the order on the vertices of B. The order between two edges (i_1, j) and (i_2, j) is arbitrary. The order enables us to split the group corresponding to B to the groups corresponding to $B_1, \cdots, B_r$ when we expand B to its subblossoms.

The search is similar to the one described in § 4.2. The labeling process is identical. During the search, whenever we have a new S-vertex i we consider in turn *all* the

edges (i, j). This requires a queue Q for new S-vertices, since we sometimes have many new S-vertices at once. When considering an edge (i, j) we distinguish between 3 cases depending on the type of B the blossom of j.

Case I (II). B is a free blossom (T-blossom). We insert (i, j) with priority π_{ij} to the active (nonactive) group corresponding to B.

Case III. B is an S-blossom. If the blossom of i is not B we insert (i, j) with priority $\pi_{ij}/2$ to the p.q.$_1$ that maintains δ_3.

During the search we compute $\delta = \min(\delta_1, \delta_2, \delta_3, \delta_4)$. If $\delta > 0$, we make a change of δ in the dual variables. This is accomplished by increasing Δ by δ, and results in a new value of $\delta = 0$.

If $\delta = 0$, we consider all $\delta_l = 0$. If $\delta_1 = 0$, then we are done. If $\delta_2 = 0$, this means that the minimum (0) is achieved on an edge (i, j) j in a free blossom B; i.e. (i, j) is useful. We delete (i, j) from the corresponding p.q. and label as in Case 1 of § 4.2. In addition the group corresponding to B becomes nonactive (B is labeled by T) and the group corresponding to C is deleted and the vertices in C (that become S-vertices) are inserted into Q. We repeat the above as long as $\delta_2 = 0$.

If $\delta_3 = 0$ we delete one by one the elements (i, j) in this p.q. with priority $\pi_{ij} = 0$. If i and j are in the same blossom we do not do anything. Otherwise ((i, j) is useful) we act as in Case 2 of § 4.2. If a new S-blossom is generated, then for all the subblossoms B_i that were T-blossoms up till now we delete the group corresponding to B_i (from the p.q.$_2$ of δ_2) and insert all the vertices of B_i to Q.

If $\delta_4 = 0$, we delete one by one the elements B_k in this p.q. with priority $z_k = 0$. For each such B_k, we expand it and label the new blossoms (the previous subblossoms of B_k) as in § 4.3 and Fig. 2. We split the corresponding group in the p.q.$_2$ of δ_2. The groups corresponding to the new free blossoms (T-blossoms) are inserted as active (nonactive) groups to the p.q.$_2$. The vertices of the new S-blossoms are inserted to Q.

To derive an $O(EV \log V)$ time bound we need to implement carefully two parts of the algorithm:

1. We maintain the sets of vertices in each blossom (for finding the blossom of a given vertex) by concatenable queues [1]. Note that the number of finds, concatenates and splits is $O(E)$ per stage.

2. Assume we consider an edge (i, j) where both i and j are S-vertices not in the same blossom. If we execute the backtracking as described above, we may need up to $O(V^3)$ time. Instead, we make a careful backtrack by backtracking one blossom on both paths each time, marking the blossoms on the way. If there are r subblossoms in the new blossom, then we will visit at most $2r$ blossoms before discovering the first common blossom on both paths (D). So the total number of blossoms that we traverse in one stage is $O(V)$. (Charge 2 each one of the corresponding nodes in the corresponding structure tree.)

The time bound is easily derived as follows. There are at most V augmentations. Between two augmentations we consider each edge at most twice and have $O(E)$ operations on (generalized) p.q.'s. (This includes 1 and 2 above.)

Note added in proof. The $Q(EV \log V)$ algorithm for finding weighted matching in general graphs has been recently improved (slightly) to $O(EV \log\log\log_d V + V^2 \log V)$, where $d = \max(E/V, 2)$ [11]. This time bound equals $O(EV)$ if $E = \Omega(V^{1+a})$, for any $a > 0$ and consequently is $o(EV \log V)$ unless $E = \Omega(V^2)$ and is $o(V^2)$ unless $E = O(V)$. The new algorithm is similar to the $O(EV \log V)$ algorithm. The main difference is the use of new data structures instead of regular p.q.'s in the p.q.$_1$'s and p.q.$_2$.

REFERENCES

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

[2] J. EDMONDS, *Path, trees and flowers*, Canad. J. Math., 17 (1965), pp. 449-467.

[3] ———, *Maximum matching and a polyhedron with* 0, 1 *vertices*, J. Res. NBS 69B (April-June 1965), pp. 125-130.

[4] H. N. GABOW, *Implementation of algorithms for maximum matching on nonbipartite graphs*, Ph.D. thesis, Stanford Univ., Stanford, CA, 1974.

[5] Z. GALIL, *Efficient algorithms for finding maximal matching in graphs*, Tech. Rep., Dept. Computer Science, Columbia Univ., New York, 1983.

[6] Z. GALIL AND A. NAAMAD, *An* $O(EV \log^2 V)$ *algorithm for the maximal flow problem*, J. Comput. System Sci., 21 (1980), pp. 203-217.

[7] A. V. KARZANOV, *Determining the maximal flow in a network by the method of preflows*, Soviet Math. Dokl., 15 (1974), pp. 434-437.

[8] E. L. LAWLER, *Combinatorial Optimization*: *Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[9] D. D. SLEATOR, *An* O $(mn \log m)$ *algorithm for maximum network flow*, Ph.D. thesis, Stanford Univ., Stanford, CA, December 1980.

[10] R. E. TARJAN, *Finding optimum branchings*, Networks, 7 (1977), pp. 25-35.

[11] H. N. GABOW, Z. GALIL AND T. H. SPENCER, *Efficient implementation of graph algorithms using contractions*, Proc. 25th IEEE Symposium on Foundations of Computer Science, 1984, pp. 347-357.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
010

OPTIMAL TERMINATION PROTOCOLS FOR NETWORK PARTITIONING*

FRANCIS CHIN† AND K. V. S. RAMARAO‡

Abstract. We address the problem of maintaining the distributed database consistency in presence of failures while maximizing the database availability. Network Partitioning is a failure which partitions the distributed system into a number of parts, no part being able to communicate with any other. Formalizations of various notions in this context are developed and two measures for the performances of protocols in presence of a network partitioning are introduced. A general optimality theory is developed for two classes of protocols—centralized and decentralized. Optimal protocols are produced in all cases.

Key words. commit protocols, consistency, database availability, distributed databases, fault-tolerance, network partitioning, optimal protocals, transaction processing

1. Introduction. A database DB consists of a collection of *entities* $D=\{d_1, d_2, \cdots, d_m\}$ such that each d_i in D has a *value set* V_i associated to it, and a set R of relations r on $\times_{i=1}^{m} V_i$ which we call *consistency constraints* for DB. An *instance* of the database DB is an element from $\times_{i=1}^{m} V_i$. An instance $(v_1, v_2, \cdots, v_m)$ of DB is *consistent* if and only if $(v_1, v_2, \cdots, v_m)$ is in r for all r in R.

User programs map the set of database instances into itself. We are primarily interested in those programs that map the set of consistent database instances into itself. Executions of such programs are known as *transactions* and play the central role in database literature [10], [12]. Formally, a transaction t is an execution of a (user) program, $t:\times_{i=1}^{m} V_i \to \times_{i=1}^{m} V_i$ such that for any consistent instance I, $t(I)$ is also consistent. If only a sequence of transactions is allowed to operate on a consistent database instance, then the final instance is also guaranteed to be consistent. We require that all transactions are ensured that the instances they are going to operate on are consistent. This guarantees the database consistency, without any need for an explicit validation of the consistency constraints. Several mechanisms are available which guarantee that any transaction sees an initial consistent instance and can map the set of consistent DB instances into itself even if several transactions are being concurrently run. See [2], [3] for a comprehensive survey on this area.

Without loss of generality, assume that, if $O_1, O_2, \cdots, O_k$ is the sequence of operations in a transaction, then no subsequence $O_1, O_2, \cdots, O_p$, $1 \leqq p < k$, guarantees the resulting database instance to be consistent. At the implementation level, this would imply that, given a consistent instance I_1 of DB, a transaction t acting on I_1 would lead to a consistent instance I_2 if and only if either t does not modify I_1 or all modifications made by t are incorporated onto I_1. Such an implementation of a transaction is known as *atomic* implementation [12]. Thus, a transaction can legally be completed in only one of two possible modes—either it is *committed* in which case all its effects are incorporated into the database instance, or it is *aborted* in which case none of its effects are incorporated.

A *distributed system* is an undirected connected graph $G=(V, E)$. Each node represents a *site* consisting of a processor and possibly storage and other modules. Each edge represents a *communication link*. A database is said to be *distributed* if it physically resides at more than one site. That is, each site contains a subset of the database entities. We make no assumptions on the degree of replication—each entity

* Received by the editors November 29, 1982, and in revised form August 25, 1984.
† Department of Computer Studies, University of Hong Kong, Pokfirlam Road, Hong Kong.
‡ Department of Computer Science, University of Pittsburgh, Pittsburgh, Pennsylvania 15260.

d_i resides at a number of sites n_i, $1 \leqq n_i \leqq |V|$. A transaction is said to be distributed if it is physically run at more than one site. A site at which a distributed transaction is executed is known as a "participating site" for that transaction. It can be seen that a distributed transaction can be atomically implemented if and only if either all participating sites commit it or all of them abort it. Communication among the participating sites is required to guarantee this condition and such *protocols* are known as *commit protocols* [10]. Thus, an execution of a commit protocol is associated with each transaction. Without loss of generality, assume that each transaction t has all sites in the distributed system participating in its execution.

A number of possible failures could occur in a distributed system. "Processor malfunctioning" is one of the most widely studied failures in the literature. See [14] for an informal survey of this topic. "Clean" site failures, where the processor at a site simply stops in case of a fault, are also extensively studied [1], [4], [8], [10], [11], [15]. A third kind of failure, the one we are interested in, is "network partitioning" where the graph G gets partitioned into a number of connected subgraphs. Existing literature on this problem is rather sparse [6], [7], [17].

In practice, commit protocols are not expected to handle failures in the system. Thus, the commit protocol is simply frozen when a fault is detected and a new type of protocol, known as "termination protocol" (TP) is invoked to handle the exception. (Formal definitions of these protocols will be given in § 2.) An execution of the TP directs the termination of an incomplete transaction. We require that the TPs also guarantee the database consistency. A commit protocol P is said to be *nonblocking* [15] to a failure type F if and only if there is a TP associated with P such that any incomplete transaction in presence of an arbitrary instance of F can be consistently completed at all operational sites by that TP. P is blocking otherwise. Obviously, one would prefer nonblocking protocols since they enhance the *availability* of the database.

Our interest in the network partitioning problem stems from the fact that there exists no commit protocol nonblocking to network partitioning [17]. Contributions made in this paper are as follows:

1. Formalization of termination protocols for network partitioning which extracts all the available information into the formalism.
2. Study of the properties of TPs, introducing the notion of "nontrivial" TPs.
3. Characterization of commit protocols allowing nontrivial TPs, recognition of a canonical commit protocol.
4. Introduction of measures for the performance of TPs.
5. Development of an optimality theory for these measures and the design of optimal termination protocols.

Organization of this paper is as follows: Section 2 develops the necessary theoretical background for the study of TPs. Notion of nontrivial TPs is introduced and a characterization theorem is proved. Section 3 presents the optimality results and optimal protocols for a class of protocols known as decentralized protocols. Section 4 deals with the centralized protocols. The effectiveness of centralized and decentralized protocols is compared and it is proved that certain centralized protocols are superior. Section 5 discusses certain consequences of our results and concludes the paper.

2. Formal background. The following model of a distributed transaction is used throughout the paper: a transaction t is initiated at some site in the system and is decomposed into an appropriate number of *subtransactions* such that each site participating in t would run exactly one subtransaction. No assumption is made on how this is accomplished—a simple scheme is for the site where t is originated itself to

compute this. Each participating site receives its subtransaction, runs it concurrently with others and makes a (local) decision of "commit" or "abort." All participating sites then cooperatively make a global "commit" or "abort" decision following a *commit rule.* Finally, they commit or abort the transaction according to the global decision. The only requirement on the commit rule is that the conditions for "commit" and "abort" global decisions partition the set of all possible combinations of local decisions. For instance, a simple commit rule is as follows: if there is a site whose local decision is "abort" then the global decision is "abort"; otherwise it is "commit." The "commit" and "abort" *actions*, once implemented, are *irreversible.* Here, observe that we are making a definite distinction between making a decision and implementing it. In theory, a decision is reversible; but, once it is implemented, it cannot be reversed.

In this model of distributed transaction, commit protocol comes into play in implementing the commit rule. Specifically, its task is to ensure that, a) a global decision according to a commit rule is made on the mode of completion, and b) the same decision is implemented by all sites. We assume that the processors do not exhibit any malicious behavior.

One simple commit protocol is to pool all local decisions at a specific site, let that site apply the commit rule and compute the global decision which can be sent to all other participating sites. Such protocols where a single site coordinates the task of completing a transaction are known as "centralized" protocols. At the other extreme, each site can independently contact all other sites. These are known as "decentralized" protocols. Though several intermediate schemes are possible, results obtained in these two cases can be easily extended to all other cases. In this paper, we study only the centralized and decentralized protocols.

We model the execution of a commit protocol by a collection of finite state automata (FSA), one associated with each participating site [17]. An FSA in a state s reads a set of *messages* from other sites, performs local computation (if any), sends a (possibly empty) set of messages to other sites, and changes its state. Each FSA satisfies the following conditions:

1. It is indeterministic.
2. Each transition is associated with a unique input.
3. The final states are partitioned into two classes—"abort states" Ab and "commit states" Com.
4. There are no transitions from a final state.
5. The state graph is acyclic.

Notation. For any FSA, we denote all nonfinal states that lead only to commit states by Pc, all nonfinal states that lead only to abort states by Pa, and all states that lead to both commit and abort states by N. When necessary, we use the identity of the site as a suffix to distinguish FSAs at different sites. For simplicity, we equate the FSA at a site to the site itself. Thus, we use "site i" to mean "the FSA at site i" when no confusion can arise.

DEFINITION 1. A protocol P represented by a collection of FSAs with the above properties is a *commit protocol* if and only if for every input (from the input alphabet of messages on local and global decisions), the FSAs all reach final states from the same class (either Ab or Com) in the absence of any failures.

2.1. Structure of the FSAs. State graphs of FSAs in a commit protocol are acyclic digraphs by definition. Now, we assume that they are in fact trees. (It is easy to see that any acyclic FSA with the above properties can be converted into a tree FSA, by introducing some dummy states if necessary.) Practical considerations introduce more

structure into the FSAs. One such aspect is the "degree of synchronization" among the FSAs [17]:

DEFINITION 2. Let P be a protocol and $F_1, F_2, \cdots, F_n$ the FSA in P. A *global state* of P, $(s_1, s_2, \cdots, s_n)$, is an element of $S(F_1) \times S(F_2) \times \cdots \times S(F_n)$ where $S(F_i)$ is the set of states in F_i, $1 \leqq i \leqq n$, with the property that there is an input for P on which F_i are concurrently in states s_i, $1 \leqq i \leqq n$.

For any state s in an FSA, let $d(s)$ represent the number of state transitions required to transform the initial state into s. A protocol P is said to be "synchronized within k states" for $k \geqq 1$, if and only if $|d(s) - d(t)| \leqq k$ for all pairs of states s, t in two different FSAs such that there is a global state of P in which both s, t occur. During the execution of a commit protocol, each site sends certain messages out and waits for similar messages or responses to its own messages from certain other sites. Thus, the sites "progress" through the execution of the protocol at approximately the same pace. In fact, no site leads any other site by more than one state transition. Thus, all commit protocols being used are synchronized within one state. For instance, if a site has reached a final state, any other site is either in a final state or in a state adjacent to a final state. Commit protocols synchronized within one state can be justified by the fact that they are inexpensive while no generality is sacrificed in using them.

LEMMA 1. *Let P be a commit protocol synchronized within one state. Then, all its* FSA *are isomorphic.*

Proof. Let F_1, F_2 be two FSAs of P. For any path from the initial state to a final state of F_1, there corresponds a path in F_2 from its initial state to a final state. The lengths of these paths can differ at most by one, due to the synchronization. When they differ, a dummy state can be introduced into the appropriate FSA without changing its behavior. Since there is a unique input associated with each transition, there correspond two different paths in F_2 for any two different paths in F_1. Hence, being trees, the state graphs of F_1, F_2 are isomorphic. Q.E.D.

All FSAs in a protocol can be identical in which case we call it a *uniform protocol.* It is nonuniform otherwise. Observe that the decentralized commit protocols are uniform while the centralized are not.

2.2. Termination protocols. Given that a site i is in state s, consider the set of all possible states any other site j could be in at the same (global) instance. In a uniform commit protocol where the FSAs are synchronized within one state, such possible states are simply the states adjacent to s (assuming the same names for states in different FSAs).

DEFINITION 3. Let P be a commit protocol. The *concurrency set* of a state s at site i, denoted by $R(i, s)$, is the set of all ordered pairs (j, t) such that there is a global state $(s_1, s_2, \cdots, s_n)$ of P satisfying $s = s_i$ and $t = s_j$.

The following property of commit protocols is immediate:

PROPERTY 1. If $s \in \text{Ab} \cup \text{Pa}$, then there does not exist an ordered pair (j, t) in $R(i, s)$ for any $t \in \text{Com} \cup \text{Pc}$ and for any sites i, j, and conversely, if $s \in \text{Com} \cup \text{Pc}$, then there does not exist (j, t) in $R(i, s)$ for any $t \in \text{Ab} \cup \text{Pa}$.

Let Q be the set of states in a uniform commit protocol P. Let J denote the power set of $V \times Q$. Let D be the maximal subset of J such that $S \in D$ if and only if

i) $(i, s) \in S$ and $(i, t) \in S$ implies $s = t$,

ii) (i, s), $(j, t) \in S$ implies $(j, t) \in R(i, s)$ and $(i, s) \in R(j, t)$.

It is not hard to extend the definition of D to nonuniform protocols. The importance of D is based on the following concern: Recall that a network partitioning partitions the graph G into a number of connected subgraphs such that there is no edge joining any two of these subgraphs. This could happen due to the deletion (failure) of certain

nodes (sites) and/or edges (communication links). When such a partitioning is detected, the commit protocol is frozen and each site remains in a well defined state. The entity D defined above provides us with the formal representation of the status of each subgraph immediately after a partitioning is detected. Any set S in D corresponds to a physically realizable subgraph, embedding the status of a transaction being run with the commit protocol P. Notice that all the information available in the physical partitioning has been extracted into this formalism: the first condition represents the fact that no site can be in more than one state at the same time, while the second ensures that the concurrency relations of states under P are preserved. We call such S in D a *component* so that D represents the set of all physically realizable components of the given network under a given commit protocol. For S in D, let site $(S) = \{i | (i, s) \in S$ for some $s \in Q\}$ and state $(S) = \{s | (i, s) \in S$ for some $i \in V\}$. Extending the above definition of concurrency to different components, we say two components S, T are *concurrent* if and only if a) site $(S) \cap$ site $(T) = \emptyset$, and b) for $(i, s) \in S$ and $(j, t) \in T$, $(i, s) \in R(j, t)$ and $(j, t) \in R(i, s)$. The second condition can be restated simply as $S \cup T \in D$. In the following, we first describe the centralized version of the general termination protocol before formally defining it.

When a network partitioning is detected, sites in each component S "elect" a single site as a "coordinator" to execute the termination protocol (TP). (See [9] for some election protocols.) Coordinator collects the identities of the sites and their states, and applies the TP based on this information. Let x, y, z represent the actions to be taken by a TP. The coordinator delays the transaction until a network reconfiguration if the action is z. It directs the other sites to commit (abort) if the action is x (y). The primary requirement for a TP is that it consistently terminates a transaction in presence of faults. The situation is quite involved in case of network partitioning since there can exist a number of components in the system, each trying to complete the incomplete transactions, independent of all other components. In spite of these independent activities, each transaction's atomic implementation and consistency should be guaranteed. The following definition formulates the notion of a TP:

DEFINITION 4. Let P be a commit protocol. Let $f: D \to \{x, y, z\}$ be any function. We say f has *preservation property* if and only if, for any S in D,

i) state $(S) \cap \text{Com} \neq \emptyset$ implies $f(S) = x$,

ii) state $(S) \cap \text{Ab} \neq \emptyset$ implies $f(S) = y$.

f has *commit property* if and only if, for any two concurrent components S, T in D, $\{f(S), f(T)\} \neq \{x, y\}$.

f is a *termination protocol* (TP) if and only if f has both preservation and commit properties.

Observe that the commit and preservation properties are mutually consistent due to Property 1. Notice that we have considered three possible actions x, y, z for a TP, in contrast to only two actions for a commit protocol. This is due to the following previously known result:

THEOREM 1 [17]. *Let P be a commit protocol. Assume that arbitrary network partitionings are possible. Then, for any* TP *f of P, there exists a component S in D such that $f(S) \neq x$ and $f(S) \neq y$.*

Thus, there are occasions on which a component should wait until a reconfiguration. Given the above limitation, we want to optimize the performance of TPs.

The above definition of TPs does not specify how a TP is implemented and how reconfigurations are handled. We leave these details unspecified. We do not require any assumptions in this respect for the purposes of this paper. The interested reader is referred to [13].

The following property of TPs is immediate from the definition.

PROPERTY 2. Let P be a commit protocol synchronized within k states, for $k \geqq 1$. For any FSA, let $Pa_k = \{s \in \text{Pa} \,|\, |d(a) - d(s)| \leqq k \text{ for some } a \in \text{Ab}\}$, $\text{Pc}_k = \{s \in \text{Pc} \,|\, |d(c) - d(s)| \leqq k \text{ for some } c \in \text{Com}\}$. Let f be any TP of P. Then, for any S in D,

i) state $(S) \subseteq (N \cup \text{Pa}_k)$ implies $f(S) \neq x$,
ii) state $(S) \subseteq \text{Pc}_k$ implies $f(S) \neq y$.

2.3. Measuring the performance of TPs. For a given commit protocol, we wish to find the TP which maximizes the number of incomplete transactions that can be completed, when used in conjunction with all possible network partitionings. This is because an incomplete transaction makes its resources unavailable to the other transactions. Since a transaction may be completed in one component and kept incomplete in another, the number of components where transactions are completed needs to be maximized. On the other hand, it is not wise to complete a transaction in a "small" component (containing a few sites) while leaving an incomplete transaction in a large component. Hence, the number of sites should also be considered in assessing the performance of a TP. Given a TP f, let $\text{CM}(f) = |\{S \in D \,|\, f(S) = z\}|$ and $\text{SM}(f) = \sum \{|S|$ where $S \in D$ and $f(S) = z\}$.

DEFINITION 5. Let P be a commit protocol. A TP f of P is *component optimal* in a class of TPs F if and only if $\text{CM}(f) \leqq \text{CM}(f')$ for all f' in F.

DEFINITION 6. A TP f of P is *site optimal* in F if and only if $\text{SM}(f) \leqq \text{SM}(f')$ for all f' in F.

2.4. Nontrivial termination protocols. As the first step towards an optimality theory, we ask: how many TPs does a commit protocol have? Clearly, any commit protocol has at least one TP defined as follows: for $S \in D$, states $(S) \cap \text{Com} \neq \varnothing$ implies $f(S) = x$, state $(S) \cap \text{Ab} \neq \varnothing$ implies $f(S) = y$, and $f(S) = z$ otherwise. This TP is trivial since it satisfies the commit property by doing nothing. We intend to find more effective, "nontrivial" TPs.

For simplicity, we restrict our attention to commit protocols synchronized within one state throughout the remainder of this paper. The results obtained can be extended without much difficulty.

DEFINITION 7. Let P be a commit protocol (synchronized within one state) and f a TP of P. For any FSA of P, let $N_P = \{s \in N \,|\, |d(t) - d(s)| = 1 \text{ for some } t \in \text{Pc} \cup \text{Com}\}$ and $P_N = \{s \in \text{Pc} \,|\, |d(t) - d(s)| = 1 \text{ for some } t \in N\}$. We say f is a *nontrivial* TP (NTP) if one of the following is true: i) There exists S in D such that coordinator $\notin$ site (S) (for centralized P), state $(S) \subseteq N_P$ and $f(S) \neq z$. ii) Let $P_N \neq \varnothing$ for some FSA. There exists S in D such that state $(S) \subseteq P_N$ and $f(S) \neq z$.

This definition extracts the cases where an intelligent behavior is demanded from the TP. Existence of NTPs depends directly on the commit protocol. For instance, consider the following decentralized version of the widely-used two-phase commit protocol [10], [11]:

Step 1. Each site sends its local decision to all other participating sites and receives the local decisions of all other participating sites.

Step 2. Using the commit rule, each site computes the global decision. The transaction is completed accordingly.

Figure 1 is the state diagram for these FSAs. Messages received are shown above the line and messages sent out below the line.

It is not hard to check that this protocol has no NTP. In fact, it can be shown that its only TP is the trivial one. Following is a characterization of the commit protocols having NTPs.

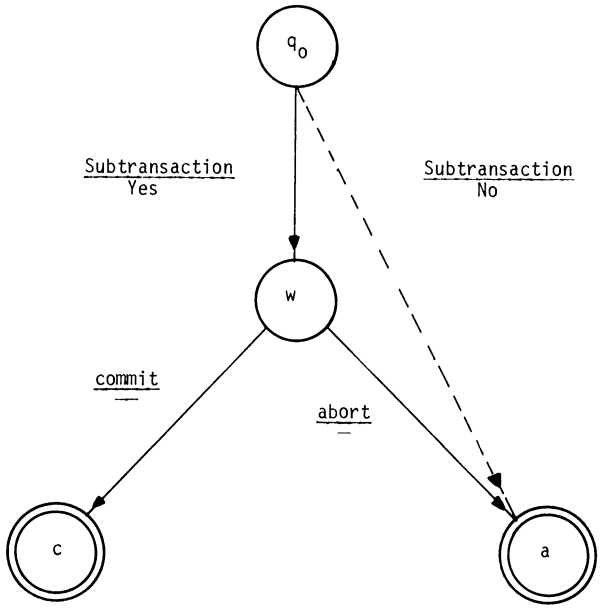


FIG. 1. *Two-phase commit protocol.*

THEOREM 2. *Let P be a commit protocol. Then, P has an* NTP *if and only if* $\mathrm{Pc} \neq \emptyset$ *for an* FSA *of P.*

Proof. Sufficiency—Assume that $\mathrm{Pc} \neq \emptyset$ for some FSA of P. Define $f: D \to \{x, y, z\}$ as follows:

$$\text{for } S \in D, \text{ state } (S) \cap (\mathrm{Com} \cup \mathrm{Pc}) \neq \emptyset \Leftrightarrow f(S) = x,$$
$$\text{state } (S) \cap (\mathrm{Ab} \cup \mathrm{Pa}) \neq \emptyset \Leftrightarrow f(S) = y, \text{ and}$$
$$f(S) = z \text{ otherwise.}$$

Let $S, T \in D$ such that $S \cap T = \emptyset$, $f(S) = x$ and $f(T) = y$. Then, state $(S) \cap (\mathrm{Com} \cup \mathrm{Pc}) \neq \emptyset$ and state $(T) \cap (\mathrm{Ab} \cup \mathrm{Pa}) \neq \emptyset$. But, this implies by Property 1 that $S \cup T \notin D$, proving that f is a TP. Since there is an FSA in which Pc is nonempty by hypothesis, P_N is nonempty for that FSA. Hence, f is an NTP.

Necessity—Assume that $\mathrm{Pc} = \emptyset$ for all FSAs of P, and that there is an NTP f of P. Thus, there exists $S \in D$ such that coordinator $\notin$ site (S), state $(S) \subseteq N_P$ and $f(S) \neq z$. As in Property 2, it can be shown that $f(S) \neq x$. Thus, $f(S) = y$. Consider $S' \in D$ such that site $(S') \subset V$-site (S) and state $(S') \subseteq$ Com. Then, $S \cup S' \in D$ by the hypothesis. Since f is a TP, $f(S') = x$ due to the preservation property. But, this implies that $(f(S), f(S')) = (y, x)$, a contradiction to the commit property of f. Q.E.D.

Consequently, any version of the two-phase commit protocol cannot have an NTP. In view of the above characterization, the simplest commit protocol with NTPs has at least one FSA with the state graph shown in Fig. 2.

Any other commit protocol with NTPs can be considered as an extension of this *canonical* commit protocol. This commit protocol is known in the literature as the "three-phase commit" [15]. For simplicity, we consider only the commit protocol with the above canonical state graph for a detailed study. This can be justified as follows: It is clear from the definition of NTPs that the only states that need special attention are the N_P and P_N states, i.e., "wait" and "commitable" states. Hence, even if a more general protocol is considered, all other states play no critical role in studying the NTPs. Thus, the TPs of the canonical commit protocol can be extended to TPs of more general protocols in a simple fashion.

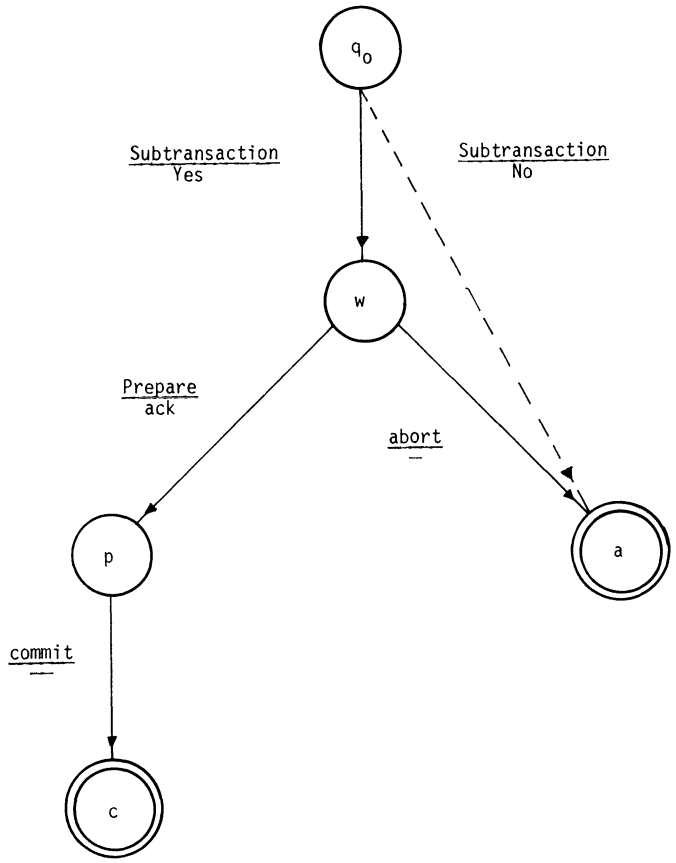


FIG. 2. *Three-phase commit protocol.*

We consider the decentralized and centralized versions of the above canonical commit protocol P: initial state is q, "wait" state is w, "commitable" state is p, "commit" state is c, and "abort" state is a.

3. Decentralized commit protocol.

DEFINITION 8. A version P of the canonical commit protocol is *decentralized* if and only if all FSAs are identical and each site communicates with all other sites.

TPs of the decentralized protocol are referred to as decentralized TPs (DTPs).

LEMMA 2. *Let f be a* DTP *of P. Let S, $T \in D$ such that* state $(S \cup T) \cap (\text{Com} \cup \text{Ab}) = \emptyset$. *Then, $f(S) = x$ and $f(T) = y$ implies that* site $(S) \cap$ site $(T) \neq \emptyset$.

Proof. Assume that the claim is false and that there exist S, T in D with the above properties. Then, state $(S) \not\subseteq N$ and state $(T) \not\subseteq$ Pc by Property 2. This, together with the assumption that site $(S) \cap$ site $(T) = \emptyset$ implies that $S \cup T \in D$, thus violating the commit property of f. Q.E.D.

From the definition of a TP, the above conditions are also sufficient. Hence, we have,

THEOREM 3. *Necessary and sufficient conditions that $f: D \to \{x, y, z\}$ is a* DTP *are,*

i) *f satisfies the preservation property,*

ii) state $(S \cup T) \cap (\text{Com} \cup \text{Ab}) = \emptyset$, $f(S) = x$ *and* $f(T) = y$ *implies that* site $(S) \cap$ site $(T) \neq \emptyset$.

Following are two simple but powerful properties of DTPs which play a critical role in obtaining the optimality results:

LEMMA 3. *Let* $S \in D$ *such that* state $(S) \subseteq N$. *Construct* T *such that* state $(T) \subseteq$ Pc *and* site $(T) \subseteq V$-site (S). *Then, for any* DTP f (*of the canonical commit protocol* P), *either* $f(S) = z$ *or* $f(T) = z$ *or both.*

Proof. Follows immediately from Theorem 3.

LEMMA 4. *Let* $S, T \in D$ *such that* site $(S) =$ site (T), state $(S) \subseteq N$, state $(T) \subseteq$ Pc, $f(S) = y$ *and* $f(T) = x$. *Then,* $f(R) = z$ *for any* R *in* D *such that* site $(R) \subseteq V$-site (S).

Proof. $S \cup R \in D$ and $T \cup R \in D$ for any such R. Owing to the commit property of f, $f(R)$ can neither be x nor y. Q.E.D.

For an intuitive understanding of the above result and the results to follow, let us consider a simple tabular representation for the components. Consider for example the case of three sites 1, 2, 3. Each site represents a column and each row is a vector of states. It is sufficient to consider only the states w and p.

Each row in Table 1 can be interpreted as a component. For instance, row 1 represents the component $\{(3, w)\}$ while row 5 represents the component $\{(1, w), (3, w)\}$. If a TP f maps row (component) 1 to y, then it is easy to see that rows (components) 12, 8, 9 should be mapped to z. Lemma 3 above formalizes this fact. Similarly, if row 1 is mapped to y and row 7 is mapped to x by f, then the rows 2, 3, 6, 8, 9, 12, 17, 18 should all be mapped to z. This is formalized by Lemma 4.

TABLE 1

sites→		1	2	3		1	2	3
states↓	1.			w	10.		p	p
	2.		w		11.	p		p
	3.	w			12.	p	p	
	4.		w	w	13.		w	p
	5.	w		w	14.		p	w
	6.	w	w		15.	w		p
	7.			p	16.	p		w
	8.		p		17.	w	p	
	9.	p			18.	p	w	

We now give a lower bound on the component measure of the TPs, which can be easily appreciated from the above tabular representation.

THEOREM 4. *Let* f *be a* DTP *of* P. *Then,* CM $(f) \geqq 2^n - 2$ *where* $n = |V|$.

Proof. Let $S \in D$ such that state $(S) \subseteq N$. Let $T \in D$ such that site $(T) = V$-site (S) and state $(T) \subseteq$ Pc. Either $f(S) = z$ or $f(T) = z$ by Lemma 3. Clearly, for any S in D such that state $(S) \subseteq N$ and $|S| \leqq n - 1$, there is a T such that either $f(S) = z$ or $f(T) = z$. But, the total number of such S in D is $2^n - 2$. Thus, CM $(f) = |\{S \in D | f(S) = z\}| \geqq 2^n - 2$. Q.E.D.

Observe that Theorem 1 follows simply as a corollary of this lower bound result.

DEFINITION 9. A *quorum protocol* f of P, characterized by an ordered pair (d, e) of positive integers satisfying $d + e > n$, is a function $f: D \to \{x, y, z\}$ such that, a) f satisfies the preservation property, b) state $(S) \cap$ Pc $\neq \varnothing$ and $|S| \geqq d$ implies $f(S) = x$, c) (state $(S) \cap$ Pc $= \varnothing$ OR $|S| <$ d) AND (state $(S) \cap N \neq \varnothing$ and $|S| \geqq e$) implies $f(S) = y$, and d) $f(S) = z$ otherwise.

Each ordered pair of integers (d, e) represents a different quorum protocol. It can be verified using Theorem 3 that any quorum protocol is a DTP. Hereafter, we refer to quorum protocols as QTPs. See [16] for a discussion of quorum protocols used as commit protocols. The following result shows that QTPs exist in pairs in a strong sense:

LEMMA 5. *Let f be a* QTP *characterized by* (d, e). *Then, there exists another* QTP f' *such that* $\mathrm{CM}(f) = \mathrm{CM}(f')$ *and* $\mathrm{SM}(f) = \mathrm{SM}(f')$.

Proof. Consider the QTP f' characterized by (e, d). Observing that for any m, $1 \leqq m \leqq n$, there exist S, T in D such that $|S| = |T| = m$, state $(S) \subseteq N$ and state $(T) \subseteq$ Pc, it is easy to check that the claim is true. Q.E.D.

THEOREM 5. *There is a* QTP g *which is component optimal among all* DTPs *of* P.

Proof. Consider the QTP g characterized by $(1, n)$. Then, for any S in D, $g(S) = z$ if and only if state $(S) \subseteq N$. Thus, there are exactly $2^n - 2$ such components in total, corresponding to all proper subsets of V. (Recall that $|N| = 1$.) Q.E.D.

The "dual" QTP given by $(n, 1)$ is also optimal by Lemma 5. In fact, it is not hard to see that the QTPs given by $(2, n-1)$ and $(n\text{-}1, 2)$ are also component optimal. However, for $n > 2$, no other QTPs are component optimal. This can be checked from the following general formula for $\mathrm{CM}(f)$ when f is the QTP given by (d, e): assuming that $d \geqq e$,

$$\mathrm{CM}(f) = \sum_{r=1}^{e-1} 2^r \binom{n}{r} + \sum_{r=e}^{d-1} \binom{n}{r}.$$

On the other hand, not all component optimal DTPs are QTPs. For instance, consider a specific site i, map the component $\{(i, w)\}$ to z and T with site $(T) = V - \{i\}$ and state $(T) = \{p\}$ to x while all other components are mapped as for the QTP given by $(n, 1)$. This new DTP is component optimal but not a QTP.

3.1. Site optimal protocols. First, we notice that the component optimal DTPs obtained above may not be site optimal. To see this, let us consider the QTPs first. For the QTP f given by (d, e), $\mathrm{SM}(f)$ can be shown to be

$$\sum_{r=1}^{e-1} r \cdot 2^r \cdot \binom{n}{r} + \sum_{r=e}^{d-1} r \cdot \binom{n}{r}$$

assuming that $d \geqq e$. Let f' be the QTP given by $(n-1, 2)$ and f'' by $(n-2, 3)$. Then, it can be verified from the above formula that $\mathrm{SM}(f'') < \mathrm{SM}(f')$ for $n > 8$. In general, let k_0 be the smallest positive integer k such that $k \geqq (n-k)(2^{n-k} - 1)$. Denote by u the QTP characterized by $(k_0, n - k_0 + 1)$. Now, we prove that u is site optimal among the DTPs of P.

THEOREM 6. *u is site optimal among all* QTPs *of P.*

Proof. For any m, $\lceil n/2 \rceil \leqq m < n$, let f_m be the QTP given by $(m, n - m + 1)$. Then,

$$\begin{aligned}\mathrm{SM}(f_m) - \mathrm{SM}(f_{m+1}) &= (n-m)2^{n-m}\binom{n}{m} - n\binom{n}{m} \\ &= [(n-m)2^{n-m} - n]\binom{n}{m}.\end{aligned}$$

For $m < k_0$, $m < (n-m)2^{n-m} - n + m$, so that $(n-m)2^{n-m} - n > 0$. Thus, $\mathrm{SM}(f_m) > \mathrm{SM}(f_{m+1})$. On the other hand, $\mathrm{SM}(f_m) \leqq \mathrm{SM}(f_{m+1})$ for $m \geqq k_0$. Hence, $\mathrm{SM}(f_m)$ is minimum for $m = k_0$. Q.E.D.

THEOREM 7. *u is site optimal among all* DTPs *of P.*

Proof. We want to show that every DTP f has a QTP of better performance under the site measure. Application of the above theorem then proves our claim.

Case 1. Consider DTP f such that there are no S, T satisfying site $(S) =$ site (T), state $(S) = N$, state $(T) =$ Pc, $f(S) = y$ and $f(T) = x$. Thus, for any $M \subseteq V$, at least one of U or W where site $(U) =$ site $(W) = M$, state $(U) =$ Pc, state $(W) = N$, is mapped

to z. Since there are subsets of V with size r, $1 \leqq r < n$, $\mathrm{SM}(f) \geqq \sum_{r=1}^{n-1} r \cdot \binom{n}{r}$. Now, consider the QTP s characterized by $(n-1, 2)$. $\mathrm{SM}(f) - \mathrm{SM}(s) = n^2 - 2n > 0$ for $n > 2$.

Case 2. Consider DTP f such that there exist S, T satisfying $M = \mathrm{site}(S) = \mathrm{site}(T)$, $\mathrm{state}(S) = N$, $\mathrm{state}(T) = \mathrm{Pc}$, $f(S) = y$ and $f(T) = x$. Assume that $f(S') \neq z$ for all S' such that $\mathrm{site}(S') = \mathrm{site}(S)$ since it would not increase $\mathrm{SM}(f)$. Let S be the smallest component satisfying the above conditions. If $S \leqq \lfloor n/2 \rfloor$, then it is not hard to check that $\mathrm{SM}(f) > \mathrm{SM}(g)$. (Recall that g is the QTP given by $(n, 1)$.) Assume that $S > \lfloor n/2 \rfloor$. Let

$$L = \{M \subseteq V \mid \text{there exist } S, S' \text{ in } D, \text{ such that } \mathrm{site}(S) = \mathrm{site}(S') = M, \mathrm{state}(S) = N, \mathrm{state}(S') = \mathrm{Pc}, f(S) = y \text{ and } f(S') = x\}.$$

If $|M'| > k_0$ for some $M' \subseteq V$ *not in* L, observe that $\mathrm{SM}(f)$ is not increased when M' is inserted into L (due to the definition of k_0 and Lemma 4). Thus, in general, all M' not in L such that $M' > k_0$ can be placed into L without increasing $\mathrm{SM}(f)$.

If $|M'| < k_0$ for some M' *in* L, then pick the smallest such M' and delete it from L. Again, by the definition of k_0, this cannot increase $\mathrm{SM}(f)$. Repeating this process, we can convert f into a QTP without increasing $\mathrm{SM}(f)$. Q.E.D.

The following results are immediate from the above theorem and Lemma 5.

COROLLARY 1. *There are at most two* QTP*s which are site optimal among the* DTP*s and these are the only site optimal* DTP*s.*

COROLLARY 2. *For* $n \geqq 9$, *no component optimal* DTP *is site optimal and vice versa.*

These results are slightly disturbing since they show that the site and component optimalities are complementary. Ideally, one would like to have a TP that simultaneously optimizes the number of sites and components. But, we have just proved that it is not possible when the commit protocol used is decentralized. Furthermore, QTPs are not very desirable due to the fact that, for any QTP f, there exist partitionings in which *all components* (and hence all sites) wait under f. Thus, they cannot guarantee that at least one component in any partitioning can be allowed to complete the incomplete transaction.

We now consider the centralized commit protocol where the situation is rather pleasant: we produce a TP which is both component and site optimal. Furthermore, this TP guarantees the transaction completion in at least one component, as long as the coordinator site is operational.

4. Centralized commit protocol. We first present a slightly generalized definition of centralized protocols. Define a partial ordering on the states of an FSA as: $t \leqq s$ if and only if the distance of s to its nearest final state is no less than the distance of t to its nearest final state. A set of FSAs LS is said to be a *leading set* if and only if for any S in D, (i, s), $(j, t) \in S$ and $i \in \mathrm{LS}$ implies $s \leqq t$.

DEFINITION 10. A version P of the canonical commit protocol is *centralized* if and only if it has a leading set of sites.

TPs of the centralized commit protocol are called centralized TPs (CTPs). Observe first that the class of DTPs considered in the previous section is a subclass of the CTPs when their domain is restricted to the realizable components for the centralized protocol. This is because the commit and preservation properties of a TP hold even if the TP is restricted over a proper subset of its domain. Secondly, we observe that there is a natural class of CTPs which is very interesting: the CTPs that explicitly exploit the existence of leading set. Notice that, when a leading site (a site whose FSA is in the leading set) is in state w ("wait"), all sites in the system are in states from $\{w, q\}$. We call the CTPs which use this fact *leading set* TPs (LTPs). Formally,

DEFINITION 11. A CTP is an LTP if and only if, for all S in D, site $(S) \cap \text{LS} \neq \emptyset$ implies $f(S) \neq z$ where LS is the leading set.

COROLLARY 3. *In absence of the simultaneous failure of all leading sites, for any transaction, there is at least one component in any partitioning that can successfully terminate that transaction when an* LTP *is used.*

In principle, it is possible to abort a transaction in a component as long as it is guaranteed that no other component commits it. But, due to practical considerations, this approach is not satisfactory. Typically, an aborted transaction is repeatedly tried until it is committed. Thus, a TP should try to commit a transaction if it is possible to do so. This is equivalent to saying that for any LTP f and S in D, $f(S) = x$ whenever state $(S) \cap (\text{Com} \cup \text{Pc}) \neq \emptyset$.

LEMMA 6. *Let f be an* LTP. *Then, $f(S) = z$ whenever* site $(S) \cap \text{LS} = \emptyset$ *and* state $(S) \subseteq N$.

Proof. Notice that, if there are S', S'' such that $S \cup S' \in D$, $S \cup S'' \in D$, (site $(S') \cup$ site $(S'')) \cap$ site $(S) = \emptyset$, $f(S') = x$, and $f(S'') = y$, then $f(S) = z$. It is easy to see that one can produce such S', S'' when f is an LTP and site $(S) \cap \text{LS} = \emptyset$. For instance, take site (S') = site (S'') = LS, state $(S'') = N$ and state (S') = Pc. Q.E.D.

Based on this lemma, we construct a simple but effective CTP. But, first we generalize the notion of quorum protocols introduced in the previous section.

DEFINITION 12. A quorum TP is *weighted* if, in the definition of the QTP, each site is assigned a weight and the size of a set is replaced by its weight, i.e. sum of the weights of its elements. The condition $d + e > n$ is replaced by $d + e >$ sum of the weights of all nonleading sites.

Now, consider a weighted QTP defined as follows: assign a weight of 1 to all sites not in the leading set and assign a weight of n to each leading site. Set $d = 1$ and $e = n$. This CTP can be explicitly given as follows: Call it h.

1. state $(S) \cap (\text{Com} \cup \text{Pc}) \neq \emptyset$ implies $h(S) = x$,
2. state $(S) \cap (\text{Com} \cup \text{Pc}) = \emptyset$ and state $(S) \cap (\text{Ab} \cup \{q\}) \neq \emptyset$ implies $h(S) = y$,
3. site $(S) \cap \text{LS} \neq \emptyset$ and state $(S) \subseteq N$ implies $h(S) = y$,
4. $h(S) = z$ otherwise.

It can be easily seen that h is indeed an LTP. Following result shows that h is the "best" among the LTPs in a very strong sense:

LEMMA 7. *Let f be an* LTP. *Then, $f(S) = z$ whenever $h(S) = z$.*

Proof. $h(S) = z$ only if site $(S) \cap \text{LS} = \emptyset$ and state $(S) \subseteq N$. $f(S) = z$ in this case, by Lemma 6. Q.E.D.

COROLLARY 4. *h is both component and site optimal among the* LTPs.

We now consider the decentralized TPs of the centralized commit protocol, which pay no attention to the existence of the leading set. Since these are only restrictions of the DTPs considered in § 3, we call them *restricted* DTPs (RDTPs).

LEMMA 8. *Let f be an* RDTP. *Then,* $\text{CM}(f) > 2^{n-|\text{LS}|} - 2$.

Proof. Let $D' = \{S \in D | \text{site}(S) \cap \text{LS} = \emptyset\}$. Then, as in the proof of Theorem 4, it can be shown that $|\{S \in D' | f(S) = z\}| \geq 2^{n-|\text{LS}|} - 2$. Since f is an RDTP, there exists $S \in D$ such that $f(S) = z$ and site $(S) \cap \text{LS} \neq \emptyset$. Q.E.D.

THEOREM 8. *Let f be an* RDTP. *Then,* $\text{CM}(f) > \text{CM}(h)$.

Proof.

$$\text{CM}(h) = \sum_{k=1}^{n-|\text{LS}|} \binom{n-|\text{LS}|}{k} = 2^{n-|\text{LS}|} - 2 < \text{CM}(f).$$

Q.E.D.

THEOREM 9. *Let f be an* RDTP. *Then,* $\text{SM}(f) > \text{SM}(h)$ *for sufficiently large n.*

Proof. The DTP u is site optimal among all DTPs by Theorem 7. It can be proved that u, when restricted to the centralized commit protocol, remains site optimal among RDTPs. Thus, it is sufficient to show that $\mathrm{SM}(u) > \mathrm{SM}(h)$ in the restricted domain. The following inequality holds for the restricted u:

$$\mathrm{SM}(u) \geqq \sum_{k=1}^{n-k_0} k \cdot 2^{k-|\mathrm{LS}|} \binom{n}{k} + \sum_{k=n-k_0+1}^{k_0-1} k \cdot \binom{n}{k}$$

$$= \sum_{k=1}^{n-1} k \cdot \binom{n}{k} + \sum_{k=1}^{n-k_0} k \cdot (2^{k-|\mathrm{LS}|} - 1) \binom{n}{k} - \sum_{k=1}^{n-k_0} (k + k_0 - 1) \binom{n}{k + k_0 - 1}.$$

On the other hand,

$$\mathrm{SM}(h) = \sum_{k=1}^{n-|\mathrm{LS}|} k \cdot \binom{n-|\mathrm{LS}|}{k} \leqq \sum_{k=1}^{n-1} k \binom{n-1}{k}.$$

Thus, $\mathrm{SM}(h) < \mathrm{SM}(f)$ for sufficiently large n. Q.E.D.

5. Conclusion. Network partitioning is harder to deal with than many other failure problems in a distributed environment. Though its occurrence is not very frequent, it cannot simply be ignored in certain applications. But, none of the existing systems have tried to consider the network partitioning problem. Being termed as a "catastrophic situation," it is manually handled [11].

In this paper, we have formalized various aspects of network partitioning problems and designed effective protocols to be used in the presence of this failure. Certain practical criteria are used for measuring the performance of termination protocols. Both decentralized and centralized versions of commit protocols are studied and optimal termination protocols are presented. For the centralized case, a single TP is shown to be optimal under both component and site measures. In the absence of coordinator failures, this optimal protocol guarantees continued transaction processing in at least one component in any partitioning. Quorum protocols are shown to be optimal in the decentralized case. This is not greatly satisfying since one cannot guarantee continued processing in any component in this case. Thus, it is desirable that a centralized TP be used when the coordinator sites are reasonably failure-free. This in turn would imply that a centralized commit protocol be employed during the normal (failure-free) operations of the system. Hence, our results can also be interpreted as lending support to the usage of centralized protocols—not only because they are less expensive (in terms of messages) than the decentralized ones, but also due to their effectiveness in handling network partitionings.

From our bounds on the component and site measures, it is not hard to see that the fraction of waiting components/sites in the event of an arbitrary network partitioning is very small when the optimal TPs are used.

We have also proved that any commit protocol should have a "committable" state to be of use in presence of partitioning. This tends to increase the cost of the normal operation. But, one can fine-tune the protocol by running FSAs with committable states only at certain critical sites.

The measures we have used involve no statistical information. They are somewhat simplified approximations of the actual system availability. Generalizations of these measures, which utilize the statistical information on partitionings and the states of the commit protocol are considered elsewhere [5].

REFERENCES

[1] P. ALSBERG AND J. DAY, *A principle for resilient sharing of distributed resources*, Proc. 2nd International Conference on Software Engineering, October 1976, pp. 562-570.

[2] P. A. BERNSTEIN AND N. GOODMAN, *Concurrency control in distributed database systems*, Comput. Surveys, 12 (1981), pp. 185-221.

[3] P. A. BERNSTEIN AND D. W. SHIPMAN, *A formal model of concurrency control mechanisms for database systems*, Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, August 1978.

[4] F. CHIN AND K. V. S. RAMARAO, *An information-based model for failure handling in Distributed databases*, IEEE Trans. Software Engrg., to appear.

[5] ———, *Maximization of database availability in presence of network partitioning*, submitted to J. Assoc. Comput. Mach.

[6] E. C. COOPER, *Analysis of distributed commit protocols*, Proc. ACM SIGMOD, 1982, pp. 175-183.

[7] S. DAVIDSON AND H. GARCIA-MOLINA, *Protocols for partitioned distributed database systems*, Proc. IEEE Symposium on Reliability in Distributed Software and Database Systems, June 1981.

[8] H. GARCIA-MOLINA, *Reliability issues for completely replicated distributed databases*, Proc. IEEE COMPCON, Fall 1980, pp. 442-449.

[9] ———, *Elections in a distributed computing system*, IEEE Trans. Comput., C-31 (1983), pp. 393-481.

[10] J. N. GRAY, *Notes on database operating systems, operating systems: an advanced course*, Lecture Notes in Computer Science 60, Springer-Verlag, New York, pp. 393-481.

[11] M. HAMMER AND D. SHIPMAN, *Reliability mechanisms for* SDD-1, ACM Trans. Database Systems, 5 (1980), pp. 431-466.

[12] B. LAMPSON, *Atomic transactions, distributed systems architecture and implementation: an advanced course*, Lecture Notes in Computer Science 100, Springer-Verlag, New York, Chapter 11.

[13] K. V. S. RAMARAO, *On the completion of distributed transactions while recovering from a network partitioning*, Proc. 1984 Princeton Conf. on Information Sciences and Systems.

[14] H. R. STRONG AND D. DOLEV, *Byzantine agreement*, Proc. IEEE COMPCON, Spring 1983, pp. 77-82.

[15] D. SKEEN, *Nonblocking commit protocols*, Proc. ACM SIGMOD, 1981, pp. 133-147.

[16] ———, *A quorum-based commit protocol*, Computer Science TR 82-483, Cornell University, Ithaca, NY, 1983.

[17] D. SKEEN AND M. STONEBRAKER, *A formal model of crash recovery in a distributed system*, IEEE Trans. Software Engrg. TSE-83.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
011

COMPUTATIONAL COMPLEXITY: ON THE GEOMETRY OF POLYNOMIALS AND A THEORY OF COST: II*

M. SHUB† AND S. SMALE‡

Abstract. This paper deals with traditional algorithms, Newton's method and a higher order generalization due to Euler. These iterations schemes and their modifications have had a great success in solving nonlinear systems of equations. We give some understanding of this phenomenon by giving estimates of efficiency. The problem we focus on is that of finding a zero of a complex polynomial.

Key words. Newton, Euler, approximate zero, polynomial, average

1. This paper deals with traditional algorithms, Newton's method and a higher order generalization due to Euler. These iteration schemes and their modifications have had a great success in solving nonlinear systems of equations. We give some understanding of this phenomenon by giving estimates of efficiency. The problem we focus on is that of finding a zero of a complex polynomial.

Following the work of Newton and Euler we define a rational map (or iteration scheme) $E: \mathbb{C} \to \mathbb{C}$ ($\mathbb{C}$ the complex numbers) which depends on three parameters:

(a) f, a polynomial, $f(z) = \sum_{i=0}^{d} a_i z^i$, $a_d \neq 0$. Often times we take f to be in the space $P_d(1)$ where

$$P_d(1) = \left\{ f \,\middle|\, f(z) = \sum_{i=0}^{d} a_i z^i,\ a_d = 1,\ |a_j| \leqq 1 \right\};$$

(b) a positive integer k (which amounts to the number of derivatives used); and

(c) A number h, $0 < h \leqq 1$.

Then define $E = E_{k,h,f}$, $E: \mathbb{C} \to \mathbb{C}$ by

$$E(z) = T_k(f_z^{-1}((1-h)f(z))).$$

Here f_z^{-1} is the branch of the inverse of f which takes $f(z)$ into z, given as an analytic function in a neighborhood of $f(z)$ (provided $f'(z) \neq 0$).

T_k is the truncation of the power series expansion in h about $h = 0$ at degree k. It is easy to check that $E_{1,1,f}$ is Newton's method. One can see a full discussion in Shub-Smale (1982) (hereafter referred to as [S-SI]).

Consider first the problem: Given (f, ε), $f \in P_d(1)$, $\varepsilon > 0$, produce a $z \in \mathbb{C}$ with $|f(z)| < \varepsilon$. For this we particularize the Newton-Euler iteration scheme by choosing k and h to depend only on f and ε, in a certain way. Let

$$k = \lceil \max (\log |\log \varepsilon|, \log d) \rceil$$

where $\lceil x \rceil$ is the least integer greater than or equal to x. We will define in § 2, universal constants H and K, approximately $\frac{1}{512}$ and 512 respectively. Then we will take

$$h = H.$$

* Received by the editors November 17, 1983, and in revised form September 15, 1984. This work was supported in part by grants from the National Science Foundation.

† Mathematics Department, Queens College and the Graduate School, City University of New York, New York, New York 11367.

‡ Mathematics Department, University of California, Berkeley, California 94720.

Thus with these specializations the Newton-Euler iteration scheme $E:\mathbb{C}\to\mathbb{C}$ depends only on (ε, f) and we write $E_\varepsilon = E$. With $\varepsilon > 0$ define

ALGORITHM (N-E)$_\varepsilon$. Let $f\in P_d(1)$ and $n = K(d+|\log \varepsilon|)$.

(1) Choose $z_0\in\mathbb{C}$, $|z_0|=3$ at random and set for $i=1,2,3,\cdots$ (an *iteration*) $z_i = E_\varepsilon(z_{i-1})$ terminating if ever $|f(z_i)|<\varepsilon$

(2) If $i=n$, go to (1) (a *cycle*).

THEOREM A. *For each f, ε, (N-E)$_\varepsilon$ terminates with probability one and produces a z satisfying $|f(z)|<\varepsilon$. The average number of cycles is less than or equal to 6. Hence the average number of iterations is less than $6K(d+|\log\varepsilon|)$.*

Here average and probability refer to the choice of the sequence of z_0 in (1) of (N-E)$_\varepsilon$.

Remark. With certainty it only takes about twice as long. See § 2 for an elucidation of this remark. In practice one can obviously do better by trying and testing $h=1, \frac{1}{2},\cdots, H$. We have not analyzed this. Also see § 2 for the total number of arithmetic operations required.

Next consider sharpening the goal $|f(z)|<\varepsilon$. Machine or discrete processes will not generally succeed in finding exact zeros of polynomials. For our theory we use the notion of approximate zero z of a polynomial f, Smale (1981), [S-SI]. This complex number z is one close to an actual zero, where closeness is defined without any arbitrary choices. The justification of close is given in both theoretical and practical terms. More precisely define

$$\rho_f = \min_{\substack{\theta \\ f'(\theta)=0}} |f(\theta)|.$$

There is a universal constant c (about $\frac{1}{12}$) and z is an *approximate zero* of f if $|f(z)|<c\rho_f$. Then this proposition follows.

PROPOSITION. *Let $E = E_{k,f,1}$ be the Newton-Euler scheme with arbitrary k and $h=1$, and E^l the composition $E\circ\cdots\circ E$ l times. Let z be an approximate zero of f, so $|f(z)|<bc\rho_f$ with $b<1$. Then*

$$|f(E^l(z))|<b^{(k+1)^l}c\rho_f.$$

The extremely rapid convergence gives some good justification for "approximate zero".

In Algorithm (N-E)$_\varepsilon$ there was a random element, the choice of z_0. Now probability enters into our analysis in a second way. We average over $f\in P_d(1)$ with respect to a uniform distribution; that is we normalize Lebesgue measure on $P_d(1)\subset\mathbb{C}^d=\mathbb{R}^{2d}$. We use these probabilities since speedy algorithms are not usually infallible.

Define for each $f\in P_d(1)$

$$\varepsilon_f = \frac{1}{(2d)^{4d}}|D_f|$$

where D_f is the discriminant of f (see Lang (1965)). With K as above let

$$n = \lceil K(d+|\log\varepsilon_f|)\rceil.$$

Let E be the Euler-Newton iteration scheme with $h=H$, and $k=\lceil\max(\log|\log\varepsilon_f|, \log d)\rceil$, so that E depends only on f.

ALGORITHM (N-E). Let $f \in P_d(1)$, satisfy $\varepsilon_f > 0$.

(1) Set $m = 1$;

($2m$) Choose $z_0 \in \mathbb{C}$, $|z_0| = 3$ at random and set $z_n = E^n(z_0)$. If $|f(z_n)| < \varepsilon_f$ terminate and print: "z_n is an approximate zero;"

(3) Otherwise let $m = m+1$ and go to (2_m).

THEOREM B. *Algorithm* (N-E) *terminates* (*and hence produces an approximate zero*) *with probability* 1 *and the average number of iterations is less than* $K_1 d \log d$ *where* K_1 *is a universal constant.*

We make the probability considerations a bit more precise.

Let S^1_R be the circle in $\mathbb{C}$ defined by $|z| = R$ and endow it with the uniform probability measure (Lebesgue measure normalized to 1). Set $R = 3$ and denote by Ω the product of S^1_R with itself a countable number of times. Thus a point z_0 of Ω is a sequence $\bar{z} = (\bar{z}_1, \bar{z}_2, \cdots)$ with $|\bar{z}_i| = 3$. Endow Ω with the product measure as well as $P_d(1) \times \Omega$. Let $T: P_d(1) \times \Omega \to Z^+$ be defined by: $T(f, \bar{z})$ is the first m such that $E^n(\bar{z}_m) < \varepsilon_f$.

Thus the total number of iterations of Algorithm (N-E) for a given f is of the form $S(f, \bar{z}) = nT(f, \bar{z})$, $n = K(d + |\log \varepsilon_f|)$. Theorem B asserts than when $\varepsilon_f > 0$, $S(f, \bar{z})$ is defined for almost all $\bar{z} \in \Omega$. Moreover $S(f) = \int_{\bar{z} \in \Omega} S(f, \bar{z})$ is defined and finite for almost all f and

$$\int_{f \in P_d(1)} S(f) \leqq K_1 d \log d.$$

By Fubini's theorem, we could equally well assert that

$$\int_{(f, \bar{z}) \in P_d(1) \times \Omega} S(f, \bar{z}) \leqq K_1 d \log d.$$

Remark 1. We are assuming exact arithmetic in the theory here. In general, because of the robust properties of Algorithms (N-E)$_\varepsilon$ and (N-E), this is reasonable. However the calculation of ε_f in Algorithm (N-E) is not so robust. In that respect, Theorem A is more satisfying than Theorem B.

Remark 2. Our work emphasizes the theoretical side, and the understanding of classic algorithms, rather than the design of new practical algorithms. Yet the results do have some implications for the latter. For example they suggest calculating derivatives up to order $\lceil \log d \rceil$ and/or $\lceil \log |\log \varepsilon| \rceil$ could give speedier routines, especially for one complex polynomial. We have not tested our algorithms on the machine.

Remark 3. The number of arithmetic operations in contrast to the number of iterations is approximately quadratic in d. This is proved in § 2.

Remark 4. Questions of variance arising in these theorems can be handled. See § 2.

At this point we review some of the motivation from Smale (1981), Hirsch and Smale (1979) and [S-SI]. Let f be a polynomial, $f: \mathbb{C} \to \mathbb{C}$, let $z \in \mathbb{C}$ and $w = f(z)$. The ray from w to 0 is the segment in the target space from w to 0. Let R_w denote this ray and f_z the branch of the inverse of f taking $f(z)$ back to z. If f_z^{-1} is defined on all of R_w then $\zeta = f_z^{-1}(0)$ is one of the zeros of f (Fig. 1). Since a polynomial maps a

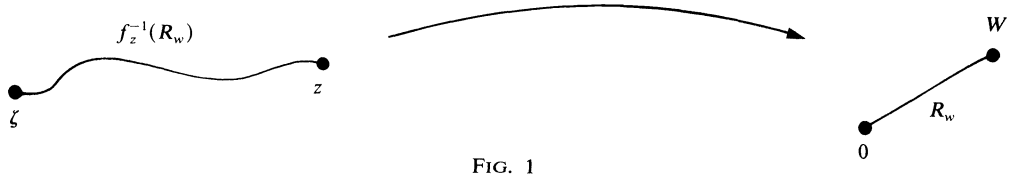


FIG. 1

neighborhood of infinity to a neighborhood of infinity and has only finitely many critical points it is a fairly simple calculus exercise to see that except for a finite number of rays (at most $d-1$) f_z^{-1} is defined on all of R_w. Thus we attempt to follow the curves $f_z^{-1}(R_w)$ from an initial starting point z to a zero ζ of t. One way to do this is to parameterize the R_w as $(1-h)f(z)$ for $0 \leqq h \leqq 1$. Then try to follow the ray by analytic continuation in h, $f_z^{-1}((1-h)f(z))$. Finally, truncate the power series at degree k in h to make the computation finite, $T_k(f_z^{-1}((1-h)f(z))$. This is $E_{k,h,f}(z)$. If $P(h)$ is positive for small positive h then $(1-P(h))$ is also further down the ray and we may try $T_k f_z^{-1}(1-P(h))f(z))$. The inverse images of the rays are also solution curves of the differential equations $\dot{z} = -f(z)/f'(z)$ and $\dot{z} = -\frac{1}{2}\operatorname{grad}|f(z)|^2$ (see Smale (1981)). Thus we may attempt to solve these equations numerically with step size h to attempt to follow the ray. These examples are given in greater detail in [S-SI].

We analyze a class of fast algorithms broader than the Euler iterations, but which still agree with the inverse of the ray to high order. First we extend the $E_{k,h,f}(z) = T_k(f_z^{-1}((1-h)f(z)))$ by replacing h on the right-hand side by $P(h) = \sum_{i=1}^{k} c_i h^i$ where c_i is real for all i and $c_1 > 0$. These generalized Euler iterations are

$$\mathrm{GE}_{p,k,h,f}(z) = T_k(f_z^{-1}((1-P(h))f(z)).$$

Thus $E_{k,h,f}$ is given by $P(h) = h$. The $\mathrm{GE}_{P,k,h,f}$ are polynomials in h of degree k. We allow modifications of these polynomial iterations by addition of a well bounded remainder term of order $k+1$ in h. We denote this largest class of iterations we consider by GEM_k (GEM = Generalized Euler with Modification). These iterations are fast.

An important ingredient in the analysis is the function $P_d \times \mathbb{C} \to R^+$. $(f, z) \to \Theta_{f,z}$ which was introduced in [S-SI]. We recall the definition of this function. Given f, z and $0 \leqq \alpha \leqq \pi/2$, let

$$W_{f,z,\alpha} = \left\{ w \in \mathbb{C} \mid 0 < |w| < 2|f(z)|, \left| \arg \frac{w}{f(z)} \right| < \alpha \right\}.$$

W is an open wedge of angle α centered at $f(z)$. Let $\Theta_{f,z}$ be the max $\alpha \leqq \pi/2$ on which f_z^{-1} is defined by analytic continuation (Fig. 2).

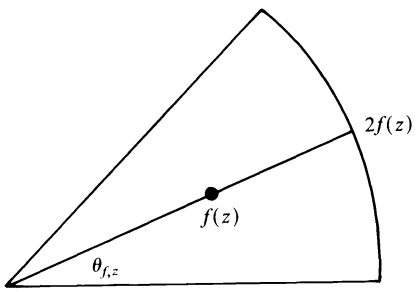


FIG. 2. *f_z^{-1} is defined on this wedge.*

In § 3 we prove Theorem C.

THEOREM C. *Suppose that $z' = I_{h,f}(z)$ is a GEM_k iteration. Then there is a constant k depending only on I such that: If $\Theta_{f,z_0} > 0$ and $|f(z_0)| > L > 0$ then there is an h given explicitly such that*

$$|f(z_n)| < L \quad \text{for } n = k\left[\frac{\log |f(z_0)/L|}{\Theta_{f,z_0}}\right]^{(k+1)/k}$$

and $z_n = (I_{h,f})^n(z_0)$.

Theorem C can be used to show that any GEM_k iteration can be adapted to produce fast algorithms as in Theorems A and B of this paper. Finally, in § 3 we show that the GEM_k iterations are precisely the efficiency k incremental algorithms defined in [S-SI] which satisfy an additional "smallness" condition.

2. The main goal of this section is to prove Theorems A and B of § 1. We require some of the main results of [S-SI]. First Proposition 1 is a special case of [S-SI, Thm. 2] at least after a short translation of constants. The constants K, K' are universal, not very large and well estimated via [S-SI]. Let $k = 1, 2, \cdots$, and $\varepsilon > 0$.

PROPOSITION 1. *There exist* K, $K' > 0$ *so that if* $0 < h \leqq K/(d + |\log \varepsilon|)^{1/k}$, $n = (K'/h)(d + |\log \varepsilon|)$, $f \in P_d(1)$ *and* $|z_0| = 3$ *with* $\Theta_{f,z_0} \geqq \pi/12$ *then*

$$|f(E^i(z_0))| < \varepsilon \quad \textit{for some } 0 \leqq i < n.$$

Here E^i *is* $E \circ \cdots \circ E$, i *times and* E *is the Euler*$_k$ *iteration of* § 1.

Next by specializing k to $k = \max(\lceil \log d \rceil, \lceil \log |\log \varepsilon| \rceil)$ we obtain

COROLLARY. *There exist universal constants* H, K *so that for* $n = K(d + |\log \varepsilon|)$, $E = E_{k,H}$, $f \in P_d(1)$ *and* $|z_0| = 3$ *with* $\Theta_{f,z_0} \geqq \pi/12$

$$|f(E^i(z_0))| < \varepsilon \quad \textit{for some } 0 \leqq i < n.$$

Finally we require the following proposition which is obtained from [S-SI, Proposition 3, § 4]. For $f \in P_d(1)$, let $V_f = \{z \,|\, |z| = 3 \text{ and } \Theta_{f,z} > \pi/12\}$. Then using the uniform probability measure on $S = \{z \,|\, |z| = 3\}$ we have Proposition 2.

PROPOSITION 2. *The measure of* $V_f \geqq \frac{1}{6}$ *for any* f.

We recall a bit of probability theory. The set S has a probability measure. Impose the product measure on Ω the (ordered) countable product of S with itself. Suppose $V \subset S$ has measure v. For $\bar{z} \in \Omega$, $\bar{z} = (\bar{z}_1, \bar{z}_2, \cdots)$. Let $m(\bar{z})$ be the m such that $\bar{z}_i \notin V$ for $i < m$ but $\bar{z}_m \in V$.

PROPOSITION 3.

$$\int_{\bar{z} \in \Omega} m(\bar{z}) = \frac{1}{v}.$$

Proof. Let $V_i = \{\bar{z} \,|\, m(\bar{z}) = i\}$, $i = 1, 2, \cdots$. Let v_i be the measure of V_i. Then $v_i = v(1-v)^{i-1}$ and moreover

$$\int_{\bar{z} \in \Omega} m(\bar{z}) = \sum_{i=1}^{\infty} i v_i = \sum_{i=1}^{\infty} i v (1-v)^{i-1} = \frac{1}{v}.$$

Now we can prove Theorem A of § 1. In fact it is an immediate consequence of the corollary of Proposition 1, and Propositions 2 and 3. Q.E.D.

We will need a few more facts to prove Theorem B. We begin with another elementary result of probability theory.

DEFINITION. Let (X, μ) be a probability space with no atoms. Let $S: X \to R_+$ be a real valued nonnegative measurable function and let $f: (0, 1) \to R$ be decreasing and Riemann integrable. We say that $S(x) \leqq f(\mu)$ with probability $1 - \mu$ if $\mu\{x \,|\, s(x) \leqq f(y)\} \geqq 1 - y$ for all $0 < y < 1$.

PROPOSITION 4. *Suppose as above that* $S(x) < f(\mu)$ *with probability* $1 - \mu$. *Then*

1) $$E(S) = \int_x S(x) \mu(dx) \leqq \int_0^1 f(\mu)\, d\mu.$$

2) $$\mathrm{Var}(S) = \int_x (S(x) - E(S))^2 \mu\,(dx) \leqq \int_0^1 f^2(\mu)\, d\mu - (E(S))^2.$$

Proof. We only prove 1). Let $y_i \in (0,1)$ for $-\infty < i < \infty$ be a decreasing sequence with 0 and 1 as limit points. Let $\cdots \supset M_{y_i} \supset M_{y_{i-1}} \supset \cdots$ be constructed so that $\mu(M_{y_i}) = 1 - y_i$ and $S(x) \leqq f(y_i)$ for $x \in M_{y_i}$. Then $\int S(x)\mu(dx) \leqq \sum f(y_i)\mu(M_{y_i} - M_{y_{i-1}}) = \sum f(y_i)(y_{i-1} - y_i)$ which converges to the Riemann integral of f.

Recall that $\rho_f = \min_{\Theta, f'(\Theta)=0} |f(\Theta)|$. From Smale (1981) we have Proposition 5.

PROPOSITION 5.

$$\text{Vol}\{f \in P_d(1) \,|\, \rho_f < \alpha\} < d\alpha^2.$$

Here Vol means normalized volume so that $\text{Vol}(P_d(1)) = 1$ and Vol is a probability measure on $P_d(1)$.

PROPOSITION 6.

$$\int_{\substack{f \in P_d(1) \\ \rho_f < 1}} |\log \rho_f| \leqq \tfrac{1}{2} \log d + 1.$$

Proof. Let $\mu = d\alpha^2$ so from Proposition 5, $\text{Vol}\{f \in P_d(1) \,|\, \rho_f < \sqrt{\mu/d}\} < \mu$ and

$$\text{Vol}\{f \in P_d(1), \rho_f < 1 \,||\log \rho_f| < \tfrac{1}{2}\log d + \tfrac{1}{2}|\log \mu|\} > 1 - \mu.$$

Now apply Proposition 4 to finish the proof.

PROPOSITION 7. *There is a constant K_2 and for $f \in P_d(1)$,*

a) $d^d \rho_f^{d-1} \leqq D_f$,

b) $\varepsilon_f \leqq c\rho_f$,

c) $\int_{P_d(1)} |\log \varepsilon_f| \leqq K_2 d \log d$.

Proof. $D_f = d^d \prod_{\theta_i \text{s.t.}, f'(\theta_i)=0} f(\theta_i)$ (see Lang (1965)) so

$$|D_f| \geqq d^d \Big(\min_{\substack{\theta_i \text{s.t.} \\ f'(\theta_i)=0}} |f(\theta_i)| \Big)^{d-1} = d^d \rho_f^{d-1}.$$

which proves a). We will use a lemma to prove b). First recall that the discriminant is given as the determinant of a $(2d-1)$ by $(2d-1)$ matrix, see Lang (1965). $D_f = R(f, f') = d^d R(f, f'/d)$.

LEMMA 1. 1) *For a polynomial f of degree d with $|a_i| \leqq 1$ for $i \geqq 1$ then*

$$\left| \frac{\partial R(f, f'/d)}{\partial a_0} \right| \leqq (2d-1)! \sum_{j=1}^{d-1} j \binom{d-1}{j} |a_0|^{j-1}.$$

2) *If $f \in P_d(1)$ then $(2d-1)!(d-1)(\rho_f + 2)^{d-2} \rho_f \geqq R(f, f'/d)$.*

Proof. The resultant is a polynomial of degree $d-1$ in a_0. Crude estimates give that the modulus of the coefficient of a_0^j by expanding the determinant is $\leqq \binom{d-1}{j}(2d-1-j)!$ so

$$\left| \frac{\partial R(f, f'/d)}{\partial a_0} \right| \leqq \sum_{j=1}^{d-1} j \binom{d-1}{j} (2d-1-j)! |a_0|^{j-1}$$

$$\leqq (2d-1)! \sum_{j=1}^{d-1} j \binom{d-1}{j} |a_0|^{j-1}.$$

Let $f'(\theta) = 0$ and $|f(\theta)| = \rho_f$.

Let $f_0 = f - f(\theta)$. Then $a_0(f_0) = a_0 - f(\theta)$, $a_i(f_0) = a_i(f)$ for $i > 1$ and $R(f_0, f_0'/d) = 0$.

Applying the mean value theorem to the line segment between f_0 and f gives

$$\begin{aligned}
|R(f,f'/d)| &= |R(f,f'/d) - R(f_0,f_0'/d)| \\
&\leqq (2d-1)! \sum_{j=1}^{d-1} j\binom{d-1}{j}(|a_0|+\rho_f)^{j-1}|f-f_0| \\
&\leqq (2d-1)! \sum_{j=1}^{d-1} j\binom{d-1}{j}(\rho_f+1)^{j-1}\rho_f \\
&= (2d-1)!\rho_f\left(\left(\sum_{j=0}^{d-1}\binom{d-1}{j}x^j\right)'\Big|_{\rho_f+1}\right) \\
&= (2d-1)!\rho_f((1+x)^{d-1})'|_{\rho_f+1} \\
&= (2d-1)!(d-1)(\rho_f+2)^{d-2}\rho_f.
\end{aligned}$$

Now to prove b), note that

$$\rho_f < d+1 \quad \text{and} \quad d^d(2d-1)!(d-1)(d+3)^{d-2} < c(2d)^{4d}.$$

For the proof of c), note that $\varepsilon_f < 1$.

By definition

$$|\log \varepsilon_f| = \left|\log \frac{D_f}{(2d)^{4d}}\right|.$$

Therefore by using Proposition 7a,

$$|\log \varepsilon_f| \leqq K_3 d|\log d + |\log \rho_f|| \quad \text{and}$$

$$\int |\log \varepsilon_f| \leqq K_3 d\left(\log d + \int_{\rho_f<1} |\log \rho_f|\right) + K_3 d\left(\log d + \int_{\rho_f>1} |\log \rho_f|\right).$$

Now the first term on the left is estimated by Proposition 6. The second is estimated using the fact that $\rho_f < d+1$. This yields Proposition 7.

With these preparations we now prove Theorem B from Theorem A. Note first that by Proposition 7b, Algorithm (N-E) does terminate with an approximate zero if it terminates. Now apply Theorem A with Algorithm $(\text{N-E})_\varepsilon$ where ε is chosen to be ε_f, and then integrate over $P_d(1)$. This yields

$$\int_{P_d(1)\times\Omega} S(f,\bar{z}) \leqq 6Kd + 6K\int_f |\log \varepsilon_f|.$$

Now apply Proposition 7c. Q.E.D.

Remark 1. There are various ways to compute the Euler iterations

$$z' = E_{k,h}(z_0) = T_k(f_z^{-1}((1-h)f(z))).$$

a) The Taylor series expansions of f at z, f_z may be calculated by the algorithm of Shaw and Traub with $(2d-1)$ multiplications, $d-1$ divisions and $d^2+d/2$ additions, after writing $f(w) = (w-z_0)Q(w)+R$. See Knuth (1981, p. 470) or Borodin-Munro (1975, p. 33). Moreover f_z may be calculated in $O(d \log d)$ arithmetic operations and perhaps in $O(d)$ operations, see Borodin-Munro (1975, p. 106).

b) $T_k(f_z^{-1})$ may then be computed in $k^3/6$ multiplications by Lagrangian power series reversion (Knuth (1981, p. 508)). The algorithm of Brent-Kung (1976) (see also Knuth (1981, p. 510)) computes $T_k f_z^{-1}$ in $O(k \log k)^{3/2}$ operations and the value $T_k f_z^{-1}((1-h)f(z))$ in $O(k \log k)$ operations once the Taylor series of f at z is known.

c) Putting together a) and b) to get a good asymptotic estimate gives that $T_k f_z^{-1}(1-h)f(z))$ is computable in $O(d+k\log k)$ operations.

d) Thus in Theorem A the average of the total number of operations is

$$O((d+|\log \varepsilon|)(d+k\log k)),$$

where $k=\max\lceil \log d, \log|\log\varepsilon|\rceil$, or simply

$$O(d^2+d|\log\varepsilon|).$$

e) For Theorem B, the situation is a bit more subtle since k depends on f. The average total number of operations is

$$O\left(\int_{P_d(1)\times\Omega} S(f,z_0)(d+k\log k)\right)$$

where $k=\max\lceil \log d, \log|\log\varepsilon_f|\rceil$, which gives eventually

$$O(d^2(\log d)^2\log\log d).$$

f) D_f may be computed in $O(d(\log d)^2)$ arithmetic operations according to J. T. Schwartz (see Knuth (1981, p. 619)).

Remark 2. An estimate for the variance of the numer of iterates in Theorem B is simple to calculate by Proposition 4. The variance is $O(\int_f (d\log d+|\log\rho_f|)^2)$ which is $O(d^2(\log d)^2)$.

Remark 3. It is possible to devise deterministic algorithms in place of Algorithms $(\text{N-E})_\varepsilon$ and (N-E). By [S-SI, Prop. 3, § 4] $\Theta_{f,z_0}\geqq \pi/12$ except for at most $2(d-1)$ arcs of S^1_3 of angle $10\pi/12d$ each. Thus if we place $24d$ points evenly around S^1_3 at least one-twelfth of them will have Θ bigger than $\pi/12$. Now for the sake of Theorem A or B we can pick from this finite set at random eventually exhausting the set and still not take more than 12 choices on the average. Thus for Theorems A and B a deterministic version is possible with a factor of 2 in the number of steps. Now the algorithms always terminate. It is not necessary to say with probability one (in the case of Theorem B as long as $\varepsilon_f>0$).

3. We begin this section with a description of the GEM (Generalized Euler with Modification) iterations that we consider. Throughout this section $P(h)=\sum_{i=1}^k c_i h^i$ is a polynomial with c_i real and $c_1>0$. Thus for small real h, $(1-P(h))f(z)$ is on the line segment between $f(z)$ and 0. The Generalized Euler iterations are $\text{GE}_{p,k,h,f}(z)=T_k(f_z^{-1}((1-P(h))f(z)))$.

DEFINITION 1. $z'=I_{n,f}(z)$ is a GEM_k iteration iff there is a polynomial $P(h)$ and constants $c>0$, $\delta>0$ such that:

$$I_{h,f}(z)=\text{GE}_{p,k,h,f}(z)+FR_{k+1}(h,f,z)$$

where

$$F=-\frac{f(z)}{f'(z)}$$

and $|R_{k+1}(h,f,z)|\leqq Ch^{k+1}\max(1,1/h_1^k)$ for $0<h<\delta\min(1,h_1)$.

The number $h_1=h_1(f,z)$ is the radius of convergence of $f_z^{-1}((1-h)f(z))$ as a power series in h around zero.

Examples of GE iterations without modification are described in [S-SI] these include incremental Newton's method, kth order incremental Euler, and kth order Taylor's method for the solution of the differential equation $dz/dt=-f(z)/f'(z)$.

It is sometimes more convenient to express the GE iterations and the modifications in terms of the polynomials $\sigma_{f,z}$ introduced in [S-SI].

$$\sigma(w) = \sigma_{f,z}(w) = \sum \sigma_i w^i$$

where

$$\sigma_0 = 0, \quad \sigma_1 = 1, \quad \sigma_i = \frac{(-1)^{i+1} f^{(i)}(z) f^{i-1}(z)}{i!(f'(z))^i}.$$

We recall some basic facts about σ. Given the iteration $z' = I_{h,f}(z)$ write

$$I_{h,f}(z) = z + FR_{(h,f)}(z).$$

We frequently write $R(h, f, z)$ or just $R(h)$, $R(z)$ or R. By Taylor's formula (see [S-SI, § 1])

$$f(z') = f(z)(1 - \sigma \circ R)$$

or

$$\frac{f(z')}{f(z)} = 1 - \sigma \circ R.$$

Thus $R_{(h,f)}(z) \in \sigma^{-1}(1 - f(z')/f(z))$. σ is a polynomial of the same degree d as f, thus there are d points z' which give the same value for $f(z')/f(z)$. If $I_{h,f}(z)$ is continuous then at least for small values of h

$$I_{h,f}(z) = z + F\left(\sigma^{-1}\left(1 - \frac{f(z')}{f(z)}\right)\right)$$

where σ^{-1} is the branch of the inverse of σ taking 0 to 0. The radius of convergence of σ^{-1} is h_1. Comparing coefficients of powers of h (see [S-SI]) shows that

$$\mathrm{GE}_{p,k,h,f}(z) = z + FT_k(\sigma^{-1}(P(h))).$$

Consequently we may restate Definition 1.

DEFINITION 1′. $z' = I_{h,f}(z)$ is a GEM_k iteration iff there is a polynomial $P(h)$ and constants $c > 0$, $\delta > 0$ such that

$$I_{h,f}(z) = z + F(T_k(\sigma^{-1}(P(h)) + R_{k+1}(h, f, z))$$

where $|R_{k+1}(h, f, z)| \leqq ch^{k+1} \max(1, 1/h_1^k)$ for $0 < h < \delta \min(1, h_1)$.

One may also express the Generalized Euler iterations in the source as follows:

PROPOSITION 1. [S-SI]. *There are universal polynomials* $P_j = P_j(\sigma_1, \cdots, \sigma_{j+1}, c_1, \cdots, c_{j+1})$ *such that given any Generalized Euler iteration* (*without modification*):

$$\mathrm{GE}_{p,k,h,f}(z) = z + F\sum_{j=0}^{k-1} P_j h^{j+1},$$

$$P_0 = c_1,$$

$$P_1 = c_2 - \sigma_1 c_1^2,$$

$$P_2 = c_3 - 2\sigma_2 c_1 c_2 - (\sigma_3 - 2\sigma_2^2)c_1^3.$$

One can write down P_j *explicitly inductively, in terms of* P_i, $i < j$, σ_{j+1} *and* c_{j+1}.

Proof. The coefficients of σ^{-1} are computed from the σ_i and $P_j(c, \sigma)$ is defined by

$$\sum_{j=0}^{k-1} P_j(c, \sigma) h^{j+1} = T_k\left(\sum_{l=1}^{\infty} \frac{(\sigma^{-1})^{(l)}}{l!} P(h)^l\right).$$

The GEM_k iterations may be similarly written with the addition of a remainder term. Examples of Generalized Euler with Modification are the simple Runge-Kutta approximation to the solution of $dz/dt = -f(z)/f'(z)$ (see [S-SI]) and an incremental Laguerre method. For the latter do the following. Instead of letting $R = T_k\sigma^{-1}(h)$ as for Euler we let $(T_k\sigma \circ R) = h$ and solve for R. We can do this for $k = 2$. That is we solve $\sigma_2 R^2 + R = h$ for R yielding $R = (-1+\sqrt{1+4\sigma_2 h})/2\sigma_2$. For $h = 1$ this is Laguerre's method of Henrici (1977, p. 53) with $\gamma = 2$.

We now turn to an analysis of the GEM iterations with the goal of showing that GEM's are cheap, Theorem C.

Given $P(h)$ there is a $\delta > 0$ such that P is injective on the disc of radius δ, $D(\delta)$. Let K be the Lipschitz constant of P on $D(\delta)$, $K = \sup_{z\in D(\delta)} |P'(z)|$. Let $\gamma_k(c_1)$ be the first positive root of

$$(1-\gamma)^2 - 4c_1\gamma(1 + B(k+1)\gamma^k)$$

where $1 \leqq B < 1.07$ and B is a constant which makes the Bieberbach conjecture true (see [S-SI]). Finally, let

$$\gamma_k(P) = \min\left(\delta, \frac{1}{K}, \gamma_k(c_1)\right).$$

LEMMA 1. *Suppose that* $0 < h_* \leqq \min(1, h_1(f, z))$ *and that* $h = ah_*$ *for some complex number* a *with* $|a| = \gamma$ *and* $0 < \gamma < \gamma_k(P)$. *Then*:

a) $|\sigma^{-1}P(h)| \leqq c_1\gamma h_*/(1-\gamma)^2$;

b) $|\sigma^{-1}P(h) - T_k\sigma^{-1}P(h)| \leqq c_1 h_* B(k+1)\gamma^{k+1}/(1-\gamma)^2$;

c) $|T_k(\sigma^{-1}P(h))| < h_*/4$;

d) $T_k(\sigma^{-1}P(h) \in \sigma^{-1}(D(h_*))$.

Here $D(h)$ *is the disc of radius* h *around* 0.

Proof. Since $h_* \leqq (1/K)h_1$ $P(D(h_*)) \subset D(h_1)$ and since $h_* < \delta h_1$ $\sigma^{-1} \circ P$ is defined and injective on $D(h_*)$. $(\sigma^{-1} \circ P)_0' = c_1$ so $(1/c_1)(\sigma^{-1} \circ P)$ is defined and injective on $D(h_*)$. It has derivative 1 at 0. Now [S-SI, Lemma 7, § 2] applied to $(1/c_1)(\sigma^{-1}P)$ proves a) and b). By the triangle inequality

$$|T_k(\sigma^{-1}(P(h))| \leqq \frac{c_1\gamma h_*}{(1-\gamma)^2} + \frac{c_1 h_* B(k+1)\gamma^{k+1}}{(1-\gamma)^2}$$

and the right-hand side is $< h_*/4$ since $\gamma < \gamma_k(c_1)$. This proves c). Since $|T_k\sigma^{-1}(P(h)| < h_*/4$, [S-SI, Lemma 5, § 2] proves d).

LEMMA 2. *Let* $I_{h,f}(z) = z + F(z)R_{(h,f)}(z)$ *be a* GEM *iteration then there is a* δ *such that*

$$|R_{(h,f)}z)| < \frac{h_1(f, z)}{4}$$

for $0 < h < \delta \min(1, h_1(f, z))$.

Proof. There is a $P(h)$ and R_{k+1} such that $R_{h,f} = T_k(\sigma^{-1}(P(h)) + R_{k+1}$. Now use Lemma 1c with $h_* = \frac{1}{2}\min(1, h_1)$ and further choose $\delta < \gamma_k(P)$ small enough so that $|R_{k+1}| < h_1/8$ for $0 < h < \delta \min(1, h_1)$.

LEMMA 3. *Let* $I_{h,f}(z) = z + F(z)R_{(h,f)}(z)$ *be a* GEM_k *iteration. Then there is a* $\delta > 0$ *such that for* $0 < h < \delta \min(1, h_1)$, $\sigma^{-1}(\sigma \circ R) = R$.

Proof. By the Koebe theorem

$$\sigma^{-1}(D(h_1)) \supset D\left(\frac{h_1}{4}\right)$$

where $D(r)$ denotes the open disc of radius r around 0 in $\mathbb{C}$. By Lemma 2 there is a δ such that

$$|R_{(h,f)}(z)| < \frac{h_1(f,z)}{4}$$

for $0 < h < \delta \min(1, h_1(f,z))$. Thus $R_{(h,f)}(z) = \sigma^{-1}(h')$ for some $h' \in D(h_1)$ $\sigma \circ R_{(h,f)}(z) = h'$ and $\sigma^{-1}(\sigma \circ R_{(h,f)}(z)) = \sigma^{-1}(h') = R_{(h,f)}(z)$.

PROPOSITION 2. *If* $z' = I_{h,f}(z) = z + F(T_K\sigma^{-1}(P(h)) + R_{k+1}(h))$ *is a* GEM_k *iteration, then there are constant* $K > 0$, $\varepsilon > 0$ *such that*

$$(*) \qquad \frac{f(z')}{f(z)} = 1 - P(h) + S_{k+1}(h)$$

where $|S_{k+1}(h)| < Kh^{k+1} \max(1, 1/h_1^k)$ *for* $0 < h < \varepsilon \min(1, h_1)$.

Proof.

$$\frac{f(z')}{f(z)} = 1 - \sigma \circ R = 1 - \sigma(T_k\sigma^{-1}P(h) + R_{k+1}(h)).$$

There are K, $\delta > 0$ such that for $0 \leqq h < \delta \min(1, h_1)$

$$S_{k+1}(h) = \sigma(\sigma^{-1}(P(h)) - \sigma(T_k\sigma^{-1}P(h) + R_{k+1}(h))$$

and

$$|\sigma^{-1}(P(h))| < 2c_1h,$$

$$|\sigma^{-1}P(h) - (T_k\sigma^{-1}(P(h)) + R_{k+1}(h))| < Kh^{k+1} \max\left(1, \frac{1}{h_1^k}\right)$$

by Lemma 1, the definition of $R_{k+1}(h)$ and the triangle inequality. Now by the generalized Loewner theorem (see Smale (1981) say) $|\sigma_i|^{1/i-1} < 4/h_1$. Now apply [S-SI, Lemma 3, § 2] with $a = 4/h_1$, $b = 2c_1h$, $c = (k/2c_1)h^{k+1} \max(1, 1/h_1^k)$. Make sure that δ is small enough to guarantee $(1+c)ab < 1$.

Iterations satisfying $(*)$ were called efficiency k in [S-SI]. One of the fundamental estimates on speed is proven for iterations of efficiency k [SS-I, Thm. 4]. Since a polynomial f of degree d is generally d to 1 a point z' is only determined by $f(z')$ up to this d to 1 ambiguity. Efficiency k iterations are determined by the ratios $f(z')/f(z)$ and thus are determined up to this same ambiguity. We impose a continuity condition on iterations.

DEFINITION. The iteration $I_{h,f}(z) = z + FR(h,f,z)$ is called small iff there is a $\delta > 0$ such that $\sigma^{-1}(\sigma(R)) = R$ for all $0 < h < \delta \min(1, h_1)$.

Here, as above, σ^{-1} takes 0 to 0 and is defined and convergent on the disc of radius $h_1 = h_1(x,z)$.

THEOREM D. *The small iterations of efficiency k are precisely the* GEM_k *iterations.*

Proof. Proposition 2 proves that the GEM_k are efficiency k and Lemma 3 that the GEM_k are small. Now to prove the converse. If

$$z' = I_{h,f}(z) = z + FR$$

and

$$\frac{f(z')}{f(z)} = 1 - (P(h) + S_{k+1}(h))$$

where $|S_{k+1}(h)| < Kh^{k+1} \max(1, 1/h_1^k)$ for $0 < h < \varepsilon \min(1, 1/h_1^k)$ then

$$\sigma \circ R = P(h) + S_{k+1}(h);$$

and there is a $0 < \delta < \varepsilon$ such that

$$R = \sigma^{-1}(P(h) + S_{k+1}(h))$$

for $0 < h < \delta \min(1, h_1)$ since I is small. Thus it only remains to prove that for

$$R_{k+1} = \sigma^{-1}(P(h) + S_{k+1}(h)) - T_k\sigma^{-1}P(h),$$

there are $c > 0$, $\delta > 0$ s.t.

$$|R_{k+1}| < ch^{k+1} \max\left(1, \frac{1}{h_1^k}\right)$$

for $0 < h < \delta \min(1, h_1)$.

$$|R_{k+1}(h)| \leqq |\sigma^{-1}(P(h) + S_{k+1}(h)) - \sigma^{-1}(P(h))| + |\sigma^{-1}(P(h)) - T_k\sigma^{-1}(P(h))|.$$

We estimate the two terms on the right. For $h_* = \min(1, h_1)$ and $h = \gamma h_*$ with $\gamma < \gamma_k(P)$ Lemma 1b asserts that

$$|\sigma^{-1}(P(h)) - T_k\sigma^{-1}(P(h))| \leqq \frac{c_1 h_* B(k+1)\gamma^{k+1}}{(1-\gamma)^2}$$

so for $0 < h < \gamma_k(P)/2 \min(1, h_1)$

$$|\sigma^{-1}(P(h)) - T_k\sigma^{-1}(P(h))| \leqq 4c_1 \frac{h^{k+1}}{h_*^k} = 4c_1 h^{k+1} \max\left(1, \frac{1}{h_1^k}\right).$$

Now estimate

$$|\sigma^{-1}(P(h) + S_{k+1}(h)) - \sigma^{-1}(P(h))|$$

as in Proposition 2.

$$|\sigma_i^{-1}|^{1/i-1} < \frac{(Bi)^{1/i-1}}{h_1} < \frac{4}{h_1}.$$

For $0 < h < (\gamma_k(P)/2) \min(1, h_1) = (\gamma_k(P)/2)h_*$,

$$|P(h)| < \frac{1}{\gamma_k(P)} h \quad \text{and} \quad |\sigma^{-1}P(h))| < 4c_1 h$$

by Lemma 1a and there are δ_1, K such that

$$|S_{k+1}(h)| < Kh^{k+1} \max\left(1, \frac{1}{h_1^k}\right) = K\frac{h^{k+1}}{h_*^k}$$

for $0 < h < \delta_1 \min(1, h_1)$.

Thus by [S-SI, Lemma 3]

$$|\sigma^{-1}(P(h) + S_{k+1}(h)) - \sigma^{-1}(P(h))|$$

$$\leqq \frac{Kh^{k+1}/h_*^k}{(1-(4/\gamma_k(P))h/h_1)(1-(1+K\gamma_k(P)(h/h_*)^k)((4/\gamma_k(P))h/h_1))}.$$

Now it is easy to produce a δ such that for $0 < h < \delta h_*$, K divided by the denominator is bounded.

LEMMA 4. *Suppose* $|f(z_1)| \leqq |f(z)|$ *and that* $z_1 = f_z^{-1}((1-h)f(z))$ *for* $0 < |h| < \sin \Theta_{f,z_0}$, *then*

$$\Theta_{f,z_1} \geqq \Theta_{f,z} - \left| \arg \frac{f(z_1)}{f(z)} \right|.$$

Proof. Since $|f(z_1)| < |f(z)|$ it suffices to see that $z_1 \in f_z^{-1}(W_{f,z})$ see [SS-I, § 3] for then $f_{z_1}^{-1}$ is defined on a wedge of angle at least the angle of $W_{f,z}$ minus $|\arg (f(z_1)/f(z)|$.

But the open disc of radius $|f(z)| \sin \Theta_{f,z}$ centered at $f(z)$ is contained in $W_{f,z}$ as Fig. 3 shows. Thus $(1-h)f(z)$ is in this disc for $0 \leqq |h| < \sin \theta_{f,z}$ and $z_1 \in f_z^{-1}(W_{f,z})$.

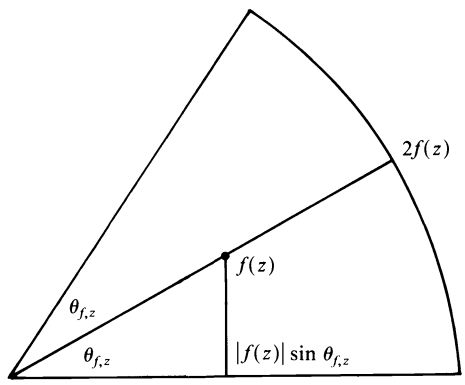


FIG. 3

LEMMA 5. *Let* $I_{h,f}(z)$ *be a* GEM_k *iteration. Then there is a constant* a, $1 \geqq a > 0$ *depending only on* I *such that*: *If* $0 < h < a \sin \Theta_{f,z}$, *then*

$$\Theta_{f,z'} \geqq \Theta_{f,z} - \left| \arg \frac{f(z')}{f(z)} \right].$$

Proof. By Lemma 4 we need only show that there is an a such that if $0 < h < a \sin \Theta_{f,z}$ then $z' = f_z^{-1}((1-h)f(z))$ for some h with $0 < |h| < \sin \Theta_{f,z}$. Now $f_z^{-1}((1-h)f(z)) = z + F\sigma^{-1}(h)$. Thus it suffices to show for $z' = I_{h,f}(z) = z + FR$ that $R(h, f, z) = \sigma^{-1}(h')$ for some h' with $0 < |h'| < \sin \Theta_{f,z}$. Since a GEM_k iteration is small, there is a $\delta_1 > 0$ such that for $0 < h < \delta_1 \min(1, h_1)$, $\sigma^{-1}(\sigma R(h, f, z)) = R$. We will let $h' = \sigma R(h, f, z)$. Since $\sin \Theta_{f,z} < \min(1, h_1)$ we know that $\delta_1 \sin \Theta_{f,z} < \delta_1 \min(1, h_1)$. Now we claim that there is an a, $0 < a \leqq \delta_1$ such that if $0 < h < a \sin \Theta_{f,z}$ then $|h'| = |\sigma R(h, f, z)| < \sin \Theta_{f,z}$. The last claim finishes the proof, for then $h' = \sigma R(h, f, z)$ satisfies $0 < |h'| < \sin \Theta_{f,z}$ and $\sigma^{-1}(h') = R(h, f, z)$. To verify the claim we may assume $\delta_1 \leqq 1$. Let L be the Lipschitz constant of $P(h)$ on $D(1)$ $\sigma R(h, f, z) = P(h) + S_{k+1}(h)$ where C, $\mu > 0$ with the property that

$$|S_{k+1}(h)| < Ch^{k+1} \max\left(1, \frac{1}{h_1^k}\right) \quad \text{for } 0 < h < \mu \min(1, h_1),$$

$$\max\left(1, \frac{1}{h_1^k}\right) = \frac{1}{\min(1, h_1)^k} < \frac{1}{\sin \Theta_{f,z}}.$$

Thus for $0 < h < \mu \sin \theta_{1,z}$

$$|S_{k+1}(h)| < \mu^{k+1} \sin \theta_{f,z}.$$

We may assume $\mu < \min(1/2L, 1/2)$ for $0 < h < \mu \sin\Theta_{f,z}\,|P(h)| < \frac{1}{2}\sin\Theta_{f,z}\,|S_{k+1}(h)| < (\frac{1}{2})^{k+1}\sin\Theta_{f,z}$ and we are done. □

We now can prove that GEM's are cheap. This is the analogy of [S-SI, Theorem 4] but for GEM's Λ_{f,z_0} can be replaced by Θ_{f,z_0}.

THEOREM C. *Suppose that $z' = I_{h,f}(z)$ is a GEM_k iteration. Then there is a constant K depending only on I such that: If $\Theta_{f,z_0} > 0$ and $|f(z_0)| > L > 0$ then there is an h given explicitly such that $|f(z_n)| < L$ for*

$$n = \left[K \left(\frac{\log |f(z_0)/L|}{\theta_{f,z_0}} \right)^{(k+1)/k} \right]$$

and $z_n = (I_{h,f})^n(z_0)$.

Proof. The proof is the same as [S-SI, Thm. 4]. Take a to be the min of the a considered there and the a of Lemma 5 above, and replace Λ_{f,z_0} by Θ_{f,z_0}.

Remark. It is easy to adapt the algorithms of Theorem A and B to GEM_k iterations and prove analogous theorems. We state a particular theorem which is the analogue of the "Main Theorem" of [SS-I] and which has the same proof starting from Theorem C.

THEOREM E. *If $z' = I_{h,f}(z)$ is a GEM_k iteration, there are positive constants K_1, K_2 depending only on $P(h)$ with the following true: Given $d > 1$, $1 > \mu > 0$ there are R, h such that: If $(z_0, f) \in S^1_R \times P_d(1)$ then $z_s = (I_{h,f})^s(z_0)$ will be an approximate zero of f with probability $1 - \mu$ for any $s \geqq K_1(d(|\log \mu|/\mu))^{(k+1)/k} + K_2$.*

4. This section consists of a series of problems and remarks.

(1) There is the general problem of comparing speeds of different algorithms for root finding of complex polynomials. See Dejon-Henrici (1969) and Henrici (1977) for a number of such algorithms. Comparison by experiment on machines can often be done readily. But the theoretical study of which algorithms are faster is another story. Such an analysis must deal seriously with round off errors and eventually consideration of the question: What are appropriate models of algorithms for this problem? Perhaps even the question must be confronted; what is the best model for the machine for this kind of study? Is a Turing machine always the right model? See the Traub-Wozniakowski (1982) critique of Khachiyan's work on linear programming for a good perspective on related questions.

For the comparison of algorithms lower bounds on speed are important. See Traub-Wozniakowski (1980).

Finally in this discussion, we note that we have just received the paper of Schönhage (1982). This takes a different approach to the study of speed related to the fundamental theorem of algebra. It seems to be quite an interesting work.

(2) We only have results for one variable. It is a wide open and central problem to extend the analysis to more than one variable, especially to polynomial maps from $\mathbb{C}^n$ to $\mathbb{C}^n$ (or $\mathbb{R}^n$ to $\mathbb{R}^n$). Since the theory of schlicht functions, used heavily here, is a one variable theory, finding the right estimates in several variables is a challenging problem. Relevant to this problem is that a mild algebraic condition on a polynomial map $f: \mathbb{C}^n \to \mathbb{C}^n$ guarantees that f is proper and hence has a zero, see Hirsch-Smale (1979).

(3) We have studied the problem of efficiency for finding one zero of a polynomial. What about the problem of finding all zeros of a polynomial f? The natural algorithms would follow paths in the space $I \times \mathbb{C}$, $I = [0, 1]$. Define $f_d(z) = \prod_{i=1}^{d}(z - \zeta_i)$ where $\zeta_1, \cdots, \zeta_d$ are the dth roots of unity. Let $F: I \times \mathbb{C} \to \mathbb{C}$ be defined by $F(t, z) = (1-t)f_d(z) + tf(z)$. Generally $F^{-1}(0) \subset I \times C$ consists of d curves leading from the

known ζ_i to zeros of f. One obtains good algorithms by following these curves simultaneously. We suspect that the speed in this case can be understood by the methods in these papers.

(4) We have used the space

$$P_d(1) = \{(a_0, \cdots, a_{d-1}, a_d) \mid |a_i| \leqq 1,\ a_d = 1\}$$

of coefficients of $f(z) = \sum_{i=0}^{d} a_i z^i$. While this parameterization has a simple immediacy, the projective space $\mathbb{C}^{d+1}/(\mathbb{C}-0)$ is more natural. Here $(a_0, \cdots, a_d) \in \mathbb{C}^{d+1}$ is equivalent to $(\lambda a_0, \cdots, \lambda a_d)$, each nonzero complex λ. It would seem reasonable for the results to go over. Also we have used one particular probability measure, the uniform one. It would be useful to make a generalization to a wide class of probability measures, say given axiomatically as in Smale (1982b). Finally the problem suggests itself to replace polynomials by other classes of comlex analytic functions. For example much of the analysis applies to rational functions which are also invertible up to the "first" critical value.

(5) We recall a problem, yet open, from Smale (1981) which has received some attention. For any polynomial f, $\deg f > 1$, and complex number z, $f'(z) \neq 0$, it seems likely that there exists a critical point $\theta (f'(\theta) = 0)$ such that

$$\left| \frac{f(z) - f(\theta)}{z - \theta} \right| \leqq |f'(z)|.$$

It is proved there that

$$\min_{\substack{\theta \\ f'(\theta)=0}} \left| \frac{f(z) - f(\theta)}{z - \theta} \right| \leqq K |f'(z)| \quad \text{with } K = 4.$$

One can express the conjecture in a slightly sharper form by making K a function of d, $K = K_d = d - 1/d$. This conjecture is the best possible as can be seen by choosing $f(z) = z^d - dz$, and $z = 0$. The conjecture is false for entire functions such as $f(z) = e^z$.

Dick Palais first pointed out to Smale that the estimate ($K = 1$) was true for f with real zeros. Nan Boultbee confirmed Smale's early calculations for degree $f \leqq 4$. And recently David Tischler (1982) has proved the conjecture when one root of f is zero and the others have the same absolute value. Linda Keen and Tischler have produced some supportive numerical evidence, but as mentioned above, the general conjecture remains open.

(6) We remark that although detailed techniques are different, there are basic similarities between the main theorems here and in Smale (1982a). Each gives a good estimate for the average number of iterations of a well-known algorithm or variation thereof. Moreover the underlying geometry of the algorithm in each case is following the inverse image of a segment in the target space.

The work of Kuhn-Zeke-Senlin (1982) and Renegar (1982) on the speed of piecewise linear algorithms to find zeros of polynomials relates to both our paper here and Smale (1982a).

(7) The algorithms in [S-SI] and in this paper start with $z_0 \in \mathbb{C}$ satisfying $|z_0| \gg 1$, e.g. $|z_0| = 3$. This is necessary for our analysis since the rough behaviour of $f \in P_d(1)$ on points z_0 with $|z_0|$ large enough is independent of f. On the other hand the large starting value contributes eventually to the d in the estimates of the main theorems. This suggests that if one started with $|z_0| \leqq 1$, e.g. $z_0 = 0$ at least that factor of d would be eliminated, sharpening the theorem drastically.

The problem here is to obtain information on the behavior of $\Theta_{f,z}$ for $|z|=1$ or even $z=0$. Consider the integral Θ_d of $\Theta_{f,z}$ over the space $P_d(1)\times S^1_1$ with the usual uniform probability measure. Is there an $\varepsilon>0$ independent of d such that $\Theta_d>\varepsilon>0$? A related question is: Do there exists universal constants $\varepsilon_1, \varepsilon_2>0$ such that $\int_{f\in P_d(1)}$ (measure of $\{z\in S^1 | \Theta_{f,z}>\varepsilon_1\})^{-1}<\varepsilon_2^{-1}$. An affirmative answer would imply that the d in d log d of Theorems A and B could be eliminated.

In the case of Theorem A this is very direct; Theorem B actually requires a slightly different algorithm. The idea is to switch to $h=1$ at some point. We develop such an algorithm a bit.

Let $\tilde{\rho}_f=\min(\frac{1}{2}c, \frac{1}{2}c\rho_f^{3/2})$ and $j=j_k=\lceil \log_{k+1}(8d\log_2(d))\rceil$. Here c is about $\frac{1}{12}$ as in the definition of approximate zero. It is a simple computation to check from the proposition of the introduction that:

LEMMA. *Suppose* $f\in P_d(1)$ *and* $\varepsilon_f>0$. *If* $|f(z_0)|<\tilde{\rho}_f$ *then* $|f(E^j_k(z_0))|<\varepsilon_f$.

Proof. If $\rho_f\leqq 1$ then $|f(z_0)|<(\rho_f^{1/2}/2)c\rho_f$. Thus

$$|f(E^j_k(z_0))|<\left(\frac{\rho_f^{1/2}}{2}\right)^{8d\log_2 d} c\rho_f\leqq\frac{c\rho_f^d}{2^{8d\log_2 d}}$$

$$\leqq\frac{c\rho_f^d}{2^{4d}2^{4d\log_2 d}}=\frac{c\rho_f^d}{(2d)^{4d}}<\varepsilon f.$$

If $\rho_f\geqq 1$ then $|f(E^j_k(z_0))|<c/(2d)^{4d}<\varepsilon_f$ by a similar calculation.

We are now ready to describe the Algorithm (N−E)′:

0) $m=1$
1) $k=\max(\lceil\log d\rceil, m)$
   $j=\lceil\log_{k+1}(8d\log_2 d)\rceil$
   $n=K(d+m)$
2) Pick z_0 with $|z_0|=3$ at random.
3) $z=E^j_{k,l,f}(E^n_{k,H,f}z_0)$.

If $|f(z)|<\varepsilon_f$ terminate and print "z is an approximate zero of f".

4) If not $m=m+1$ and go to 1.

Once $2^{-m}<\tilde{\rho}_f$ all that is needed for this algorithm to produce an approximate zero is to pick a z_0 with $\Theta_{f,z_0}\geqq\pi/12$. So in the mean the algorithm goes through six additional cycles of the loop. Thus for fixed f with $\varepsilon_f>0$, Algorithm (N-E)′ terminates with probability one.

Let $m(f)=\lceil\log\tilde{\rho}_f\rceil$.

$$S(f)\leqq m(f)[K(d+m(f)+j_2]+\sum_{k=1}^{\infty}\frac{1}{6^k}(K(d+m(f)+k)+j_{m(f)+k})$$

$$\int_{f\in P_d(1)} m(f)<C_1\log d \quad \text{for } C_1 \text{ a constant.}$$

This shows that Theorem B applies to Algorithm (N-E)′. The extra computations involved in increasing k do not seriously effect the total number of arithmetic operations either.

REFERENCES

A. BORODIN AND I. MUNRO, (1975), *The Computational Complexity of Algebraic and Numerical Problems*, American Elsevier, New York.

R. BRENT AND KUNG (1976), *$O((n \log n)^{3/2})$ algorithms for composition and reversion of power series*, in Analytic Computational Complexity, J. F. Traub, ed., Academic Press, New York, pp. 217-225.

——— (1978), *Fast algorithms for manipulating formal series*, J. Assoc. Comput. Mach., 25, pp. 581-595.

B. DEJON AND P. HENRICI (1969), *Constructive Aspects of the Fundamental Theorem of Algebra*, John Wiley, New York.

P. HENRICI 1977, *Applied and Computational Complex Analysis*, John Wiley, New York.

M. HIRSCH AND S. SMALE (1979), *On algorithms for solving $f(x) = 0$*, Comm. Pure Appl. Math., 32, pp. 281-312.

D. KNUTH (1981), *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 2nd ed., Addison-Wesley, Reading, MA.

H. KUHN, W. ZEKE AND X. SENLIN (1982), *On the cost of computing roots of polynomials*, preprint.

S. LANG (1965), *Algebra*, Addison-Wesley, Reading MA.

J. RENEGAR (1982), *On the complexity of a piecewise linear algorithm for approximating roots of complex polynomials.*

A. SCHÖNHAGE (1982), *The fundamental theorem of algebra in terms of computational complexity-preliminary report*, Math. Inst. der Univ. Tübingen.

M. SHUB AND S. SMALE (1982), *Computational complexity; on the geometry of polynomials and a theory of cost* (*Part* I), Ann. Scient. Ec. Norm. Sup. 4 serie, 18 (1985).

S. SMALE (1981), *The fundamental theorem of algebra and complexity theory*, Bull. Amer. Math. Soc., (New Series), 4, pp. 1-36.

——— (1982a), *On the average speed of the simplex method of linear programming*, preprint.

——— (1982b), *The problem of the average speed of the simplex method*, to appear in the Proceedings of the International Symposium of Mathematical Programming, Bonn, 1982, Springer, Heidelberg.

D. TISCHLER (1982), preprint.

J. TRAUB AND H. WOZNIAKOWSKI (1980), *A General Theory of Optimal Algorithms*, Academic Press, New York.

——— (1982), *Complexity of linear programming*, Oper. Res. Lett., 1, pp. 59-62.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
012

A PROVABLY GOOD ALGORITHM FOR THE TWO MODULE ROUTING PROBLEM*

BRENDA S. BAKER†

Abstract. In the Mead-Conway design methodology for LSI, modules are designed and then connected by wires to form larger modules in a hierarchical fashion. It would be helpful to have a design aid that would do the routing automatically and be guaranteed of coming within some fixed percentage of the size of an optimal routing. With this goal in mind, we investigate the problem of routing two-terminal nets between two modules of the same width but possibly different heights, assuming that the sides are aligned vertically. The terminals may lie on any of the sides of either module. Wires must be routed according to the "Manhattan" reserved-layer model, in which all wires must lie on a rectilinear grid, and wires running the same direction must be separated by at least unit distance. Finding an optimal routing for this problem is NP-hard, where the measure of performance is the perimeter of the bounding box around the whole routing region. We describe an algorithm whose worst-case performance is asymptotically at most 19/10 times that of an optimal routing. The algorithm runs in $O(n \log n)$ time, where n is the number of nets.

One of the problems encountered in routing is how to evaluate a routing when the optimal routing is not available for comparison. The techniques given here can be used to calculate lower bounds on the size of an optimal routing. Thus, these techniques may be useful in evaluating routings produced by methods other than the algorithm in this paper.

Key words. wire routing, channel routing, NP-hard, worst-case performance bound

1. Introduction. In the Mead-Conway design methodology for LSI, modules (cells) are designed and then connected by wires to form larger modules in a hierarchical fashion. The modules may be objects like PLAs that are generated automatically from logic descriptions, or they may be individually designed cells performing some particular function. It would be helpful to have software to do the routing of the interconnection wires automatically. With this goal in mind, this paper develops and analyzes an algorithm for routing two-terminal nets between two rectangular modules that can have terminals on any of their sides. The modules must have the same width and be aligned vertically. The ideas in this paper could also be extended to multiterminal nets.

For our wiring model, we use the Manhattan reserved-layer wiring model which is designed for a technology in which two layers are available for wires and a routing discipline in which one layer is reserved for horizontal wires and the other for vertical wires. Thus, it specifies that all wire segments must be vertical or horizontal, and that wire segments running the same direction must be separated by at least unit distance. Perpendicular wires may cross. Wires must also stay at least unit distance from sides of the modules except where they connect to terminals. For convenience, we assume that all terminals, wire segments, and sides of modules lie on the lines of a unit grid. Vertical and horizontal lines of the grid are termed *columns* and *tracks*, respectively.

In our two-module routing problem, the modules are rectangles with the same width but possibly different heights. Their left and right sides are aligned as shown in Fig. 1. The distance between the modules is not fixed, i.e. it may be adjusted by the routing algorithm. Nets are specified as pairs of terminals that may lie on any of the four sides of the two components. The goal is to connect the pair of terminals for each net with a wire, subject to the constraints of the Manhattan model.

* Received by the editors March 13, 1983, and in revised form October 22, 1984.
† AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

An algorithm can only be evaluated with respect to some performance measure. Two obvious possible measures are the area and perimeter (in grid units) of the bounding box of the routing region, including the modules and all wires, as shown in Fig. 1. Note that of all rectangles with the same area, a square has the smallest perimeter, while for all rectangles of fixed perimeter, the smallest area is achieved by letting one dimension approach zero. Therefore, it is reasonable to expect that optimizing with respect to perimeter may tend to produce aspect ratios close to 1, which are desirable for some applications, while optimizing with respect to area may not.

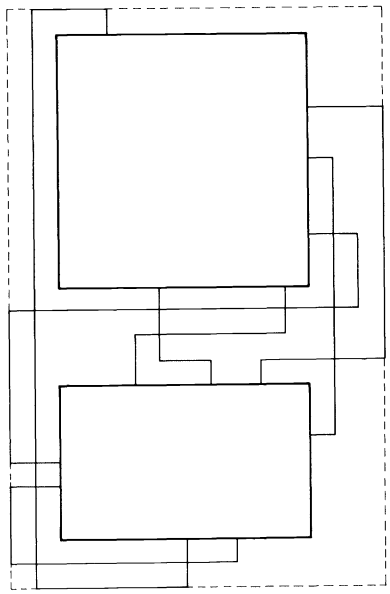


FIG. 1. *An example of two-module routing. The bounding box around the routing region is drawn with dotted lines.*

In this paper, we describe an $O(n \log n)$ time algorithm PAIR-SPLIT (PS) for the two-module problem. We show that for every two-module routing problem, the perimeter of the routing produced by PS is at most a constant plus 19/10 times the perimeter of an optimal routing. For channel routing, a subproblem of the two-module routing problem in which all terminals are on the facing sides of the modules, we give an algorithm SPLIT that produces a routing whose perimeter is never worse than a constant plus $\frac{3}{2}$ times the perimeter of an optimal routing. Examples are given to show that these performance bounds are tight. These algorithms are among a small number of routing algorithms [1], [3], [5], [9], [10], [11], [12], [14] that have been analyzed to be provably good in some sense.

Channel routing is an aspect of the two-module problem that is difficult to handle in the general case. In fact, producing an optimal routing is NP-complete, even for two-terminal nets, using either the channel width or perimeter as a measure of performance [13]. The channel-routing algorithm SPLIT used in PS is guaranteed always to find a routing, unlike some previous channel routing algorithms. However, it will not perform as well in many cases as other heuristics that may have worse worst-case performance bounds or that handle only restricted instances of the problem, but still seem to perform well on real applications. Such an algorithm can be substituted for

the general channel routing algorithm given here, since the rest of PS is independent of the channel routing algorithm used. Thus, the algorithm of this paper guarantees that a routing can always be achieved for any instance of the two-module problem, while the flexibility afforded by substitutability of other channel routing algorithms allows for potential improvements for some instances of the problem.

It is also possible that use of a different channel-routing algorithm could lead to a better worst-case bounds for PS. An obvious candidate is the algorithm of Baker, Bhatt, and Leighton [1] that produces routings for which the channel width is asymptotically within a constant times optimal. Its behavior has not been analyzed with respect to the perimeter measure, but very likely its worst-case performance is better than that of SPLIT with respect to the perimeter measure, and its use might improve the worst-case performance of PS as well.

The two-module problem is also related to two previously studied problems other than channel routing. For a single module with two-terminal nets whose terminals can be on any side, LaPaugh [9] and Gonzalez and Lee [7] give polynomial time algorithms that produce routings optimal for both area and perimeter. For a restricted case of the two-module problem, Chandrasekhar and Breuer [3] give an algorithm that produces a routing that is area-optimal over a class of routings that excludes certain paths for wires and sharing of tracks between wires for certain types of nets.

One of the difficulties in routing is knowing how good a routing is when the optimal routing is not available for comparison. Since the size of the PS routing is bounded compared to that of the optimal routing, the PS routing can be use to estimate the size of an optimal routing and hence the degree to which the current routing could potentially be improved. The time consumed in obtaining the estimate would be low thanks to the fast running time of PS. Thus, even in cases where other heuristics perform better than PS, the PS routing may be useful. In fact, the techniques used to obtain the performance bound for PS can be used directly to compute lower bounds on the size of the optimal routing.

Section 2 discusses the difficulties inherent in the two-module problem, the approach used by the PS algorithm, and the basic ideas used in the proof of the performance bound. Section 3 gives the complete description of SPLIT and PS. Section 4 proves the performance bounds for both SPLIT and PS.

2. Challenges presented by the two-module problem. This section discusses where the difficulties lie in finding a routing for the two-module problem, the general approach of PS, and the proof techniques used in obtaining the performance bound.

We begin by describing the channels available for routing. The region available for routing may be thought of as a figure eight around the modules, which are denoted by M_T and M_B. For our two-module problem, the figure eight may be divided into labeled (finite or semi-infinite) rectangular channels by extending either the horizontal or vertical sides of the modules, as shown in Fig. 2. Note that the two divisions give

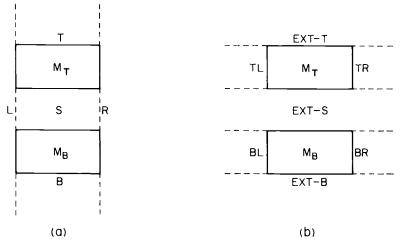


FIG. 2. *Channels around M_T and M_B.*

us overlapping labelled channels; both divisions into channels will be useful. In the figure, *T*, *B*, *L*, *R*, and *S* stand for top, bottom, left, right, and street, respectively. *EXT-T*, *EXT-S*, and *EXT-B* stand for "extensions" of channels *T*, *S*, and *B*, respectively.

Finding a routing for the two-module problem has two distinct but interrelated aspects: *channel assignment* and *channel routing*. Channel assignment is the determination of a set of channels to be crossed by the wire for each net, without determining the specific tracks or columns to be traversed within each channel. Once the set of nets crossing or entering each channel has been determined by channel assignment, channel routing is the determination of the exact path (specific tracks and columns) each wire is to follow within the channel.

Issues in channel assignment. Ideally, we would like to assign channels to nets as in some perimeter-optimal routing. For some nets, channel assignment is easy. For example, if a net has both terminals in channel *T*, it should be routed with a wire entirely within channel *T*. There may be optimal routings in which the wire goes all the way around M_T and/or M_B. However, for any such routing, there is another optimal routing with the wire entirely within channel *T*. This observation holds because a wire crossing the entire horizontal extent of channels *S* or *B* must use an entire track, while a wire going directly between the terminals in channel *T* can do no worse than to use an entire track.

Unfortunately, for some nets, optimal channel assignment cannot be done for each net independently of other nets. For example, compare Figs. 3a and 3b. The nets with terminals labeled A and B on the top and bottom of M_T must be routed in opposite directions in Fig. 3a but in the same direction (to the left) in Fig. 3b to achieve an optimal routing. Which channel assignment leads to an optimal routing depends on what nets are available to share tracks. This example illustrates the difficulties in channel assignment inherent even in routing around a single rectangular module. LaPaugh's algorithm [9] for an optimal solution for this problem runs in time $O(n^3)$, where n is the number of nets, and is very complicated. More recently, Gonzalez and Lee [7] found an $O(n \log n)$ time algorithm for the same problem.

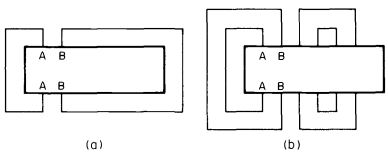


FIG. 3. *Two examples of optimal channel assignment.*

The approach in PS is to use simple heuristics for channel assignment, rather than to attempt to find an optimal assignment. Depending on the location of the terminals, each net is routed either along the shortest path between the terminals, or in a direction that will allow a segment of its path to share a track or column with a segment of another net of the same type.

Issues in channel routing. Once channel assignment has been done, individual channels must be routed. Note that channels *S* and (equivalently) *EXT-S* can have terminals on both the top and bottom edges, while other channels have terminals only on one side. Channel routing is easy in the latter case but can be difficult in the former case.

If a channel has terminals only on one side, the n nets with terminals in the channel can be routed optimally in $O(n \log n)$ time by a simple *interval coloring* algorithm [6]. Consider a horizontal channel and a set of two-terminal nets with both terminals on the bottom edge of the channel. Each net is routed using exactly one *jog*

(horizontal wire segment) connecting vertical segments attached to its terminals. Thus, the horizontal interval traversed by the net is simply the interval between its terminals. Nets are assigned tracks (colors) as follows. The nets are processed in order of increasing leftmost endpoint. Each net is assigned to the first track in which it can be placed without overlapping the interval of any net already in the track. This algorithm uses a number of tracks that is equal to the channel *density*, which is the maximum number of nets whose intervals include any single column. Since any routing must use at least this many tracks, the interval coloring routing is optimal.

Now, consider a horizontal channel that has terminals on both its top and bottom edges, as in Fig. 4. If no two terminals lie in the same column, interval coloring can still be used to produce an optimal routing. When columns can contain two terminals, routing is more difficult. Suppose a terminal of net n_2 lies above a terminal of net n_1 in the same column. Since two vertical wire segments cannot overlap in the reserved two-layer model, the jog of n_1 with one end in this column must lie below the jog of n_2 with one end in this column. These constraints may be described by a *constraint graph*. The constraint graph is a directed graph in which the nodes are the nets, and an arc is directed from node n_1 to node n_2 if and only if a terminal of net n_1 lies directly below a terminal of net n_2 in the same column. The constraint graph of the routing problem of Fig. 4 is given in Fig. 5.

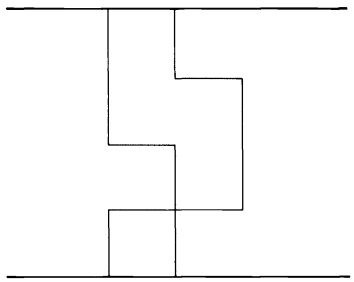


FIG. 4. *A channel routing problem containing a cycle.*

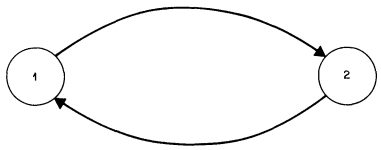


FIG. 5. *Constraint graph for the nets of Fig.* 4.

If the constraint graph contains no cycles, then the nets can be routed with a single jog each, by Deutsch's algorithm [4], for example. However, such a routing may not be optimal; in fact, there are examples for which such a routing requires n tracks while an optimal routing requires only on the order of $\sqrt{n}$ tracks [2].

A cycle in the constraint graph prevents routing every net with at most one jog (see Fig. 4, for example). Since every node has degree two, distinct cycles in the constraint graph must be disjoint. A cycle may be handled by using a *dogleg* (vertical wire segment connecting two horizontal segments) for the net corresponding to one node in the cycle, as in Fig. 4. Note that the dogleg may be forced to lie in a column outside the region between the modules, if there are no free columns within this region.

Thus, we must regard the channel available for routing as extending beyond the left and right sides of the modules (i.e. the extended street rather than just the street). It is possible to get by with a single extra column by choosing one net from each cycle and giving it a dogleg in this column [8]; however, in this solution no nets in cycles can share tracks, and if every net is in a cycle, the routing will require $n+c$ tracks, where n is the number of nets and c is the number of cycles.

Our approach is to find a compromise between sharing of tracks and use of extra columns for doglegs that requires at most $d+6$ extra columns and at most $(n+d)/2+4$ tracks, where d is the channel density and n is the number of nets. Since an optimal routing uses at least d tracks and the channel has at least n columns, the algorithm produces a routing whose perimeter is at most a constant plus $\frac{3}{2}$ that of an optimal routing.

Proof techniques. Part of the value of this work lies in showing that some rather simple proof techniques can be developed into an analysis of PS giving a tight worst-case performance bound. Let P be an instance of the two-module routing problem, PS (P) the routing obtained by PS, and OPT (P) a routing of P that is optimal with respect to the perimeter measure. Obtaining a worst-case performance bound requires analyzing both PS (P) and OPT (P). A key idea in the proof is that it is possible to derive lower bounds on the number of tracks and columns that OPT (P) must use for a net without knowing precisely how the net is routed. A simple example of such an analysis was given above (in the discussion of channel assignment) for nets with both terminals in channel T. In the proof, lower bounds are derived for OPT (P) based on three ideas:

(a) Certain nets require whole tracks or columns within some channel no matter how they are routed. For example, a net with one terminal in channel L and one terminal in channel R needs to get its wire from one side to the other within T, S, or B. The number of such nets within all such channels gives a lower bound on the sum of the channel widths.

(b) PS (P) is only worse than OPT (P) insofar as some nets are routed differently in the two routings. For each channel, the nets can be divided into two classes according to whether they are routed through the same part of the channel in both routings. Since certain nets in PS (P) are routed the shortest way, the nets routed differently in OPT (P) may be forced to use extra tracks or columns. For example, a net with both terminals in channel T is always routed by PS with a wire that has one horizontal segment extending between the terminals, contributing at most one to the width of the bounding box and at most two to the perimeter. If the wire for the same net in OPT (P) does not cover the interval between the terminals, it must go almost all the way around one or both modules.

(c) To formalize how certain routings use more tracks and columns than others, we assign weights to wires. Given a routing, let each wire accumulate a weight of $\frac{1}{2}$ for each time it crosses each dashed line in Fig. 6, and a weight of $\frac{1}{2}$ for each terminal in channels L, R, and S. Assign a weight of 0 to other wire segments and to other terminals. A single track can contain at most two horizontal wire segments of weight $\frac{1}{2}$ in channels T or B, or two terminals of weight $\frac{1}{2}$ in channels L or R. Similarly, a single column can contain at most two vertical segments of weight $\frac{1}{2}$ in channels L or R, or two terminals of weight $\frac{1}{2}$ in channel S. Consequently, the perimeter of OPT (P) is at least twice the sum of the weights of all the wires and terminals in OPT (P). Since the actual routing of OPT (P) is not available, a lower bound on the weight of each wire is deduced from the locations of its terminals. For example, a wire with both terminals in channel T has a weight of zero if it is routed straight across from one terminal to the other. On the other hand, if it is routed so that it does not cover all of

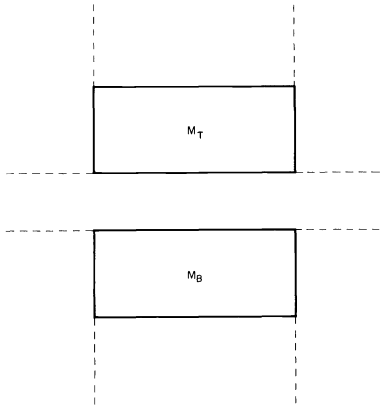


FIG. 6. *A wire accumulates weight* $\frac{1}{2}$ *for each crossing of a dashed line.*

this interval, it goes around one or both modules; in the process, it crosses at least two vertical lines and at least two horizontal lines for a total weight of at least two.

Developing the above ideas into a tight worst case performance bound requires a large amount of case analysis and algebraic manipulation. The details of the analysis are quite complicated, partly due to the large number of types of nets.

3. The algorithm. In this section we describe the two-module routing algorithm PS. PS calls a procedure PAIRUP for routing nets around a single rectangle, as well as the channel-routing algorithm SPLIT. We begin by describing PAIRUP and SPLIT.

PAIRUP. Let M be a rectangular module. Let N be a set of nets, each of which has one terminal on the top of M and the other on the bottom. The choice in channel assignment for each net is whether to route it around the module to the left or to the right. The goal is to assign directions (and hence channels) that allow the following channel routing phase to arrange for nets to share tracks, as in Fig. 7.

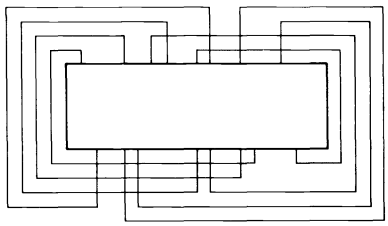


FIG. 7. *Applying* PAIRUP *around* M_B.

Number the terminals on each edge from left to right in increasing order. Define *net* (T, i) to be the net in N containing the ith top terminal, for $1 \leqq i \leqq |N|$. Define *net* (B, i) similarly for bottom terminals. For a net n, let *bottom* (n) be the number of n's bottom terminal.

```
procedure PAIRUP (N)
        begin
        constant left = 0, right = 1;
        var alt: boolean;
        for all nets n ∈ {N}, let direction (n) be undefined;
        alt = left;
```

```
while ∃n ∈ N such that direction (n) is undefined
    begin
    let t be the number of the leftmost top terminal such that
            direction (net (T, t)) is undefined;
    if for every b, bottom (net (T, t)) < b ≦ |N|,
            direction (net (B, b)) is defined
            then begin
            direction (net( T, t)) = alt;
            alt = not (alt);
            end
    else
            begin
            direction (net (T, t)) = left;
            for some b > bottom (net (T, t)) such that
                    direction (net (B, b)) is undefined
                    direction (net (B, b)) = right;
            end
    end
end
```

In the case in which N is the only set of nets to be routed, the channel assignment produced by PAIRUP can be extended to a routing in which all jogs are paired in tracks except for possibly two jogs. In particular, jogs may be paired as follows (see Fig. 7). Pairs of nets that are assigned opposite directions together in the loop of PAIRUP can share tracks for both their top and bottom jogs. Nets that are assigned a direction that is the current value of alt form a sequence in which the top jog of an odd net in the sequence can be paired with the top jog of the next even net, and the bottom jog of an even net in the sequence can be paired with the bottom jog of the next odd net. Thus, all but possibly two jogs are paired.

PAIRUP may be implemented by using linked lists for terminals on the top and bottom not assigned directions, or by using variables to keep track of the leftmost unassigned top terminal and rightmost unassigned bottom terminal. Except for constructing linked lists or tables for *net* and *bottom*, which might require $O(n \log n)$ time for sorting, PAIRUP runs in linear time.

PAIRUP and the associated channel routing solve a subproblem of the problem studied by LaPaugh [9] and Gonzalez and Lee [7]. Their problem is more general in allowing terminals to lie on any side. The PAIRUP approach comes within one track of optimal for the subproblem, since n nets must use at least n tracks for $2n$ jogs. Fig. 8 shows that the algorithm can in fact be one worse than optimal. PAIRUP is simpler than the algorithms of [7], [9].

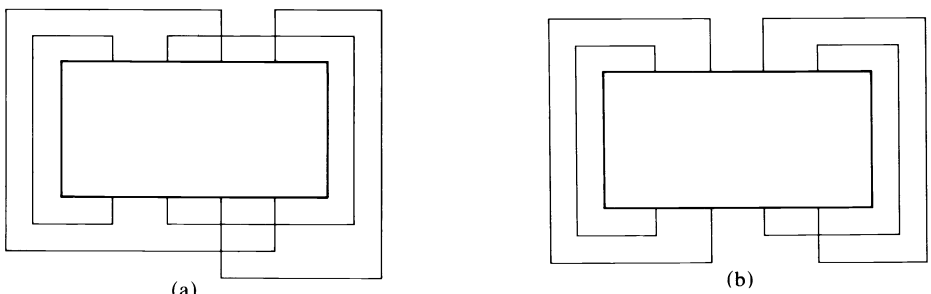


FIG. 8. *An example on which* PAIRUP *does not produce an optimal routing.* (a) *A routing produced by* PAIRUP. (b) *An optimal routing.*

SPLIT. SPLIT routes all nets with both terminals in the same column by running a vertical wire within this column. In the following, we therefore ignore such nets and columns and consider only a set N of nets whose top and bottom terminals lie in different columns.

Recall from § 2 that the difficulty in channel routing arises from constraints between nets whose terminals lie in the same column. The SPLIT algorithm splits the channel at some column and breaks all constraints between the nets to the left of the column and the nets to the right of the column by using doglegs for nets crossing this column. Consequently, the nets to the left of the column can be routed independently of the nets to the right of the column.

Let c be the number of columns in the modules bordering the channels. Number the columns of the modules from left to right. For i, $1 \leqq i \leqq c$, define $S(i)$ to be the set of nets with one terminal in columns 1 through i and one terminal in columns $i+1$ through c, $S^L(i)$ to be the set of nets with both terminals in columns 1 through i, and $S^R(i)$ to be the set of nets with both terminals in columns $i+1$ through c. It is easy to verify that for some r, $1 \leqq r \leqq c$, $S^L(r)$ contains exactly

$$\left\lfloor \frac{|N| - |S(r)|}{2} \right\rfloor$$

net and $S^R(r)$ contains exactly

$$\left\lceil \frac{|N| - |S(r)|}{2} \right\rceil$$

nets. SPLIT breaks the constraints between nets in $S^L(r)$ and $S^R(r)$ by using doglegs for nets in $S(r)$. Nets in $S^L(r)$ and nets in $S^R(r)$ are then routed independently using the same set of tracks.

The organization of the routing is shown in Fig. 9. First, all nets in $S(r)$ with both terminals on top are routed with a single jog in the tracks immediately below the top of the channel, one track per net. Let T_1 denote the set of these tracks. Below T_1, all nets in $S(r)$ with both terminals on the bottom are routed with a single jog, one track per net. Let T_2 denote the set of these tracks. Between T_1 and T_2 is an imaginary rectangle (outlined in dashes in Fig. 9) that will contain the jogs of wires in $S^L(r)$ and $S^R(r)$. The nets in $S(r)$ with one terminal on top and one on the bottom will be routed with wires containing doglegs and running around this rectangle. The doglegs will lie to the left or right of all columns with terminals. To decide on the direction for each dogleg, PAIRUP will be applied to these nets. Note that for PAIRUP, it does not matter that the terminals are not located on the rectangle itself; only the horizontal positions of the terminals are needed. After PAIRUP has determined the direction of each dogleg, the jogs of these nets may be paired in tracks as described in § 2. The paired jogs will be laid out between T_1 and T_2 such that all top jogs lie above the

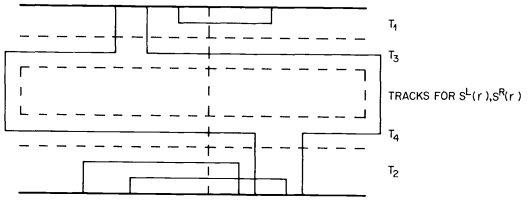


FIG. 9. *Routing the nets in* $S(r)$.

imaginary rectangle and all bottom jogs lie below it (as shown in Fig. 9). Let T_3 denote the set of tracks used for the top jogs and T_4 the set of tracks used for the bottom jogs.

The nets in $S^L(r)$ and $S^R(r)$ will be routed in tracks lying in the rectangle between T_3 and T_4. By placing the jogs for these nets in the region between T_3 and T_4, all constraints between nets of $S(r)$ and those of $S^L(r)$ and $S^R(r)$ are automatically satisfied. At most $|S(r)|+1$ tracks are used for $S(r)$.

The nets in $S^L(r)$ and $S^R(r)$ are routed independently and symmetrically. We will describe only the routing of $S^R(r)$. If there are no cycles in the subgraph of the constraint graph induced by $S^R(r)$, then each net can be routed in a separate track with a single horizontal jog as long as the tracks are ordered so as to satisfy any constraints. However, if there are constraint cycles, it is necessary to break the cycles by using additional outside doglegs.

Recall that distinct cycles are disjoint. Let Y be a set of nets which includes exactly one net from each cycle in the subgraph induced by $S^R(r)$. Each column contains at most one terminal belonging to a net in Y. Each net in Y will be routed using an outside dogleg. The doglegs to the left of the terminals will lie in a single column, while the doglegs to the right of the terminals will lie in one of three columns c_1, c_2, and c_3 to the right of all the terminals. For the left column, we pick the leftmost column containing a terminal of a net y in Y. Let c_y denote this column. Net y will be routed using a dogleg in column c_3. Its top jog lies in the track immediately below T_3 and its bottom jog lies in the track immediately above T_4.

PAIRUP is applied to $Y-\{y\}$ to assign a direction to each dogleg, and the jogs are paired in tracks as described in § 2. Recall that PAIRUP may assign some pairs of nets opposite directions so that the top jogs may occupy the same track and the bottom jogs may occupy the same track. These pairs of nets are laid out as suggested by Fig. 10, with successive pairs occupying successively lower tracks, and all doglegs in column c_y or c_1. Recall that the remaining nets form a sequence in which the top (bottom) jog of an odd (even) net is paired with the top (bottom) jog of the next even (odd) net, with the bottom jog of the first net and the top jog of the last net unpaired.

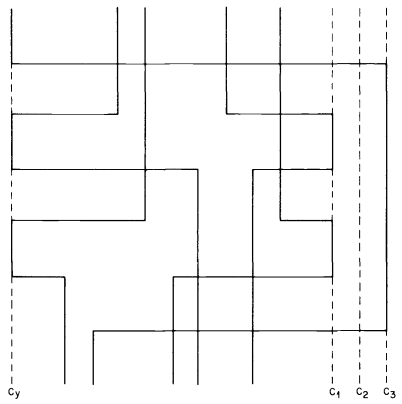


FIG. 10. *Paired nets breaking cycles in* SPLIT.

These nets are laid out as suggested by Fig. 11. First, all of the odd nets in the sequence, i.e. the nets with doglegs toward the left, are laid out in successively lower tracks. Then all of the even nets, *i.e.* the nets with doglegs toward the right, are laid out so that their jogs are in the same tracks as the jogs of the odd nets that they have been paired with. The doglegs toward the right must alternate between column c_1 and column c_2.

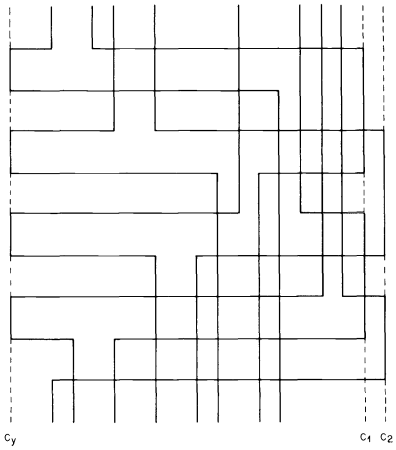


FIG. 11. *Laying out a sequence of cycle-breaking nets.*

The remaining nets of $S^R(r) - Y$ are laid out using one jog and one track each. In particular, the jogs of nets belonging to a cycle must be placed in tracks lying between the top and bottom jogs of the net of Y of the same cycle (see Fig. 12). Moreover, all of the tracks for jogs of $S^R(r) - Y$ must be ordered so as to satisfy the acyclic constraints between them.

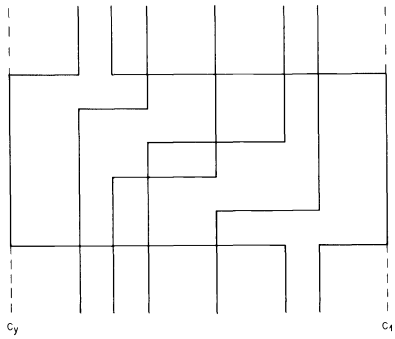


FIG. 12. *Adding the remaining nets to a pair of cycles.*

SPLIT can be implemented to run in $O(n \log n)$ time. In fact, aside from sorting the terminals, it can be implemented to run in linear time.

PS. Since PS treats nets in different ways according to the location of their terminals, we begin by categorizing nets (see Fig. 13). Recall that Fig. 2 defined some labelled channels around the modules. A net is a *straight* net if both terminals lie in channel X, for X one of L, R, T, B, or S. It is a *corner* net if one terminal lies in channel L or R and the other lies in channel T, S, or B. It is a *cross* net if one terminal lies in channel TL and the other in channel BR, or if one terminal lies in channel BL and the other in channel TR. Finally, it is an *opposite* net if one terminal lies in channel TL and the other in channel TR, or if one terminal lies in channel BL and the other in channel BR, or if the terminals lie in two of channels T, S, and B. Note that these categories of nets are pairwise disjoint and that every net is either a straight, corner, cross, or opposite net. A straight net is also a *street* net if its terminals both line in channel S.

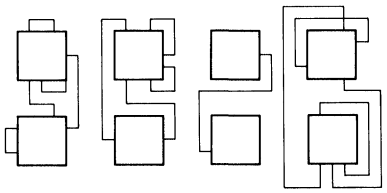


FIG. 13. *Examples of each class of local nets. (a) Straight nets. (b) Corner nets. (c) Cross nets. (d) Opposite nets.*

The PS algorithm has five stages:

(a) Channel assignment, to be described subsequently.

(b) Channel routing for channels T and B, done via interval coloring. Recall that channels T and B are bounded on the left and right by vertical lines passing through the sides of the modules. For the application of interval coloring, the interval assigned to a net is the minimal horizontal interval including its terminals and/or points of entry into the channel at the sides. (The corners to the right and left of channels T and B cannot be routed until the sides have also been routed.)

(c) Tentative channel routing for (i) the union of channels *TL* and *BL* (excluding the region between them) and (ii) the union of channels *TR* and *BR* (excluding the region between them), done via interval coloring. The region between each pair of channels is ignored at this point because the nets passing from these channels into channel S have not yet been assigned to tracks. For the application of interval coloring, the interval assigned to a net is the union of the minimal vertical intervals including its terminals and/or points of entry for channels *TL* and *BL* or *TR* and *BR*.

(d) Channel routing for channel *EXT-S* (to be described subsequently), where wires entering *EXT-S* from above or below (from channels *TL*, *TR*, *BL*, or *BR*) are treated as if they have terminals at their entry points.

(e) Final channel routing for L and R, done via interval coloring. The interval for a net is the minimal vertical interval including the terminals and/or points of entry into the channel. Note that the routing of channels S, T and B has fixed the tracks on which nets enter channels L and R from them.

Since interval coloring was described in § 2, it is only necessary to elaborate on (a) and (d). After that, an example of a PS routing is given and the running time of PS is analyzed.

Channel assignment. Channel assignment is straightforward for all nets except opposite nets. In particular, a street net remains within channel *EXT-S*. A straight net other than a street net remains within whichever one of channels L, R, T, or B contains its terminals. A corner net is assigned the channels crossed by a shortest wire connecting its terminals and satisfying the Manhattan reserved-layer constraints. A cross net is assigned channel *EXT-S* and the channels containing its terminals, namely *TL* and *BR* or *BL* and *TR*. Examples of routings based on these channel assignments are given in Fig. 13(a)-(c).

Groups of opposite nets with terminals on the same sides of modules are treated together in channel assignment. Thus, there are seven groups, according to whether the terminals are: (i-ii) on the left and right sides of the same module, (iii-iv) on the top and bottom of the same module, (v) on the top of M_T and the bottom of M_B, (vi) on the top of the two modules, or (vii) on the bottom of the two modules.

Channel assignment for opposite nets is determined by applying PAIRUP to each of the seven groups separately. For groups (v)-(vii), where the terminals are on two different modules, PAIRUP is applied as if the terminals are on opposite sides of a

single module, but in the same order. For these groups, "top" and "bottom" terminals are determined in the obvious way for PAIRUP. For the other groups, "top" and "bottom" are chosen to reduce the additive constant in the performance bound. In particular, PAIRUP is applied to groups (i) and (ii) as if the "right" side lies in channel S, and to groups (iii) and (iv) as if the "top" lies in channel S.

Channel routing for channel EXT-S. Channel routing is done by a slightly modified version EXSPLIT of SPLIT. EXSPLIT produces somewhat better routings for the situation in PS, where some "terminals" in channel *EXT-S* are actually wires entering channel *EXT-S* from above or below.

SPLIT may be applied directly to channel *EXT-S* if the wires entering *EXT-S* from above or below are treated like terminals. However, certain peculiarities can arise, such as a wire with an exit point at the right end of *EXT-S* having a dogleg at the left end of *EXT-S*. To avoid such peculiarities (without necessarily improving the perimeter), we consider an algorithm for an extended channel routing problem as follows. A net may have zero, one, or two terminals in S and two, one, or zero points of exit from *EXT-S* into channels *TR*, *TL*, *BR*, and *BL*. An exit point is specified as being on either the top or bottom of the channel and as being either to the left of the modules or the right of the modules. However, the column in which the exit point lies is not fixed otherwise. Nets in $S(r)$ with at least one exit point and a terminal or exit point on both the top and bottom of the channel are routed separately from other nets in $S(r)$ and excluded from the set to which PAIRUP is applied. To avoid making the perimeter larger, we need only make sure that each such net is routed using at most one extra exterior column and one track. But this can be done since one exit point for the net can be moved to a new exterior column, and the net routed with a single jog in a track lying within the region allowed for T_2 or T_4, depending on the (at most one) constraint remaining for the net. The modified version of SPLIT, which we call EXSPLIT, produces a routing that uses at most $d/2+n/2+4$ tracks and $d+c+c^L+c^R+6$ columns, where c is the number of columns in the modules, n is the number of nets, and d is the density relative to an initial assignment of exit points to c^L and c^R left and right exterior columns, respectively.

Discussion of PS. PS can be implemented to run in $O(n \log n)$ time, where n is the number of nets. In fact, $O(n \log n)$ time is needed only for sorting the input (the terminals) and the rest of the algorithm runs in linear time.

An example of a PS routing is given in Fig. 14. The use of doglegs for every net crossing the middle of the street is not very appealing. Unfortunately, the problem of

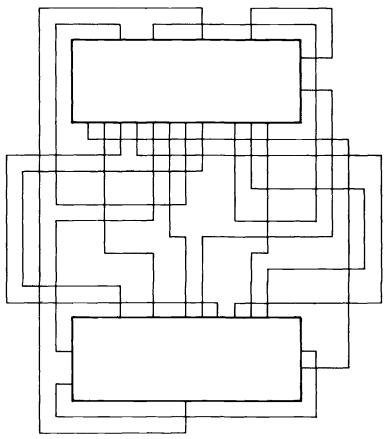


FIG. 14. *A* PS *routing.*

handling the general case of cycles and "chains" of constraints is difficult. By eye it is easily seen that several tracks and columns could be saved in this example. Two obvious improvements would be to move doglegs to the empty columns in the street and to let jogs share tracks if they do not overlap and are not involved in constraints. However, these improvements would not improve the worst case example given in the next section for PS.

4. The performance bounds for PS and SPLIT. In this section, we prove the performance bounds for the two-module routing algorithm PS and for the channel routing algorithm SPLIT.

For any routing R, let $|R|$ be the half-perimeter of R, i.e. the number of tracks and columns in the bounding box around R. In the case of channel routing, the modules may be ignored, and the bounding box drawn around the wires in the channel. For an instance P of a routing problem and an algorithm A, let $A(P)$ be the routing of P produced by A, and let $\mathrm{OPT}(P)$ be a perimeter-optimal routing of P. With these definitions, we can state the following theorem.

THEOREM 1. *For any instance P of the channel routing problem for two-terminal nets,*

$$|\mathrm{SPLIT}(P)| \leqq \tfrac{3}{2}|\mathrm{OPT}(P)| + 10,$$

using the half-perimeter measure. Moreover, the bound is tight in the sense that the multiplicative constant cannot be made smaller.

Proof. We assume the definitions given in the description of SPLIT. First, we prove the upper bound of $\frac{3}{2}$. The routing of Y requires three extra columns in addition to those of the modules. At most $|Y|$ tracks are needed for $Y-\{y\}$, and two tracks are needed for y. $S^R(r)-Y$ requires $|S^R(r)-Y|$ tracks. Thus, $S^R(r)$ requires a total of at most three extra columns and $|S^R(r)|+2$ tracks. Similarly, $S^L(r)$ requires at most $|S^L(r)+2|$ tracks. Note that tracks can be shared between $S^L(r)$ and $S^R(r)$, and

$$|S^L(r)| \leqq |S^R(r)| = \left\lceil \frac{|N|-|S(r)|}{2} \right\rceil.$$

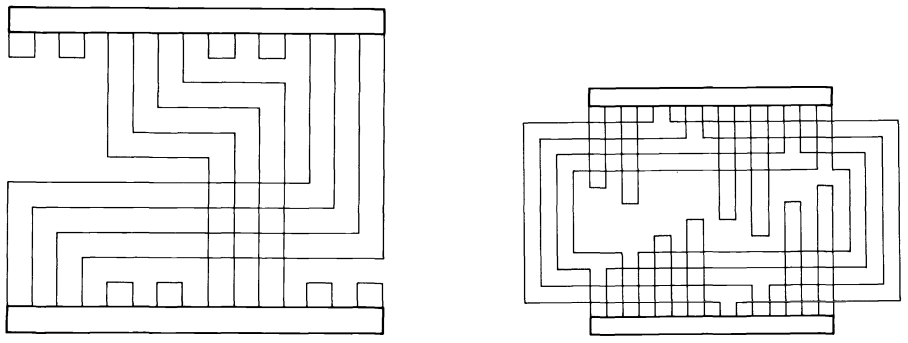
Adding in the (at most) $|S(r)|$ extra columns and $|S(r)|+1$ tracks used for $S(r)$, we see that SPLIT uses a total of at most $|S(r)|+6$ extra columns and

$$\left\lceil \frac{|N|-|S(r)|}{2} \right\rceil + |S(r)| + 3 \leqq \left\lceil \frac{|N|+|S(r)|}{2} \right\rceil + 3$$

tracks. Since the number of columns in the modules is c and the channel density is at least $|S(r)|$, an optimal routing uses at least $c+|S(r)|$ columns and tracks. Since $c \geqq N$, we have

$$\frac{|\mathrm{SPLIT}(P)|-10}{|\mathrm{OPT}(P)|} \leqq \frac{c+\frac{3}{2}|S(r)|+\frac{1}{2}|N|}{c+|S(r)|} \leqq \frac{\frac{3}{2}|N|+\frac{3}{2}|S(r)|}{|N|+|S(r)|} \leqq \frac{3}{2}.$$

For tightness, consider the following routing problem P, illustrated in Fig. 15. Let n be an even positive integer. For $1 \leqq i \leqq n$, there is a net with the top terminal in column $n+i$ and the bottom terminal in column $2n+i$ on L_B, and another net with the bottom terminal in column i and the top terminal in column $3n+i$. For $1 \leqq i \leqq n/2$, there are nets with terminals in columns $2i-1$ and $2i$ on L_T, columns $2n+2i-1$ and $2n+2i$ on L_T, columns $n+2i-1$ and $n+2i$ on L_B, and columns $3n+2i-1$ and $3n+2i$ on L_B. An optimal routing, depicted in Fig. 15a for $n=4$, requires $4n$ columns and $2n+2$ tracks. A SPLIT routing, such as the one in Fig. 15b, requires $6n$ columns and $3n$ tracks. Thus, for sufficiently large n, the ratio of $|\mathrm{SPLIT}(P)|$ and $|\mathrm{OPT}(P)|$ comes arbitrarily close to $\frac{3}{2}$. □



(a) (b)

FIG. 15. *A worst-case example for* SPLIT. (a) *An optimal routing.* (b) *A routing done by* SPLIT.

Now, we begin the proof of the main result of the paper: for any two-module routing problem P,

$$|\mathrm{PS}(P)| \leq \tfrac{19}{10}|\mathrm{OPT}(P)| + 24.$$

Since the proof is complicated, it will be helpful to develop some notation and prove some lemmas. The reader may like to (re)read the overview of the proof ideas given in § 2 before continuing.

Recall that the type (cross, straight, etc.) of a net is determined by the location of its terminals. The set of nets with one terminal on side U of module M_X and the other terminal on side V of module M_Y will be denoted by NAME (X, U, Y, V), where NAME is STRAIGHT, CROSS, CORN (corner), or OPP (opposite), according to the type of net that has terminals in these positions. For example, STRAIGHT (T, L, B, L) is the set of straight nets with one terminal on the left side of M_T and the other on the left side of M_B, and OPP (T, B, B, B) is the set of opposite nets with one terminal on the bottom of M_T and the other on the bottom of M_B. Note that NAME is actually redundant since it can be deduced from the location of the terminals; it is given for readability. Also, note that for any NAME, NAME $(X, U, Y, V) =$ NAME (Y, V, X, U).

For $s \in \{L, R, T, B\}$, define $\bar{s}$ to be the opposite of s, e.g., L instead of R or T instead of B.

Define the interval *covered* by a wire in channel X to be the horizontal interval between its leftmost and rightmost points in X if X is a "horizontal" channel (T, B, S, or the extensions of these channels), or the vertical interval between its topmost and bottommost points in X if X is a "vertical" channel (TL, TR, BL, BR, L, or R).

Given the PS routing, define the *natural interval* of a net within a channel to be the interval between its terminals if both terminals lie in the channel, and the interval covered by the net otherwise. In PS (P) street nets cover at least their natural intervals in channel S; other nets cover exactly their natural intervals.

In OPT (P), a net may have one of a number of possible routings. For example, for a net in CORN (T, T, B, R), the wire may run from the terminal on top of M_T right and down, or it may run left, down, between the modules, and down, or it may run left, down, below B, and up. Recall that PS routes such a wire right and down. (In the above descriptions, we ignore the initial and final segments of the wire that connect it to the terminals.) In order to deal with the possible routings in OPT (P), we compare the intervals covered by wires in OPT (P) with their natural intervals according to the PS channel assignment. For any set N of nets and any channel X, define $\mathrm{SAME}_X(N)$

to be the set of nets n in N such that the natural interval of n in channel X is a subset of the interval covered by n's wire in channel X in OPT(P). Define $\text{DIFF}_X(N) = N - \text{SAME}_X(N)$.

A net is *local* to channel X if it is a corner, straight, or cross net and in PS(P) some part of its wire lies in channel X. The only nets that are never local nets are opposite nets. The set of nets local to channel X is denoted by LOCAL(X). For example, LOCAL(TR) is the union of the following sets:

$$\text{CROSS}(B, L, T, R), \text{CORN}(T, T, B, R), \text{STRAIGHT}(T, R, Y, R),$$

$$\text{CORN}(T, R, Y, Y), \text{CORN}(T, R, \bar{Y}, Y), \quad \text{where } Y \in \{T, B\}.$$

Note that it is possible for a net local to a channel not to have a terminal in the channel, as in the case of CORN(T, T, B, R) in this example.

Recall that for any routing, the width of a channel must be wide enough to accommodate the maximum number of overlapping intervals of wires passing through the channel. (This number is at least the channel density.) For any channel X, define LAPSAME_X to be the maximum number of overlapping intervals of the wires in $\text{SAME}_X(\text{LOCAL}(X))$ in OPT(P). This quantity is of interest since the width of channel X in both OPT(P) and PS(P) must be at least LAPSAME_X.

For any channel X in a routing $A(P)$, where A is PS or OPT, define TRACKS(A, X) to be the number of nonempty tracks in this channel in $A(P)$ and COL(A, X) to be the number of nonempty columns in this channel in $A(P)$.

Lower bounds on $|\text{OPT}(P)|$. The heart of the proof lies in deriving lower bounds on $|\text{OPT}(P)|$ in a form that allows an easy comparison with PS(P). We use two different techniques in deriving these lower bounds. One involves simply counting how many nets must use different tracks or columns within single channels, and the other involves assigning a weight to each net to reflect how many nets of various types can share certain tracks or columns.

LEMMA 1. *The following is a lower bound on the number of tracks in the horizontal channels of* OPT(P):

$$\sum_{X \in \{T,B,S\}} |\text{TRACKS}(\text{OPT}, X)| \geqq \sum_{X \in \{T,B,S\}} \text{LAPSAME}_X + \sum_{X \in \{T,B,S\}} |\text{DIFF}_X(\text{LOCAL}(X))| + \sum_{X \in \{T,B\}} |\text{OPP}(X, L, X, R)|.$$

Proof. Each net in $\text{DIFF}_X(\text{LOCAL}(X))$, $X \in \{T, B, S\}$, or $\text{OPP}(X, L, X, R)$, $X \in \{T, B\}$, covers all of channel T, B or S in OPT(P). For nets in $\text{OPP}(X, L, X, R)$, this follows from the location of the terminals. In the case of the DIFF expressions, a wire must run the long way around a module to connect the terminals if it cannot cover the wire's natural interval in channel X.

In addition, LAPSAME_X tracks are needed in channel X, for $X \in \{T, B, S\}$. □

LEMMA 2. *The following are two lower bounds on the number of columns in the vertical channels of* OPT(P):

(a) $$\sum_{X \in \{L,R\}} |\text{COL}(\text{OPT}, X)| \geqq \tfrac{1}{2} \sum_{X \in \{TL, TR, BL, BR\}} \text{LAPSAME}_X + |\text{OPP}(T, T, B, B)| + \tfrac{1}{2} \sum_{X \in \{T,B\}} |\text{OPP}(X, T, X, B)| + \tfrac{1}{2} \sum_{X \in \{T,B\}} |\text{OPP}(T, X, B, X)|$$

(b) $$\sum_{X\in\{L,R\}} |\mathrm{COL}(\mathrm{OPT}, X)|$$

$$\geqq \sum_{X\in\{TL,TR,BL,BR\}} \mathrm{LAPSAME}_X + \sum_{X\in\{TL,TR,BL,BR\}} |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))|$$

$$+|\mathrm{OPP}(T, T, B, B)|+|\mathrm{OPP}(T, T, T, B)|+|\mathrm{OPP}(T, T, B, T)|.$$

Proof. (a) For $Y\in\{T, B\}$ and $X\in\{L, R\}$, define z_{YX} to be the number of nets in $\mathrm{OPP}(Y, T, Y, B)\cup\mathrm{OPP}(T, Y, B, Y)\cup\mathrm{OPP}(T, T, B, B)$ that cover the entire channel YX in $\mathrm{OPT}(P)$. Since

$$\mathrm{COL}(\mathrm{OPT}, R)\geqq \max\{z_{BR}+\mathrm{LAPSAME}_{BR}, z_{TR}+\mathrm{LAPSAME}_{TR}\},$$

we have

$$|\mathrm{COL}(\mathrm{OPT}, R)|\geqq \tfrac{1}{2}\mathrm{LAPSAME}_{TR}+\tfrac{1}{2}\mathrm{LAPSAME}_{BR}+\tfrac{1}{2}z_{TR}+\tfrac{1}{2}z_{BR}.$$

Similarly,

$$|\mathrm{COL}(\mathrm{OPT}, L)|\geqq \tfrac{1}{2}\mathrm{LAPSAME}_{TL}+\tfrac{1}{2}\mathrm{LAPSAME}_{BL}+\tfrac{1}{2}z_{TL}+\tfrac{1}{2}z_{BL}.$$

Now,

$$\sum_{X\in\{TL,TR,BL,BR\}} z_X \geqq 2|\mathrm{OPP}(T, T, B, B)|+\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, T, X, B)|$$

$$+\sum_{X\in\{T,B\}} |\mathrm{OPP}(T, X, B, X)|$$

since every such opposite net must go on one side or the other of one or both modules. Combining these facts gives formula (a).

(b) We count the number of nets in $\mathrm{OPT}(P)$ that use the entire channel TL or TR plus the number of nets using overlapping intervals in these channels. Every opposite net with a terminal on the top of M_T uses either the entire channel TL or the entire channel TR. Consider a net n in $\mathrm{DIFF}(\mathrm{LOCAL}(X))$, $X\in\{TL, TR\}$. Because n is in $\mathrm{LOCAL}(X)$, n must be a corner net, straight net, or cross net. Because of the direction assigned to each such net in $\mathrm{PS}(P)$, its presence in $\mathrm{DIFF}(\mathrm{LOCAL}(X))$ means that it must be routed around the module and through the entire upper channel on the far side. For example, if n is a corner net, with one terminal in X and one terminal on M_B, it is routed down through X in $\mathrm{PS}(P)$, and therefore in $\mathrm{OPT}(P)$ it is routed up, over the top, and down the other side. Recall that $\mathrm{LAPSAME}_X$ is the maximum number of overlapping intervals in $\mathrm{SAME}_X(\mathrm{LOCAL}(X))$. Therefore, $\mathrm{LAPSAME}_X$ does not count any of the nets discussed above, and the intervals contributing to $\mathrm{LAPSAME}_X$ occupy columns distinct from those used for the nets above. □

For the next lower bound, we assign a weight to each net in $\mathrm{OPT}(P)$ such that $\mathrm{OPT}(P)$ is at least the total weight of all the nets. The weight of a net will be the sum of two weights assigned to its wire and a weight assigned to its terminals.

First, define the weighting functions on wires as follows. A wire accumulates weight for crossing certain horizontal and vertical lines (see Fig. 6). It accumulates a weight of $\frac{1}{2}$ for each crossing of the line $x=\mathrm{LEFT}$ or $x=\mathrm{RIGHT}$ above $\mathrm{TOP}(T)$ or below $\mathrm{BOTTOM}(B)$. For any net n, $W_V(n)$ is the total weight accumulated by its wire in $\mathrm{OPT}(P)$ by crossing the above vertical lines. A wire also accumulates a weight of $\frac{1}{2}$ for each crossing of the line $y=\mathrm{BOTTOM}(T)$ or $y=\mathrm{TOP}(B)$ to the left of LEFT or to the right of RIGHT. For any net n, $W_H(n)$ is the total weight accumulated by its wire in $\mathrm{OPT}(P)$ by crossing the above horizontal lines. Let $\bar{W}_H$ and $\bar{W}_V$ be the sum of $W_H(n)$ and $W_V(n)$, respectively, over all nets n. It is easy to see that the number

of tracks in channels T and B plus the number of columns in channels L and R is at least $\bar{W}_H$ plus $\bar{W}_V$.

Define a weighting function on terminals as follows. A terminal has a weight of $\frac{1}{2}$ if it lies on the sides of the street or on the left or right side of either module. Terminals on the top of M_T or on the bottom of M_B have weight 0. Let $W_T(n)$ be the sum of the weights of the terminals of net n. Let $\bar{W}_T$ be the sum of the weights of all terminals. It is easy to see that the number of columns between LEFT and RIGHT plus the number of tracks in M_T and M_B is at least $\bar{W}_T$.

PROPOSITION. $|\text{OPT}(P)| \geqq \bar{W}_H + \bar{W}_V + \bar{W}_T + |\text{TRACKS (OPT, } S)|$.

The real key to the proof of the main theorem is in calculating a lower bound on the weight of each net. For each net n, define $W(n)$ to be the sum of $W_H(n)$, $W_V(n)$, and $W_T(n)$. Tables 1-4 give minimum values of $W(n)$ for various classes of nets n.

The values in the table are calculated by considering all possible ways of routing nets in OPT (P). In general, different weights are calculated for a net in LOCAL (X), where X is a channel, according to whether its wire is in SAME_X (LOCAL (X)) or DIFF_X (LOCAL (X)); which set the wire is in is represented in the table by an "s" for SAME and a "d" for "DIFF". For example, consider a straight net n with both terminals in channel TR. No matter how n's wire is routed in OPT (P), $W_T(n) = 1$. If n's wire in OPT (P) is in SAME_{TR} (LOCAL (TR)), then $W_H(n)$ and $W_V(n)$ may both be zero; thanks to $W_T(n)$, the total weight is at least 1. However, if n's wire in OPT (P) is in DIFF_{TR} (LOCAL (TR)), then no matter how the wire is routed, it must go nearly all the way around at least one component, forcing it to have $W_V(n) \geqq 1$ and $W_T(n) \geqq 1$; adding in $W_T(n)$, the total weight is at least 3.

From the tables of lower bounds given above for the weights of various types of nets, we obtain the following lower bound on $|\text{OPT}(P)|$. The lower bound looks peculiar, since the weights assigned to various classes of nets include assorted fractions.

TABLE 1

Minimum weights for straight nets, where "s" and "d" stand for "same" and "diff", respectively.

(a) *Straight nets with both terminals in channel S, T, B, XL, or XR, where* $X \in \{T, B\}$.

Restrictions on Z	SAME/DIFF in channel Z	Weight			
		W_T	W_V	W_H	W
T/B	s	0	0	0	0
T/B	d	0	1	1	2
$XL/XR/S$	s	1	0	0	1
$XL/XR/S$	d	1	1	1	3

(b) *Straight nets with one terminal in channel TX and the other in channel BX, where* $X \in \{L, R\}$.

SAME/DIFF in channels		Weight			
TX	BX	W_T	W_V	W_H	W
s	s	1	0	1	2
s	d	1	1	1	3
d	d	1	2	1	4

TABLE 2
Minimum weights for corner nets.

(a) *Corner nets with one terminal in a horizontal channel X, $X \in \{T, B, S\}$, and the other in an adjacent vertical channel ZY, $Y \in \{L, R\}$, $Z \in \{T, B\}$, where $Z = X$, if $X \in \{T, B\}$.*

Restrictions on X	SAME/DIFF in channels		Weight			
	X	ZY	W_T	W_V	W_H	W
S	s	s	1	0	.5	1.5
S	s/d	d	1	1	.5	2.5
S	d	s	1	1	.5	2.5
T/B	s	s	.5	.5	0	1
T/B	s/d	d	.5	.5	1	2
T/B	d	s	.5	.5	1	2

(b) *Corner nets with one terminal in a horizontal channel X, $X \in \{T, B\}$, and the other in a nonadjacent vertical channel $\bar{X}Y$, $Y \in \{L, R\}$. Note that in* PS (*P*), *the wire also passes through channel XY.*

SAME/DIFF in channels			Weight			
X	XY	XY	W_T	W_V	W_H	W
s/d	s/d	s	.5	.5	1	2
s/d	s/d	d	.5	1.5	1	3

TABLE 3
Minimum weights for cross nets, with one terminal in channel XL, and the other in channel XR, $X \in \{T, B\}$. Note that in PS (*P*), *a cross net wire passes through S in addition to the channels containing the terminals.*

SAME/DIFF in channels			Weight			
S	XL	XR	W_T	W_V	W_H	W
s	s	s	1	0	1	2
s/d	s	s/d	1	1	1	3
s	d	d	1	2	1	4
d	d	d	—	—	—	—

TABLE 4
Minimum weights for opposite nets. Terminals are in the indicated channels, where $X \in \{T, B\}$. For XL, XR, the parentheses indicate two possible assignments of weights, depending on whether the wire passes through S.

Channels	Weight			
	W_T	W_V	W_H	W
XL, XR	1	1(0)	0(1)	2
$S, T/B$	.5	.5	.5	1.5
T, B	0	1	1	2

In fact, there is a good deal of flexibility in what fractions are assigned to various classes to make the total come out right. The particular fractions used in the lemma are chosen because they are easily compared to the upper bounds on $|\mathrm{PS}(P)|$ obtained later on.

LEMMA 3.

$$
\begin{aligned}
|\mathrm{OPT}(P)| \geqq & \tfrac{5}{9} \sum_{X \in \{S,TR,TL,BR,BL\}} |\mathrm{SAME}_X(\mathrm{LOCAL}(X))| + \tfrac{5}{9}\mathrm{LAPSAME}_S \\
& + \tfrac{10}{9} \sum_{X \in \{S,TR,TL,BR,BL\}} |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))| \\
& + 2 \sum_{X \in \{T,B\}} |\mathrm{OPP}(X,L,X,R)| + 2|\mathrm{OPP}(T,T,B,B)| \\
& + \tfrac{35}{18} \sum_{X \in \{T,B\}} |\mathrm{OPP}(X,T,X,B)| + \tfrac{35}{18} \sum_{X \in \{T,B\}} |\mathrm{OPP}(T,X,B,X)|.
\end{aligned}
$$

Proof. Much of the following analysis deals with nets local to S. For convenience, we define SCORNERS to be the set of corner nets with one terminal on a side of the street, and SNETS to be the set of straight nets with both terminals in channel S.

By the proposition, we can derive a lower bound on $|\mathrm{OPT}(P)|$ from the weights of the nets plus the number of tracks in S.

At least

$$
\begin{aligned}
t = & \tfrac{1}{2} \sum_{X \in \{T,B\}} |\mathrm{OPP}(X,T,X,B)| + \tfrac{1}{2} \sum_{X \in \{T,B\}} |\mathrm{OPP}(T,X,B,X)| \\
& + \tfrac{1}{2}|\mathrm{SAME}_S(\mathrm{SCORNERS})| + \tfrac{1}{2} \sum_{X \in \{T,B\}} |\mathrm{SAME}_S(\mathrm{CROSS}(X,L,\bar{X},R))|
\end{aligned}
$$

tracks are needed in S by $\mathrm{OPT}(P)$ to get these wires into S from the sides. Also, at least $\mathrm{LAPSAME}_S$ tracks are needed in the street for the nets local to the street. Therefore, S has at least $\max(t, \mathrm{LAPSAME}_S)$ tracks in $\mathrm{OPT}(P)$.

Now, we examine the weights of the various nets. Cross nets, straight nets, and corner nets may be local to more than one channel, as indicated in Tables 1–3. By distributing the minimum weights for these nets across the expressions for various channels in the following formula, as described below, we obtain the following lower bound on the total weight of nets in $\mathrm{OPT}(P)$.

$$
\begin{aligned}
\bar{W}_H + \bar{W}_V + \bar{W}_T \geqq & \tfrac{5}{9} \sum_{X \in \{TR,TL,BR,BL\}} |\mathrm{SAME}_X(\mathrm{LOCAL}(X))| \\
& + \tfrac{10}{9} \sum_{X \in \{TR,TL,BR,BL\}} |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))| \\
& + \tfrac{1}{2}|\mathrm{SAME}_S(\mathrm{LOCAL}(S))| \\
& + \tfrac{10}{9}|\mathrm{DIFF}_S(\mathrm{LOCAL}(S))| + \tfrac{1}{2}|\mathrm{SAME}_S(\mathrm{SNETS})| \\
& + \tfrac{1}{18}|\mathrm{SCORNERS}| + \tfrac{1}{18} \sum_{X \in \{T,B\}} |\mathrm{CROSS}(X,L,\bar{X},R)| \\
& + 2 \sum_{X \in \{T,B\}} |\mathrm{OPP}(X,L,X,R)| + 2|\mathrm{OPP}(T,T,B,B)| \\
& + \tfrac{3}{2} \sum_{X \in \{T,B\}} |\mathrm{OPP}(X,T,X,B) \cup \mathrm{OPP}(T,X,B,X)|.
\end{aligned}
$$

To see how to prove this formula correct, consider a net n in SCORNERS, with terminals in channels S and BR. This net is in either $\mathrm{SAME}_S(\mathrm{LOCAL}(S))$ or $\mathrm{DIFF}_S(\mathrm{LOCAL}(S))$ and in either $\mathrm{SAME}_{BR}(\mathrm{LOCAL}(BR))$ or $\mathrm{DIFF}_{BR}(\mathrm{LOCAL}$

(BR)). Suppose it is in $\text{SAME}_S(\text{LOCAL}(S))$ and $\text{SAME}_{BR}(\text{LOCAL}(BR))$. In the formula, n accounts for a weight of $\frac{1}{2}+\frac{5}{9}+\frac{1}{18}$, these being the weights applied to $\text{SAME}_S(\text{LOCAL}(S))$, $\text{SAME}B_R(\text{LOCAL}(BR))$, and SCORNERS, respectively. The total, $\frac{10}{9}$ is less than the minimum weight of 1.5 for n obtained from Table 2. Similarly, if n is in $\text{DIFF}_S(\text{LOCAL}(S))$ and $\text{DIFF}_{BR}(\text{LOCAL}(BR))$, it accounts for a weight of $\frac{10}{9}+\frac{10}{9}+\frac{1}{18}$, which is less than n's minimum weight of 2.5 obtained from Table 2. It is straightforward but tedious to verify for each net that no matter how it is routed in $\text{OPT}(P)$, the weight that it accounts for in the above lower bound is at most the weight given in the table for that routing.

Combining the above formulas, using the definition of t, and using the fact that $\text{LOCAL}(S)$ consists of cross nets and nets in SNETS and SCORNERS, we get the following lower bound on $|\text{OPT}(P)|$.

$$\begin{aligned}|\text{OPT}(P)| \geqq & \tfrac{5}{9} \sum_{X\in\{TR,TL,BR,BL\}} |\text{SAME}_X(\text{LOCAL}(X))| \\ & + \tfrac{10}{9} \sum_{X\in\{TR,TL,BR,BL\}} |\text{DIFF}_X(\text{LOCAL}(X))| \\ & + 2 \sum_{Y\in\{T,B|} |\text{OPP}(Y, L, Y, R)| + 2|\text{OPP}(T, T, B, B)| \\ & + \tfrac{10}{9}|\text{DIFF}_S(\text{LOCAL}(S))| - \tfrac{35}{18}|\text{SAME}_S(\text{SCORNERS})| \\ & - \tfrac{35}{18} \sum_{X\in\{T,B\}} |\text{SAME}_S(\text{CROSS}(X, L, \bar{X}, R))| \\ & + |\text{SAME}_S(\text{LOCAL}(S))| + 3t + \max\{\text{LAPSAME}_S, t\}.\end{aligned}$$

Let $x = \text{LAPSAME}_S$ and $n = |\text{SAME}_S(\text{LOCAL}(S))|$. Note that $x \leqq n$. We claim that

$$\tfrac{9}{10}[n+3t+\max\{x, t\}] \geqq \tfrac{1}{2}n+\tfrac{1}{2}x+\tfrac{7}{2}t.$$

For if $t \leqq x$ then

$$\begin{aligned}\tfrac{1}{2}n+\tfrac{1}{2}x+\tfrac{7}{2}t &\leqq \tfrac{1}{2}n+\tfrac{1}{2}x+\tfrac{2}{5}(x+n)+(\tfrac{7}{2}-\tfrac{4}{5})t \\ &= \tfrac{9}{10}(n+x+3t) \\ &= \tfrac{9}{10}[n+3t+\max\{x, t\}].\end{aligned}$$

On the other hand, if $x < t$ then

$$\begin{aligned}\tfrac{1}{2}n+\tfrac{1}{2}x+\tfrac{7}{2}t &\leqq \tfrac{1}{2}n+(\tfrac{2}{5}n+\tfrac{1}{10}t)+\tfrac{7}{2}t \\ &= \tfrac{9}{10}(n+4t) \\ &= \tfrac{9}{10}[n+3t+\max\{x, t\}].\end{aligned}$$

It is interesting to note that it is only the $\frac{9}{10}$ in the above formula that ultimately leads to the exact upper bound of $\frac{19}{10}$ in the main theorem.

Using our claim, we can substitute $\frac{10}{9}(\frac{1}{2}n+\frac{1}{2}x+\frac{7}{2}t)$ for $n+3t+\max\{x, t\}$ in the above lower bound to obtain

$$\begin{aligned}|\text{OPT}(P)| \geqq & \tfrac{5}{9} \sum_{X\in\{TR,TL,BR,BL\}} |\text{SAME}_X(\text{LOCAL}(X))| \\ & + \tfrac{10}{9} \sum_{X\in\{TR,TL,BR,BL\}} |\text{DIFF}_X(\text{LOCAL}(X))| \\ & + 2 \sum_{Y\in\{T,B|} |\text{OPP}(Y, L, Y, R)| + 2|\text{OPP}(T, T, B, B)|\end{aligned}$$

$$+\tfrac{10}{9}|\mathrm{DIFF}_S(\mathrm{LOCAL}(S))| - \tfrac{35}{18}|\mathrm{SAME}_S(\mathrm{SCORNERS})|$$
$$-\tfrac{35}{18}\sum_{X\in\{T,B\}}|\mathrm{SAME}_S(\mathrm{CROSS}(X, L, \bar{X}, R))|$$
$$+\tfrac{5}{9}|\mathrm{SAME}_S(\mathrm{LOCAL}(S))| + \tfrac{35}{9}t + \tfrac{5}{9}\mathrm{LAPSAME}_S.$$

Substituting the definition of t for t in the last formula gives the lower bound of the lemma. □

Analysis of PS. It remains to analyze how many tracks and columns can be used by PS.

LEMMA 4.

$$\sum_{X\in\{T,B\}}\mathrm{TRACKS}(\mathrm{PS}, X) \leqq \tfrac{1}{2}\mathrm{LAPSAME}_S + \sum_{X\in\{T,B\}}\mathrm{LAPSAME}_X$$
$$+\tfrac{1}{2}|\mathrm{SAME}_S(\mathrm{LOCAL}(S))|$$
$$+\sum_{X\in\{S,T,B\}}|\mathrm{DIFF}_X\ \mathrm{LOCAL}(X))|$$
$$+|\mathrm{OPP}(T, T, B, B)| + \sum_{X\in\{T,B\}}|\mathrm{OPP}(X, L, X, R)|$$
$$+\tfrac{5}{4}\sum_{X\in\{T,B\}}(|\mathrm{OPP}(X, T, X, B)| + |\mathrm{OPP}(T, X, B, X)|) + 11.$$

Proof. The wires in channels T, B, and S in PS(P) belong either to local nets or to opposite nets. For $X \in \{T, B\}$, the maximum overlap of the nets in LOCAL(X) is at most

$$\mathrm{LAPSAME}_X + |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))|.$$

When PAIRUP is applied to m nets, the side defined as "top" uses $\lceil m/2 \rceil$ tracks while the "bottom" uses $\lceil (m+1)/2 \rceil$ tracks; the "left" uses $\lceil m/2 \rceil$ columns and the "right" uses $\lfloor m/2 \rfloor$ columns. For $X \in \{T, B\}$, recall that PAIRUP is applied to OPP(X, T, X, B), OPP(T, X, B, X), OPP(T, T, B, B), and OPP(X, L, X, R) such that the "top" is the street, and to OPP(X, L, X, R) such that the "right" side is the street. Therefore, a maximum of

$$\tfrac{1}{2}|\mathrm{OPP}(X, T, X, B)| + \tfrac{1}{2}|\mathrm{OPP}(T, X, B, X)|$$
$$+\tfrac{1}{2}|\mathrm{OPP}(T, T, B, B)| + \lceil \tfrac{1}{2}|\mathrm{OPP}(X, L, X, R)| \rceil + 3$$

of these nets may overlap in channel X, $X \in \{T, B\}$. Thus, for $X \in \{T, B\}$,

$$\mathrm{TRACKS}(\mathrm{PS}, X) \leqq \mathrm{LAPSAME}_X + |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))|$$
$$+\tfrac{1}{2}|\mathrm{OPP}(X, T, X, B)| + \tfrac{1}{2}|\mathrm{OPP}(T, X, B, X)|$$
$$+\tfrac{1}{2}|\mathrm{OPP}(T, T, B, B)| + \lceil \tfrac{1}{2}|\mathrm{OPP}(X, L, X, R)| \rceil + 3.$$

Now, we calculate the number of tracks used in the street in PS(P). Recall that for a street with m nets and density d, EXSPLIT uses at most $m/2 + d/2 + 4$ tracks. In this case, d represents the channel density in PS(P) before applying EXSPLIT. We have

$$d \leqq \sum_{X\in\{T,B\}}(\tfrac{1}{2}|\mathrm{OPP}(X, T, X, B)| + \tfrac{1}{2}|\mathrm{OPP}(T, X, B, X)|)$$
$$+\sum_{X\in\{T,B\}}\lfloor \tfrac{1}{2}|\mathrm{OPP}(X, L, X, R)| \rfloor + \mathrm{LAPSAME}_S$$
$$+|\mathrm{DIFF}_S(\mathrm{LOCAL}(S))| + 2.$$

Recall that nets routed from channel TR to BR or from channel TL to BL go straight across the extended street in PS (P) and are not routed by EXSPLIT. Hence, the number m of nets routed by EXSPLIT is at most

$$|\text{LOCAL}(S)| + \sum_{X\in\{T,B\}} (\lfloor \tfrac{1}{2}|\text{OPP}(X, L, X, R)|\rfloor + |\text{OPP}(X, T, X, B)| + |\text{OPP}(T, X, B, X)|).$$

From these facts, we obtain the following bound on the number of tracks in S.

$$\begin{aligned}\text{TRACKS}(\text{PS}, S) \leqq\ & \tfrac{1}{2}\text{LAPSAME}_S + \tfrac{1}{2}|\text{SAME}_S(\text{LOCAL}(S))| \\ & + |\text{DIFF}_S(\text{LOCAL}(S))| + \sum_{X\in\{T,B\}} \lfloor \tfrac{1}{2}|\text{OPP}(X, L, X, R)|\rfloor \\ & + \tfrac{3}{4}\sum_{X\in\{T,B\}} (|\text{OPP}(X, T, X, B)| + |\text{OPP}(T, X, B, X)|) + 5.\end{aligned}$$

Combining the above formulas yields the lemma. □

THEOREM 2. *For any routing problem P,*

$$|\text{PS}(P)| \leqq \tfrac{19}{10}|\text{OPT}(P)| + 24.$$

Moreover, this bound is tight in the sense that the multiplicative factor cannot be made smaller no matter what the value is for the additive constant.

Proof. Note that

$$\begin{aligned}|\text{PS}(P)| - |\text{OPT}(P)| = & \sum_{X\in\{T,B,S\}} \text{TRACKS}(\text{PS}, X) - \sum_{X\in\{T,B,S\}} \text{TRACKS}(\text{OPT}, X) \\ & + \sum_{X\in\{L,R\}} \text{COL}(\text{PS}, X) - \sum_{X\in\{L,R\}} \text{COL}(\text{OPT}, X)\end{aligned}$$

since in addition to the tracks and columns appearing in this formula there are exactly RIGHT − LEFT columns and TOP (T) − BOTTOM (T) + TOP (B) − BOTTOM (B) tracks in both PS (P) and OPT (P). We will calculate upper bounds on

$$\sum_{X\in\{T,B,S\}} \text{TRACKS}(\text{PS}, X) - \sum_{X\in\{T,B,S\}} \text{TRACKS}(\text{OPT}, X)$$

and

$$\sum_{X\in\{L,R\}} \text{COL}(\text{PS}, X) - \sum_{X\in\{L,R\}} \text{COL}(\text{OPT}, X)$$

and show that their sum is at most $\frac{9}{10}|\text{OPT}(P)| + 24$. The theorem follows.

Combining Lemma 4 and Lemma 1, we have

$$\begin{aligned}& \sum_{X\in\{T,S,B\}} \text{TRACKS}(\text{PS}, X) - \sum_{X\in\{T,S,B\}} \text{TRACKS}(\text{OPT}, X) \\ & \leqq \tfrac{1}{2}|\text{SAME}_S(\text{LOCAL}(S)| \\ & \quad + \tfrac{5}{4}\sum_{X\in\{T,B\}} (|\text{OPT}(X, T, X, B)| + |\text{OPT}(T, X, B, X)|) \\ & + |\text{OPT}(T, T, B, B)| - \tfrac{1}{2}\text{LAPSAME}_S + 11.\end{aligned} \tag{1}$$

Obtaining the desired bound on columns is more complicated. Recall that EXSPLIT uses at most $d+6$ columns in addition to the columns initially containing exit points for nets. For $X \in \{T, B\}$ and $T \in \{L, R\}$, let e_{XY} be the number of nonempty columns in channel XY. Since interval coloring is applied to channels TL and BL and to TR and BR to decide the initial assignment of exit points to columns in step (4), the number of columns initially containing exit points is at most $\max\{e_{TX}, e_{BX}\}$, for

$X \in \{L, R\}$. Using the upper bound on d from the proof of Lemma 4, we have

$$\sum_{X\in\{L,R\}} \mathrm{COL}(\mathrm{PS}, X) \leqq \sum_{X\in\{L,R\}} \max\{e_{TX}, e_{BX}\} + \mathrm{LAPSAME}_S + |\mathrm{DIFF}_s(\mathrm{LOCAL}(S))|$$

$$+\tfrac{1}{2}\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, T, X, B)| + \tfrac{1}{2}\sum_{X\in\{T,B\}} |\mathrm{OPP}(T, X, B, X)| \tag{2}$$

$$+\tfrac{1}{2}\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, L, X, R)| + 8.$$

Note that for $X \in \{TR, TL, BR, BL\}$,

$$e_x \leqq \mathrm{LAPSAME}_X + |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))|$$

$$+ \lceil \tfrac{1}{2}(|\mathrm{OPP}(X, L, X, R)| + 1) \rceil + \lceil \tfrac{1}{2}|\mathrm{OPP}(T, T, B, B)| \rceil \tag{3}$$

$$+ \lceil \tfrac{1}{2}|\mathrm{OPP}(X, T, X, B)| \rceil + \lceil \tfrac{1}{2}|\mathrm{OPP}(T, X, B, X)| \rceil.$$

We consider two cases according to whether or not the maximum of e_{TL} and e_{BL} and the maximum of e_{TR} and e_{BR} are achieved in the bottom channel on one side and the top on the other or both in the top (bottom) channels.)

Case 1. The maxima of e_{TX} and e_{BX} are achieved in the bottom channel on one side and the top in the other.

Without loss of generality, we may assume that the maxima occur in the top right and bottom left channels. Substituting the bound on e_X, $X \in \{TR, BL\}$, of formula (3) into formula (2), we have

$$\sum_{X\in\{L,R\}} \mathrm{COL}(\mathrm{PS}, X) \leqq \sum_{X\in\{TR,BL\}} \mathrm{LAPSAME}_X + \sum_{X\in\{TR,BL\}} |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))|$$

$$+ \sum_{X\in\{T,B\}} |\mathrm{OPP}(X, L, X, R)| + |\mathrm{OPP}(T, T, B, B)|$$

$$+ \sum_{X\in\{T,B\}} |\mathrm{OPP}(T, X, B, X)| + \sum_{X\in\{T,B\}} |\mathrm{OPP}(X, T, X, B)|$$

$$+ \mathrm{LAPSAME}_S + |\mathrm{DIFF}_S(\mathrm{LOCAL}(S))| + 13.$$

Using Lemma 2a, we obtain

$$\sum_{X\in\{L,R\}} \mathrm{COL}(\mathrm{PS}, X) - \sum_{X\in\{L,R\}} \mathrm{COL}(\mathrm{OPT}, X)$$

$$\leqq \tfrac{1}{2}\sum_{X\in\{L,R\}} \mathrm{LAPSAME}_X + \sum_{X\in\{TR,BL\}} |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))|$$

$$+ \sum_{X\in\{T,B\}} |\mathrm{OPP}(X, L, X, R)| + \sum_{X\in\{T,B\}} \tfrac{1}{2}|\mathrm{OPP}(T, X, B, X)| \tag{4}$$

$$+ \sum_{X\in\{T,B\}} \tfrac{1}{2}|\mathrm{OPP}(X, T, X, B)| + \mathrm{LAPSAME}_S$$

$$+ |\mathrm{DIFF}_S(\mathrm{LOCAL}(S))| + 13.$$

Combining inequalities (1) and (4), we have

$$|\mathrm{PS}(P)| - |\mathrm{OPT}(P)| \leqq \tfrac{1}{2}\sum_{X\in\{TR,BL\}} \mathrm{LAPSAME}_X + \sum_{X\in\{TR,BL\}} |\mathrm{DIFF}_X(\mathrm{LOCAL}(X))|$$

$$+ \sum_{X\in\{T,B\}} |\mathrm{OPP}(X, L, X, R)|$$

$$+ |\mathrm{OPP}(T, T, B, B)| + |\mathrm{DIFF}_S(\mathrm{LOCAL}(S))|$$

$$+ \tfrac{7}{4}\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, T, X, B) \cup \mathrm{OPP}(T, X, B, X)|$$

$$+ \tfrac{1}{2}|\mathrm{SAME}_S(\mathrm{LOCAL}(S))| + \tfrac{1}{2}\mathrm{LAPSAME}_S + 24.$$

Comparing this formula with Lemma 3, we see that

$$|\mathrm{PS}(P)| - |\mathrm{OPT}(P)| \leq \tfrac{9}{10}|\mathrm{OPT}(P)| + 24.$$

Case 2. The maxima of e_{TX} and e_{BX} are both in the top side channels or both in the bottom side channels.

Without loss of generality, we may assume that the maxima are in the top channels on both sides. Using formulas (2) and (3), we have

$$\begin{aligned}\sum_{X\in\{L,R\}} \mathrm{COL}(\mathrm{PS}, X) \leqq & \sum_{X\in\{TR,TL\}} \mathrm{LAPSAME}_X + \sum_{X\in\{TR,TL\}} \mathrm{DIFF}_X(\mathrm{LOCAL}(X)) \\ & + \tfrac{3}{2}\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, L, X, R)| + |\mathrm{OPP}(T, T, B, B)| \\ & + \tfrac{3}{2}|\mathrm{OPP}(T, T, T, B)| + \tfrac{3}{2}|\mathrm{OPP}(T, T, B, T)| \\ & + \tfrac{1}{2}|\mathrm{OPP}(T, B, B, B)| + \tfrac{1}{2}|\mathrm{OPP}(B, T, B, B)| \\ & + \mathrm{LAPSAME}_S + |\mathrm{DIFF}_S(\mathrm{LOCAL}(S))| + 13.\end{aligned}$$

Using Lemma 2b, we see that

$$\begin{aligned}&\sum_{X\in\{L,R\}} \mathrm{COL}(\mathrm{PS}, X) - \sum_{X\in\{L,R\}} \mathrm{COL}(\mathrm{OPT}, X) \\ (5)\qquad &\leqq \tfrac{3}{2}\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, L, X, R)| + \tfrac{1}{2}\sum_{X\in\{T,B\}} |\mathrm{OPP}(T, X, B, X)| \\ &\quad + \tfrac{1}{2}\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, T, X, B)| + \mathrm{LAPSAME}_S + |\mathrm{DIFF}_S(\mathrm{LOCAL}(S))| + 13.\end{aligned}$$

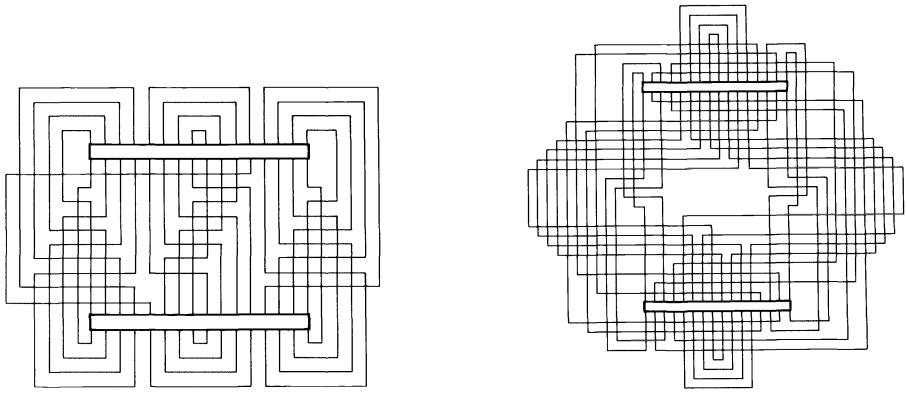
Combining inequalities (1) and (5), we have

$$\begin{aligned}|\mathrm{PS}(P)| - |\mathrm{OPT}(P)| \leqq & \tfrac{7}{4}\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, T, X, B)| + \tfrac{7}{4}\sum_{X\in\{T,B\}} |\mathrm{OPP}(T, T, B, T)| \\ & + |\mathrm{OPP}(T, T, B, B)| + \tfrac{3}{2}\sum_{X\in\{T,B\}} |\mathrm{OPP}(X, L, X, R)| \\ & + |\mathrm{DIFF}_S(\mathrm{LOCAL}(S))| + \tfrac{1}{2}|\mathrm{SAME}_S(\mathrm{LOCAL}(S))| \\ & + \tfrac{1}{2}\mathrm{LAPSAME}_S + 24.\end{aligned}$$

Comparing this formula with Lemma 3, we see that

$$|\mathrm{PS}(P)| - |\mathrm{OPT}(P)| \leq \tfrac{9}{10}|\mathrm{OPT}(P)| + 24.$$

Finally, we show that the bound is tight, in the sense that the factor of $\frac{19}{10}$ cannot be made smaller. For any positive integer n, the routing problem P contains nets as follows. (See Fig. 16.) Each module contains $4n$ columns. Number the columns of the modules from left to right. For $1 \leqq i \leqq n$ and $3n+1 \leqq i \leqq 4n$, a terminal in column i on the top of M_T is to be connected to a terminal in column i on the top of M_B. Similarly, for $1 \leqq i \leqq n$ and $3n+1 \leqq i \leqq 4n$, a terminal in column i on the bottom of M_T is to be connected to a terminal in column i on the bottom of M_B. For $1 \leqq i \leqq n$, terminals in columns $n+i$ and $3n-i+1$ on the top of M_T are to be connected and terminals in columns $n+i$ and $3n-i+1$ on the top of M_B are to be connected. For $1 \leqq i \leqq n$, a terminal in column $n+i$ on the bottom of M_T is to be connected to a terminal in column $2n+i$ on the top of M_B. For $1 \leqq i \leqq n-1$, a terminal in column $n+i+1$ on the top of M_B is to be connected to a terminal in column $2n+i$ on the bottom of M_T. Finally, a terminal in column $3n$ on the bottom of M_T is to be connected to a terminal in column $n+1$ on the top of M_B. An optimal routing will use at most $4n+2$ tracks



(a) (b)

FIG. 16. *A worst-case example for* PS. (*a*) *An optimal routing.* (*b*) *The* PS *routing.*

and $2n+2$ columns in addition to those intersecting the modules. (See Fig. 16a.) However, PS may use as many as $2n$ tracks above M_T, $2n$ tracks below M_B, $5n+2$ tracks between the modules, and $3n$ columns on each side of the modules (see Fig. 16b). Since there are no terminals on the sides of the modules, we may allow their height to be a constant k independent of n. Adding up the above and the $4n$ columns of the modules, we have

$$\frac{|\mathrm{PS}(P)|}{|\mathrm{OPT}(P)|} \geq \frac{19n+2+k}{10n+4+k}.$$

Therefore, the constant factor of $\frac{19}{10}$ cannot be made smaller. □

5. Conclusions. The main contribution of this paper lies in showing that there is an algorithm whose worst-case performance is asymptotically at most $\frac{19}{10}$ times that of an optimal routing with respect to the perimeter measure. It is hoped that this work will stimulate interest in developing better general algorithms whose worst-case performance can be proved to be bounded relative to optimal. Most previous work has been directed at solving restricted problems or at finding heuristics that sometimes do very badly even though they usually perform well.

A second contribution of the paper lies in showing that some simple proof ideas can be used to produce lower bounds on the size of an optimal routing that are good enough to lead to a tight analysis of the performance of PS. These techniques may be useful in analyzing the worst-case performance of other algorithms. They may also be useful in evaluating routings by calculating an estimate of the size of an optimal routing for comparison.

A natural variation of the two-module problem studied here is the subcase in which every net has a terminal on each module. The worst-case performance of PS is probably better than $\frac{19}{10}$ in this case, but has not been analyzed.

It would be interesting to analyze how much of the $\frac{19}{10}$ factor is due to channel routing. A better channel routing algorithm could probably improve the worst-case performance substantially. Baker, Bhatt, and Leighton [1] have developed an approximation algorithm for channel routing that routes a channel in a width that is within a constant times the optimal width. The constant is quite large, in fact greater than 50. However, for the perimeter measure and two-terminal nets, this approximation algorithm probably has better asymptotic performance than PS. It would be interesting to analyze the performance of the PS algorithm with this approximation algorithm substituted for SPLIT.

The techniques of this paper can be extended to multiterminal nets. The techniques for calculating lower bounds on the size of an optimal routing, in particular, can be extended in a straightforward manner to multiterminal nets at the expense of more case analysis. It is also clear that an algorithm can be produced whose worst-case performance is asymptotically at most a constant times optimal.

It would be interesting to find an algorithm for the two-module problem whose worst-case performance is at most a constant times optimal for the area measure. The area measure does not allow the same kind of tradeoff between tracks and columns that was exploited in the proof for PS. It would also be interesting to obtain bounds for the average performance of PS or other algorithms.

Acknowledgment. The author would like to thank A. V. Aho and T. G. Szymanski for their helpful comments on an earlier draft of this paper.

REFERENCES

[1] B. S. BAKER, S. N. BHATT AND F. T. LEIGHTON, *An approximation algorithm for Manhattan routing*, Proc. 15th Annual Symposium on Theory of Computing, Boston, MA, April, 1983, pp. 477–486.

[2] D. J. BROWN AND R. L. RIVEST, *New lower bounds for channel width*, in VLSI Systems and Computations, H. T. Kung, Bob Sproull and Guy Steele, eds., Computer Science Press, Rockville, MD, 1981, pp 178–185.

[3] M. S. CHANDRASEKHAR AND M. A. BREUER, *Optimum placement of two rectangular blocks*, Technical Report, Dept. Electrical Engineering, Univ. Southern California, Los Angeles, CA 90007.

[4] D. N. DEUTSCH, *A dogleg channel router*, in Proc. 13th Design Autmation Conference, IEEE, 1976, pp. 425–433.

[5] D. DOLEV, K. KARPLUS, A. STRONG AND J. ULLMAN, *Optimal wiring between rectangles*, in Proc. 13th Annual Symposium on Theory of Computing, Milwaukee, WI, May, 1981, pp. 312–317.

[6] F. GAVRIL, *Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph*, this Journal, 1 (1972), pp. 180–187.

[7] T. F. GONZALEZ AND S.-L. LEE, *An optimal algorithm for optimal routing around a rectangle*, Proc. Twentieth Annual Allerton Conference on Communication, Control, and Computing, Allerton, IL, 1982, pp. 636–645.

[8] T. KAWAMOTO AND Y. KAJITANI, *The minimum width routing of a 2-row 2-layer polycell layout*, Proc. 16th Design Automation Conference, 1979, pp. 290–296.

[9] A. S. LAPAUGH, *A polynomial time algorithm for optimal routing around a rectangle*, Proc. of 21st Annual Symposium on Foundations of Computer Science, Syracuse, NY, Oct., 1980, pp. 282–293.

[10] R. Y. PINTER, *Optimal routing in rectilinear channels*, in VLSI Systems and Computations, H. T. Kung, Bob Sproull and Guy Steele, eds., Computer Science Press, Rockville, MD, 1981, pp. 160–177.

[11] R. L. RIVEST, A. E. BARATZ AND G. MILLER, *Provably good channel routing algorithms*, in VLSI Systems and Computations, H. T. Kung, Bob Sproull and Guy Steele, eds., Computer Science Press, Rockville, MD, 1981, pp. 153–159.

[12] A. SIEGEL AND D. DOLEV, *The separation for general single-layer wiring barriers*, in VLSI Systems and Computations, H. T. Kung, Bob Sproull and Guy Steele, eds., Computer Science Press, Rockville, MD, 1981, pp. 143–152.

[13] T. G. SZYMANSKI AND M. YANNAKAKIS, personal communication, 1982.

[14] M. TOMPA, *An optimal solution to a wire-routing problem*, in Proc. of 12th Annual Symposium on Theory of Computing, Los Angeles, CA, April, 1980, pp. 161–176.

SIAM J. COMPUT.
Vol. 15, No. 1 February 1986
© 1986 Society for Industrial and Applied Mathematics
013

ALPHABETIC MINIMAX TREES OF DEGREE AT MOST t*

D. COPPERSMITH†, M. M. KLAWE‡ AND N. J. PIPPENGER‡

Abstract. Problems in circuit fan-out reduction motivate the study of constructing various types of weighted trees that are optimal with respect to maximum weighted path length. An upper bound on the maximum weighted path length and an efficient construction algorithm will be presented for trees of degree at most t, along with their implications for circuit fan-out reduction.

Key words. optimal weighted tree, minimax tree, t-ary tree, fanout reduction, logical circuits

In this paper we consider the problem of constructing, for any list $w_1, \cdots, w_n$ of integers, a tree T with maximum degree at most t (where $t \geqq 2$ is a fixed integer) and leaves $v_1, \cdots, v_n$ in left to right order such that $f_T(w_1, \cdots, w_n) = \max_{1 \leqq i \leqq n} (l_i + w_i)$ is minimized, where l_i denotes the length of the path in T from the root to the leaf v_i. We will call the minimum value $f(w_1, \cdots, w_n) = \min_T f_T(w_1, \cdots, w_n)$ the *minimax weighted path length.*

This work was motivated by the results of Kirkpatrick and Klawe [2] dealing with the analogous problem of constructing t-ary trees, that is, trees in which the degree of every internal vertex is exactly t. As in [2], we obtain a linear algorithm for the case of integer weights and prove a tight upper bound on $f(w_1, \cdots, w_n)$ in terms of $w_1, \cdots, w_n$. Like those in [2], these results can be applied to obtain a circuit fan-out reduction algorithm that preserves size and depth to within constant multiplicative factors without increasing the number of edge crossings. Our relaxation of the constraint on the degrees of internal vertices in the tree results in a smaller multiplicative factor for depth, but a larger multiplicative factor for size. This relaxation also causes some of the proofs to be easier than those in [2]; indeed the ideas in this paper inspired simplifications of both the algorithm and the proof of the upper bound in [2]. Kirkpatrick and Klawe show that an $O(n \log n)$ algorithm for real weights can be obtained from their linear integral weight algorithm, and that the upper bound also applies to the case of real weights. The same methods could be applied to our results to yield analogous results for the case of real weights.

If the leaves of a tree are weighted, we can extend the weighting to the internal vertices of the tree by defining the weight of an internal vertex to be one plus the maximum of the weights of its sons. With this extension, if the leaves $v_1, \cdots, v_n$ have weights $w_1, \cdots, w_n$, then the weight of the root is exactly $f_T(w_1, \cdots, w_n)$. This yields an equivalent formulation of our problem as that of constructing a tree with maximum degree at most t with leaf weights $w_1, \cdots, w_n$ in left to right order such that the weight of the root is minimized. The next lemma gives three modifications which can be made to a list of weights without increasing the minimax weighted path length.

LEMMA 1. *If $w_1, \cdots, w_n$ is a list of weights, then none of the following modifications increase the minimax weighted path length. Define $w_0 = w_{n+1} = \infty$.*

(a) *If $n > 1$ and $w_i \leqq \min(w_{i-1}, w_{i+1})$ for some i with $1 \leqq i \leqq n$, then replace w_i by* $\min(w_{i-1}, w_{i+1})$.

(b) *If $\min(w_i, w_{i+s+1}) \geqq 1 + \max(w_{i+1}, \cdots, w_{i+s})$ for some $s \leqq t$ and i with $0 \leqq i \leqq n - s$, then replace the s weights $w_{i+1}, \cdots, w_{i+s}$ by the single weight* $1 + \max(w_{i+1}, \cdots, w_{i+s})$.

* Received by the editors May 2, 1983, and in revised form May 21, 1984.

† IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.
‡ IBM Research Laboratory, San Jose, California 95193.

(c) *If* $w_i = w_{i+1} = \cdots = w_{i+t-1} \leqq w_{i+t} - 1$ *for some* i *with* $1 \leqq i \leqq n-t+1$, *then replace the* t *weights* $w_i, \cdots, w_{i+t-1}$ *by the single weight* $1 + w_{i+t-1}$.

Proof. In each case the proof consists of indicating how an optimal tree for the original list of weights can be altered to obtain a tree for the modified list in such a way that the weight of the root is not increased. Let T be a tree that is optimal for $w_1, \cdots, w_n$. In case (a), v_i must have v_{i-1}, v_{i+1} or one of their ancestors as a brother, so that increasing the weight of v_i cannot increase the weight of its father, and hence cannot increase the weight of the root. In case (b), choose a vertex x in T such that all the leaves in the subtree rooted at x are in the set $\{v_{i+1}, \cdots, v_{i+s}\}$ and such that the distance from x to the root is minimal. Note that x must have v_i, v_{i+s+1} or one of their ancestors as a brother. Thus replacing the subtree rooted at x by a single vertex with weight $1 + \max(w_{i+1}, \cdots, w_{i+s})$ and removing any leaves in $\{v_{i+1}, \cdots, v_{i+s}\}$ that are outside the subtree rooted at x cannot increase the weight of the root. Finally, in case (c), there are two possibilities. The first is that the brothers of v_{i+t-1} are precisely $\{v_{i+j}, \cdots, v_{i+t-2}\}$ for some j with $0 \leqq j \leqq t-2$. In this case the change corresponds to replacing the weights of $v_{i+j}, \cdots, v_{i+t-1}$ by the weight of their father which has weight $1 + w_{i+t-1}$, and removing the other leaves $v_i, \cdots, v_{i+j-1}$. In the second possibility v_{i+t-1} has as a brother v_{i+t}, an ancestor of v_{i+t}, or an ancestor of v_{i+j} for some j with $0 \leqq j \leqq t-2$. Thus removing the leaves v_{i+j} for $0 \leqq j \leqq t-2$ and increasing w_{i+t-1} by 1 does not increase the weight of the root. □

We now sketch an algorithm that, given a list $w_1, \cdots, w_n$ of integer weights, constructs an optimal tree. First add dummy weights $w_0 = w_{n+1} = \infty$ to each end of the list. At any stage of execution there will be a list of weights of vertices that have not yet been assigned fathers and a pointer dividing the list into two parts, the left sublist and the right sublist. The algorithm will operate so that the left sublist always forms a nonincreasing sequence from left to right. We call a weight K in the left sublist a *step weight* if K is strictly larger than the weight on its right or if K is the rightmost weight in the left sublist. Initially the pointer is placed so that it points between w_0 and w_1. We now describe the main procedure of the algorithm. Suppose the weights lying immediately to the left and right of the pointer are L and R respectively. If $L \geqq R$, then the algorithm simply moves the pointer past R. Otherwise, let K be the rightmost step weight such that either $K \geqq R$ or there are at least t weights lying strictly between K and R in the list.

First suppose that there are at least t weights between K and R. If at least two of these weights are step weights, find the leftmost such step weight, say K', remove all weights lying between K' and R and insert a new weight equal to K' between K' and the pointer. (We rely here on (b) followed by (a) in Lemma 1 and on the fact that all weights are integers.) If L is the only step weight, remove the t rightmost weights to the left of the pointer and insert a new weight equal to $L+1$ to the right of the pointer. (We rely here on (c) in Lemma 1.) Now suppose that there are less than t weights between K and R, and hence that $K \geqq R$. Remove all weights between K and R and insert a new weight equal to R to the left of the pointer. (We rely again on (b) followed by (a) in Lemma 1.) Note that after applying this procedure the weights to the left of the pointer still form a nonincreasing sequence.

The algorithm operates by repeating this procedure until exactly three weights are left in the list. As the ∞ weights are never removed, the final list is of the form ∞, w, ∞. Interpreting modifications of types (b) and (c) in the obvious manner of making the new weight the weight of the father of the vertices whose weights were removed from the list, it is clear that this algorithm constructs an optimal tree and that the minimax weighted path length is w.

To implement the algorithm efficiently, it is only necessary to maintain the position of the pointer and (in a doubly-linked list) the step weights and their positions in the left sublist. In any execution of the main procedure, all but the leftmost of the step weights examined will no longer be step weights at the end of the procedure. From this and by examining the other operations in the procedure it is easy to see that the running time of the algorithm is at most linear in the total number of vertices in the tree (which is at most $2n-1$), with a coefficient that is independent of t.

We now prove an upper bound on the minimax weighted path length for the case of integer weights.

LEMMA 2. *If $w_1, \cdots, w_n$ are integers, then*

$$f(w_1, \cdots, w_n) < 1 + \log_t 2 + \log_t \left(\sum_{1 \leq i \leq n} t^{(w_i)} \right).$$

Proof. For W the list of weights $w_1, \cdots, w_n$, define $g(W) = \sum_{1 \leq i \leq n-1} t^{\max(w_i, w_{i+1})}$. Then it is easy to verify that if W' is any list obtained by modifying W according to (a), (b) or (c) of Lemma 1, then $g(W') \leq g(W)$. Suppose w is the weight of the root of the tree constructed by our algorithm and suppose the weights of the sons of the root are $x_1, \cdots, x_s$. Let X be the list $x_1, \cdots, x_s$. By iterating the observation above, $g(X) \leq g(W)$. Combining this with the obvious inequalities $t^w \leq t g(X)$ and $g(W) < 2 \sum_{1 \leq i \leq n} t^{(w_i)}$ and taking logarithms completes the proof. □

Remark 3. The corresponding upper bound in [2] for t-ary trees is $2 + \log_t (\sum_{1 \leq i \leq n} t^{(w_i)})$.

We now describe the application to circuit fan-out reduction in more detail, in order to compare the effect of using various tree constructions. Suppose G is an acyclic directed graph with fan-in bounded by s. In [1], an algorithm is given that constructs a new graph G' with fan-out at most t, by replacing each vertex of G that has fan-out greater than t with a tree connecting that vertex to its sons. By choosing trees that minimize the increase in depth while having degrees bounded by t, it can be proved that $\text{Size}(G') \leq (1 + (s-1)/(t-1))\ \text{Size}(G) + (q-1)/(t-1)$ and $\text{Depth}(G') \leq (1 + \log_t s)\ \text{Depth}(G) + \log_t q$, where q is the number of outputs of G. Unfortunately, however, using trees that minimize the increase in depth will generally increase the number of edge crossings.

In [2] it is observed that using alphabetic minimax trees avoids the increase in edge crossings in exchange for a poorer bound on the depth of the new graph. Thus, although the size bound remains the same, the depth bound becomes $\text{Depth}(G') < (2 + \log_t s)\ \text{Depth}(G) + \log_t q$. Finally, using the trees described in this paper also avoids the increase in edge crossings with a better depth bound than that of [2] but a poorer size bound. More precisely, if our algorithm is used, the bounds become $\text{Depth}(G') \leq (1 + \log_t (2s))\ \text{Depth}(G) + \log_t q$ and $\text{Size}(G') \leq s\ \text{Size}(G) + q - 1$. We will see, however, that the size bound can be improved to $\text{Size}(G') \leq (1 + (s-1)/(t-1) + (s+1)(t-2)/3(t-1))\ \text{Size}(G) + (q+1)/(t-1) + (q+1)(t-2)/3(t-1)$, by adding an extra phase to our algorithm to reduce the size of the optimal tree.

Although our algorithm constructs an optimal tree, it does not necessarily construct the optimal tree with the smallest number of vertices. In the worst case, which occurs for sequences of the form $2j-1, 2j-3, \cdots, 3, 1, 2, 4, \cdots, 2j-2, 2j$, our tree has $n-1$ internal vertices, although there is an optimal tree with $\lceil (n-1)/(t-1) \rceil$ internal vertices. Although we have been unable to find a linear algorithm which produces the smallest optimal tree, by applying a simple linear "compaction" algorithm to our optimal tree we obtain a tree in which the number of internal vertices is at most $\lfloor (n-1)/(t-1) +$

$(t-2)(n+1)/3(t-1)\rfloor$. We will also give an example showing that there are sequences for which the smallest optimal tree has this many internal vertices.

A *leaflet* is defined to be an internal vertex that has only leaves as sons and has degree less than t. The object of the compaction algorithm is to produce a tree satisfying the following three conditions.

(1) Each internal vertex either has degree t or is a leaflet.

(2) No two adjacent leaves are the sons of different leaflets.

(3) Each leaflet has degree at least 2.

It is not hard to design a linear algorithm that accomplishes this. Condition (1) can be met by a phase that processes the vertices in preorder (or any other order that visits each vertex before its sons) and raises the degree of nonleaflet internal vertices by making grandsons into sons. Condition (2) can be met by a phase that, whenever two "offending" adjacent leaves are located, moves sons from the leaflet with smaller weight to the leaflet with larger weight until either the first leaflet has degree 1 and can be collapsed (i.e. replaced by its son), or the second has degree t and is no longer a leaflet. Finally, condition (3) can be met by collapsing leaflets with only one son.

We shall show that any tree satisfying the three conditions above has at most $\lfloor(n-1)/(t-1)+(t-2)(n+1)/3(t-1)\rfloor$ internal vertices.

LEMMA 4. *If T is tree with n leaves satisfying conditions* (1), (2) *and* (3), *then t has at most $\lfloor(n-1)/(t-1)+(t-2)(n+1)/3(t-1)\rfloor$ internal vertices.*

Proof. Let k be the number of leaflets and let p be the number of leaves that are sons of leaflets. Obviously, $p \geqq 2k$, by condition (3). Thus the number of internal vertices is at most $k+(n-k-1)/(t-1)$, since if we remove all leaves that are sons of leaflets from T, the remaining tree is a t-ary tree with $n-p+k$ leaves and so its number of internal vertices is exactly $(n-p+k-1)/(t-1)$. Finally, it is easy to see that $k \leqq (n+1)/3$, by conditions (2) and (3), which yields the stated bound. □

We conclude our paper with examples which show that even after compaction our optimal tree is not necessarily the smallest optimal tree, and also that there are lists of weights for which the number of internal vertices in the smallest optimal tree attains the bound in the preceding lemma. Let $W(n)$ be the list $w_1, \cdots, w_n$, where $w_i = 1$ for $i = 0 \bmod 3$ and $w_i = 0$ otherwise. For $n = 9$ and $t = 3$, the balanced ternary tree is optimal and has only four internal vertices. Our algorithm, however, begins by pairing the three pairs of 0 weights, and it can easily be checked that no matter how the compaction algorithm is implemented, the resulting compacted tree will have five internal vertices. On the other hand, if $n = \lceil 3t^k/2\rceil$ for some $k \geqq 1$ and $t \geqq 3$, then there is only one optimal tree for $W(n)$, and it has $\lfloor(n-1)/(t-1)+(t-2)(n+1)/3(t-1)\rfloor$ internal vertices.

REFERENCES

[1] H. J. HOOVER, M. M. KLAWE AND N. J. PIPPENGER, *Bounding fan-out in logical networks*, J. Assoc. Comput. Mach., 31 (1984), pp. 13-18.

[2] D. G. KIRKPATRICK AND M. M. KLAWE, *Alphabetic minimax trees*, this Journal, 14 (1985), pp. 514-526.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986

© 1985 Society for Industrial and Applied Mathematics

ON SHORTEST PATHS IN POLYHEDRAL SPACES*

MICHA SHARIR†‡ AND AMIR SCHORR†

Abstract. We consider the problem of computing the shortest path between two points in two- or three-dimensional space bounded by polyhedral surfaces. In the 2-D case the problem is easily solved in time $O(n^2 \log n)$. In the general 3-D case the problem is quite hard to solve, and is not even discrete; we present a doubly-exponential procedure for solving the discrete subproblem of determining the sequence of boundary edges through which the shortest path passes. Finally we consider a favorable special case of the 3-D shortest path problem, namely that of finding the shortest path between two points along the surface of a convex polyhedron, and solve it in time $O(n^3 \log n)$.

Key words. Euclidean shortest paths, convex polyhedron, computational geometry

1. Introduction.

The problem of finding the shortest path between two points in Euclidean space bounded by a finite collection of polyhedral obstacles is a special case of the more general problem of planning optimal collision-free paths for a given robot system (here we treat the robot as a single moving point).

In two-dimensional space the problem is easy to solve, because the shortest path between two given points must be a polygonal line whose vertices are corners of the given polygonal obstacles, so that the problem can be immediately reduced to a discrete graph searching, and can be solved in time $O(n^2 \log n)$, where n is the number of obstacle corners. This two-dimensional problem has been considered by Lozano-Perez and Wesley [LW], and later also by Lee and Preparata [LP]. In some special cases, considerably more efficient algorithms exist. For example, if the free space within which the shortest path is sought is the interior of a simple polygon, then the shortest path can be found in time $O(n \log n)$ [LP], [Ch]. As another example, if all the barriers are straight segments parallel to each other, then the problem can again be solved in time $O(n \log n)$ [LP] (a similar favorable case, but with a somewhat different solution, is discussed below). Another favorable case is noted by Tompa [To], where the obstacles are all convex and aligned in a certain manner along a straight line.

In three-dimensional space the problem becomes much harder. In this case the shortest path between two given points can also be shown to be a polygonal line, but all we can say about its vertices is that they lie on edges of the given polyhedral obstacles. Thus the problem is by no means discrete. Even if one knew the sequence of obstacle edges through which the desired shortest path passes, the calculation of the points of contact of the path with these edges requires solution of high-degree algebraic equations, which must be accomplished either by numerical approximate methods, or by precise, but very inefficient, symbolic algebraic calculations. Even the calculation of the sequence of obstacle edges through which the shortest path passes seems to be very difficult, and we do not know of better than doubly-exponential-time algorithms for this subproblem. Papadimitriou [Pa] has recently presented an approximating algorithm for the general three-dimensional polyhedral shortest path problem, which runs in pseudo polynomial time.

* Received by the editors December 19, 1983, and in revised form November 3, 1984. A preliminary version of this paper (Copyright 1984, Association for Computing Machinery, Inc.) appeared in Proceedings of the Sixteenth ACM Symposium on the Theory of Computing, Washington, DC, 1984, pp. 144–153.

† School of Mathematical Sciences, Tel Aviv University, Ramat Aviv, 69978 Tel Aviv, Israel.

‡ The work of this author has been supported in part by a grant from the U.S.-Israeli Binational Science Foundation.

However, in certain special cases the 3-D problem is not so hard to solve. We will consider the case of finding the shortest path between two points along the surface of a convex polyhedron, a problem that has been suggested originally by H. E. Dudeney as a mathematical puzzle in 1903 (see [Ga, p. 36]; in his original formulation a spider has to crawl along the surface of a cube to reach a fly in the shortest possible manner). We present an $O(n^3 \log n)$ algorithm for this problem, which exploits the special structure of "geodesic" paths along the surface of a convex polyhedron. Our technique has later been extended by O'Rourke, Suri and Booth [OSB] to find shortest paths along the surface of a nonconvex polyhedron, and also been improved by Mount [Mo].

The paper is organized as follows. Section 2 presents a straightforward solution to the 2-D problem; § 3 discusses the general 3-D problem, develops the relevant theory, and presents a doubly-exponential algorithm for solving the problem. Finally, § 4 analyzes the problem of shortest paths along a convex polyhedron and an algorithm for solving this problem is presented in § 5.

2. The two-dimensional case. If one assumes that the solid obstacles are all orthogonal prisms whose heights are all parallel to, say, the z-axis, and are enclosed between a floor and a ceiling, and if the two points X, Y, between which an optimal path is sought, are assumed both to have the same height, then the 3-D case of our problem may be reduced to the 2-D case, since then the optimal path will lie entirely in a horizontal plane containing X and Y, and the given solid obstacles will intersect this plane in a collection of polygonal obstacles. Let us therefore assume that V is a closed two-dimensional region bounded by a collection of polygonal walls and other polygonal obstacles, and let X, Y be two points in V. The 2-D shortest path problem is then to find a (Euclidean) shortest path between X and Y which is wholly contained in V. This problem, and several special cases of it, has already been considered by several other people ([LW], [LP], [Ch], [To]). The general approach used below has been suggested by Lozano-Perez and Wesley [LW], but without any explicit analysis of its complexity. Most of the other relevant work has involved special cases of the problem, where more efficient solutions than the one presented below exist. To simplify the following discussion, we assume that each corner of the boundary of V is incident to just two edges (although the method described below will also apply in cases where this assumption does not hold). It is easily seen that the shortest route from X to Y which is wholly contained in V is a polygonal path connecting X to Y whose intermediate corners are all vertices of the polygonal walls and obstacles bounding V. Furthermore, a vertex p will follow another vertex q in such a route only if p and q are *visible* from each other (in V), i.e. only if the straight segment joining p and q lies wholly in V. See Fig. 2.1 for illustration of the concepts just discussed.

Thus, to solve the 2-D version of our problem, we first construct a *visibility graph* VG, whose nodes are X, Y, and the vertices of the boundary of V, and each of whose edges connects a pair of vertices visible from each other, and has length equal to the distance between these vertices. Then we search through VG to find the shortest path in VG from X to Y. This yields a polynomial-time discrete algorithm which can be implemented to run in time $O(n^2 \log n)$. For the sake of completeness, we sketch here such a straightforward implementation.

Let N denote the set containing X, Y, and all the vertices of the boundary of V (bd (V) for short). For each $u \in N$, we will find all points in N visible from u by using the following algorithm which is based upon a "plane-sweeping" technique similar to those used in [Sh], [NP]. Specifically, for each orientation θ let $l = l_\theta$ denote a ray extending from u at orientation θ. Let $M(\theta)$ denote the list of all edges in bd (V)

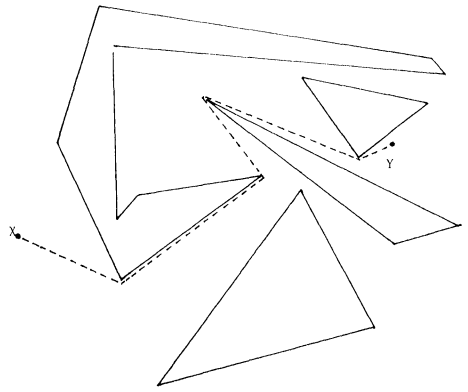


FIG. 2.1. *The shortest path problem in* 2-D *polygonal space.*

whose interior intersects l, sorted in increasing distance of these intersections from u. We maintain $M(\theta)$ as a 2-3 tree, and can easily compute an initial value of M for some starting orientation θ_0 in time $O(n \log n)$.

We next rotate the ray l in clockwise direction about u, and update the value of $M(\theta)$ each time l crosses an orientation θ at which M changes, either due to addition of new edges into M or deletion of edges from M, or both. It is clear that such changes can occur only at orientations θ of l at which l passes through another corner in N. We thus iterate through these critical orientations in clockwise order (including also the orientations of the rays connecting u to X, Y). Let θ_p be such an orientation at which l passes through $p \in N$. Then M will change at θ_p so that the edges incident to p and presently in M are deleted from M, and the other edges incident to p are inserted into M. However, if the ray l passes simultaneously through several points $p \in N$, then we have to remove from M all edges incident to any of these corners, and to add to M all other such incident edges, except for edges which lie on l, which are ignored in this updating process.

At each such orientation θ_p we also check whether the first edge in M has changed. If it did, then that corner p on l incident to this edge is visible from u, and we update VG by adding to it the edge (u, p). Otherwise none of these p's is visible from u. If p is X or Y, then it will be visible from u if and only if it precedes along l the first edge in M.

Iterating in this manner through all critical orientations θ_p, we find all points in N visible from u in time $O(n \log n)$. Hence, if we repeat this procedure for each $u \in N$, we can construct the visible graph VG in time $O(n^2 \log n)$.

Having constructed the visibility graph, we can then search through it to find the shortest path in VG connecting X and Y, using Dijkstra's algorithm (see e.g. [AHU]), which will run in time $O(n^2)$.

Improving the efficiency of the algorithm. We next discuss an alternative approach to the 2-D shortest path problem which leads in some special cases to more efficient algorithms. We begin by the following observation.

DEFINITION. For each point $Z \in V$, let $\pi(Z)$ denote a shortest path from X to Z through V.

LEMMA 2.1. *Let $Z_1, Z_2 \in V$. Then either $\pi(Z_1)$ and $\pi(Z_2)$ do not intersect each other or, if they do intersect at some last point Z, then Z must be a corner in* bd (V), *and the lengths of the initial portions of both paths between X and Z are equal.*

Proof. If $\pi(Z_1)$ and $\pi(Z_2)$ meet at a point Z, then the lengths of their initial portions up to Z must be equal, or else we could replace the longer such initial portion by the shorter one, and so shorten the length of one of these paths. The same argument also implies that Z must be a corner, because otherwise, after replacing one initial portion by the other one, we would obtain a shortest path to one of the points Z_1, Z_2 which is polygonal and has a corner at an interior point of V, contradicting the basic properties of such shortest paths noted above. Q.E.D.

We can use this observation to obtain an $O(n \log n)$ algorithm for finding the shortest path in the following special case (which is similar to the second special case considered in [LP]): Suppose that the boundary of V consists of k vertical lines, denoted $l_1, \cdots, l_k$, each of which contains several point apertures through which one can cross from one side of the line to the other. The passage is blocked however at all other points on these barriers. Suppose further that X lies to the left of all these barriers and that Y lies to the right of all of them. Note that the shortest path from X to Y through V must pass through exactly one aperture at each of the lines $l_1, \cdots, l_k$. Hence it can be found in $O(n^2)$ time, using a standard dynamic programming approach. However, using Lemma 2.1, we can improve this procedure as follows (a similar divide-and-conquer approach has been used by Reif [Re] for a different problem involving planar networks): Suppose that the barrier l_j has n_j apertures, $j = 1, \cdots, k$ (so that $\sum_{j=1}^{k} n_j = n$). We will process the barriers from left to right. For each barrier l_j we will compute for each aperture $Z \in l_j$ the length $d(Z)$ of the shortest path from X to Z through V, and also the aperture $p(Z) \in l_{j-1}$ through which the shortest path passes just before reaching Z. For each $Z \in l_1$ we put $d(Z) = |XZ|$ and $p(Z) = X$.

Suppose that these maps have already been computed for all apertures lying on l_{j-1}. Let Z be the median of all apertures along l_j. We compute $d(Z)$ and $p(Z)$ by trying to pass the path $\pi(Z)$ through each of the apertures in l_{j-1} (as in the standard dynamic programming approach). Let $W = p(Z)$, and assume that W is the mth highest point along l_{j-1}. Since, by Lemma 2.1, shortest paths can be assumed not to cross each other, it follows that we can partition the problem into two subproblems: First find the shortest paths leading to the highest half of the apertures along l_j, using only the highest m apertures along l_{j-1} as possible predecessors along such paths, and then repeat this procedure for the lowest half of the apertures on l_j, using this time only the lowest $n_{j-1} - m + 1$ apertures along l_{j-1} as possible predecessors.

Repeating this procedure recursively, it is easily seen that it will find (correctly) the shortest paths to all apertures on l_j in time $O((n_{j-1} + n_j) \log n_j)$. Hence if we iterate in this manner through all barriers l_j, we obtain an $O(n \log n)$ algorithm for finding the desired shortest path from X to Y.

The special case just considered has led to a favorable algorithm because of the regular structure of shortest paths in this case. Other special cases have been considered in [LP], [Ch], [To]. In the general case shortest paths may behave less regularly, although they still do not intersect each other. In fact, it is easily seen that, for a given starting point X, the set A containing all the corners of bd (V) and X, can be arranged in a tree T with X as the root, such that each corner u is the son of a point v if the last straight segment on $\pi(u)$ is vu. Moreover, for each $u \in A$ let $\Gamma(u)$ denote the set of all points $y \in V$ for which the last segment on $\pi(y)$ is uy. Note that $\Gamma(u)$ is nonempty only if the angle within V between the last segment vu on $\pi(u)$ and one of the edges e of bd (V) incident to u is greater than 180 degrees. In this case $\Gamma(u)$ is contained in the wedge formed between e and the straight ray continuing vu past u. It is also easy to show, by techniques similar to those used to analyze Voronoi diagrams (cf.

[Sh], [FAV]) that the boundary arcs between adjacent regions $\Gamma(u)$ are all straight or hyperbolic arcs, and that there are at most $O(n)$ such arcs.

In summary, the collection of shortest paths within V from some fixed starting point X can be characterized by a combinatorial structure whose size is $O(n)$, and it therefore seems likely that faster than quadratic algorithms for its construction should exist. There exist some other special cases where this is indeed the case. For example, if V is the interior of a simple polygon, then shortest paths within V can be computed in time $O(n \log n)$ (cf. [LP], [Ch]); another favorable case has been noted by Tompa [To] in connection with wire routing problems in VLSI. However, for general polygonal regions V the problem of computing shortest paths within V in faster than $O(n^2 \log n)$ time is still open.

3. The three-dimensional case. The situation becomes much more complicated when we pass to the 3-dimensional version of the problem. Here it is easy to check that the shortest path from X to Y consists of a polygonal path whose vertices (except for X and Y) lie on some of the edges of $\mathrm{bd}(V)$. The problem therefore is not immediately seen to be discrete, since there seems to be a continuum of potential paths to be considered. We will see, however, that the problem can be discretized, and develop an algorithm for finding the shortest path which runs in doubly exponential time in the number of wall edges.

We will find it useful to regard the wall edges as open segments, so that the wall corners are disjoint from the edges. The collection of wall edges and corners will sometimes be referred to as "wall objects".

We will first consider the following subproblem: Given a sequence $\boldsymbol{\xi} = (\xi_1, \cdots, \xi_n)$ of wall objects, find the shortest path π from X to Y constrained to pass through each of the objects $\xi_1, \cdots, \xi_n$ in this order, assuming that no other constraint is being imposed on the path. Let π consist of the segments $\pi_0, \cdots, \pi_n$.

LEMMA 3.1. *For each $i = 1, \cdots, n$, if ξ_i is a wall edge (rather than a wall corner) then the angles that π_{i-1} and π_i subtend at ξ_i are equal.*

Proof. This is well known, but to see this take the two planes formed by π_{i-1} and ξ_i and by π_i and ξ_i, and "unfold" them about ξ_i so as to make them coincident, with π_{i-1} and π_i lying on different sides of ξ_i. Since π is the shortest possible, the two segments π_{i-1} and π_i must be collinear in this common plane, and the angles which they form with ξ_i must therefore be equal. Q.E.D.

Let us temporarily assume that all wall objects $\xi_1, \cdots, \xi_n$ are segments, rather than points. An initial attempt to exploit Lemma 3.1 in finding the required shortest path π, is to determine the point Z_1 of contact of π with ξ_1. This point determines the angle at which π_0 meets ξ_1, which must be equal to the angle at which π_1 leaves ξ_1. Knowing this angle, we can determine the point(s) Z_2 at which π_1 will intersect ξ_2, and continue to proceed in this manner through all segments in $\boldsymbol{\xi}$. A correct choice for Z_1 is one in which the final angle at which π_n must leave ξ_n from their point of contact Z_n, is equal to the angle subtended at ξ_n by Z_nY.

However, this approach is problematical in the sense that each time we try to extend the path π from a point Z_i on ξ_i to ξ_{i+1}, there can exist two points Z, Z' on this edge for which the two angles subtended at ξ_i by the segments Z_iZ and Z_iZ' are both equal to the angle subtended at ξ_i by π_{i-1}. Thus, even if we fix the first point of contact Z_1, the number of paths that may arise in the way described above may be exponential in n. To overcome this difficulty, we will make use of the following observation.

LEMMA 3.2. *Let l_1, l_2 be two lines in 3-space, let A, B be two distinct points on l_1, and let C, D be two points on l_2. Let the angle between the vectors $\overrightarrow{AB}$ and $\overrightarrow{AC}$ (resp. between $\overrightarrow{AB}$ and $\overrightarrow{BD}$) be α (resp. β). Similarly, let the angle between the vector $\overrightarrow{CD}$ and $\overrightarrow{AC}$ (resp. between $\overrightarrow{CD}$ and $\overrightarrow{BD}$) be γ (resp. δ). Then $\alpha > \beta$ implies that $\gamma > \delta$ (see Fig. 3.1).*

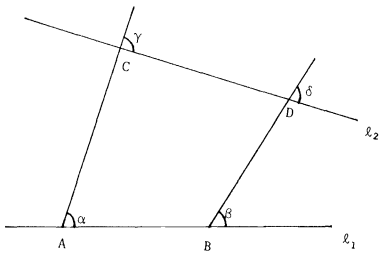


FIG. 3.1

Proof. First note that under these conditions we must have $C \neq D$ (and thus in particular γ and δ are well defined), for otherwise α would be an interior angle in the triangle ABC which is larger than the exterior angle β, which is impossible. Put $\overrightarrow{AB} = \mathbf{u}$, $\overrightarrow{AC} = \mathbf{x}$, $\overrightarrow{BD} = \mathbf{y}$. Then $\overrightarrow{CD} = \mathbf{u} + \mathbf{y} - \mathbf{x}$. Since $\alpha > \beta$ we have

$$\frac{\mathbf{x} \cdot \mathbf{u}}{|\mathbf{x}|} \leqq \frac{\mathbf{y} \cdot \mathbf{u}}{|\mathbf{y}|}.$$

To prove $\gamma > \delta$ we need to show that

$$\frac{\mathbf{x} \cdot (\mathbf{u}+\mathbf{y}-\mathbf{x})}{|\mathbf{x}|} < \frac{\mathbf{y} \cdot (\mathbf{u}+\mathbf{y}-\mathbf{x})}{|\mathbf{y}|}.$$

By the first inequality, it suffices to show that

$$\frac{\mathbf{x} \cdot (\mathbf{y}-\mathbf{x})}{|\mathbf{x}|} < \frac{\mathbf{y} \cdot (\mathbf{y}-\mathbf{x})}{|\mathbf{y}|}$$

or that

$$\frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}|} - |\mathbf{x}| < |\mathbf{y}| - \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{y}|}$$

or

$$\mathbf{x} \cdot \mathbf{y} < |\mathbf{x}||\mathbf{y}|,$$

which is immediate, since $\alpha \neq \beta$. Q.E.D.

Remark. The assertion in the preceding lemma can be extended to the case $A = B$ (so that α and β are undefined). If $C \neq D$ then we always have $\gamma > \delta$ since γ is an exterior angle and δ is another interior angle in the triangle ACD.

LEMMA 3.3. *The shortest path π from X to Y which passes through the sequence of lines $\xi_1, \cdots, \xi_n$ in this order is unique. (Note that we assume here that $\xi_1, \cdots, \xi_n$ are full lines, or, alternatively that π passes through interior points of $\xi_1, \cdots, \xi_n$.)*

Proof. Suppose that there exist two shortest paths π, π' from X to Y through the lines $\xi_1, \cdots, \xi_n$. Apply Lemma 3.2 for each of the (skew) quadrangles whose edges are $\xi_i, \xi_{i+1}, \pi_i, \pi_i'$, $i = 0, \cdots, n$, where π_i (resp. π_i') is the ith segment along π (resp. π') (see Fig. 3.2).

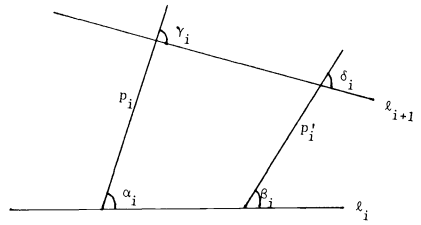


FIG. 3.2

Lemma 3.1 implies that $\alpha_{i+1} = \gamma_i$ and $\beta_{i+1} = \delta_i$ for each $i = 0, \cdots, n-1$. The first divergence of π and π' (i.e. the first j for which π_j and π_j' have the same starting point but different endpoints) forms a triangle, and we therefore have $\gamma_j > \delta_j$. Thus, inductive applications of Lemma 3.2 imply that $\alpha_i > \beta_i$ for each $i = j+1, \cdots, n$. In particular, $\alpha_n > \beta_n$, which is impossible since π_n and π_n' meet at Y, forming a second triangle (which might degenerate to a single segment), a contradiction which proves the lemma. Q.E.D.

Lemma 3.3 can be strengthened as follows: Call a path π from X to Y which passes through the lines $\xi_1, \cdots, \xi_n$ *geodesic* (or *locally shortest*) if, for each $i = 1, \cdots, n$, the path π enters and leaves ξ_i at equal angles. It is easily checked that every path whose length is a local extremum (as a function of its points of contact with $\xi_1, \cdots, \xi_n$) is geodesic. It is plain that Lemma 3.3 remains true if one assumes that the paths in question are only geodesic. Hence we have

COROLLARY 3.4. *There exists a unique geodesic path π from X to Y which passes through a given sequence of lines $\xi_1, \cdots, \xi_n$.*

In other words, the length of the path (as a function of the contact points) has one global minimum, and no other local extrema. Thus, if we continue to assume that $\xi_1, \cdots, \xi_n$ are full lines, then we can use two different techniques for the calculation of the contact points of the required shortest path with these lines. The first technique uses approximate numerical methods for finding the required minimum. For example, we can initially pass a path π^0 through an arbitrary sequence of points, one on each of the given lines. Then, iteratively, improve the path by replacing each contact point at which the incoming and outgoing angles are not equal by another point on the same line at which these angles become equal (without changing the other contact points). An explicit formula for finding the new point of contact can be readily obtained, using elementary vector techniques. (The problem involved here is, given two points $\mathbf{a}$ and $\mathbf{b}$ outside a given line l, to find a point $\mathbf{x}$ on l such that both vectors $\mathbf{x}-\mathbf{a}$ and $\mathbf{b}-\mathbf{x}$ form the same angle with l. We leave it to the reader to verify that this condition can be expressed by an equation which is quadratic in the coordinates of each of the points $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{x}$, and whose overall degree is 4.) Each such iterative step shortens the length of the path, and the sequence of paths thus obtained will converge to a path of locally extremal length, and hence to the desired shortest path, in virtue of Corollary 3.4.

If we wish to avoid numerical analysis, and insist on obtaining a precise solution using only symbolic calculations, then we can write down a system of n quartic equations in the n positions of the contact points of the path with the given lines, each such equation corresponding to one of the constraints given by Lemma 3.1, namely that at each line ξ the incoming and outgoing angles subtended by the shortest path be equal. This system can then be solved by elimination techniques (cf. [Wa]), leading to a single polynomial equation $p(x_1) = 0$ in, say, the position x_1 of the first point of contact. By Corollary 3.4, this equation will have a unique real solution which can then be rationally approximated to any desired degree of accuracy.

Note however that the resulting polynomial $p(x)$ will in general be of degree which is doubly exponential in n, because each elimination step computes resultants of polynomials, and thus results in polynomials of one less variable but of degree which is roughly the square of the degree of the previous polynomials, and because $n-1$ such elimination steps are required to eliminate all but one of the n variables in question. This indicates that the problem of computing the points of contacts (and thus also the length) of a geodesic path with a sequence of line segments is probably intractable. To be more precise, suppose that we have two candidate sequences ξ and η of lines, and we wish to determine whether the geodesic path passing through ξ is longer than the geodesic path passing through η. This can be solved precisely by standard methods involving symbolic calculations (as reviewed e.g. in [SS]; see also below), but these methods, which use space decomposition techniques closely related to the elimination method just noted, would also require doubly exponential time. We have not been able to prove that this problem is intractable. However, it is well-known that testing similar properties of real roots of a system of low degree polynomial equations in n variables (or even determining whether such roots exist) is NP-complete (this is shown e.g. by direct reduction from 3-SAT).

Another technical problem that should be noted is that if $\xi_1, \cdots, \xi_n$ are only segments and not full lines, the global minimum may be attained at points lying outside those segments. In this case it is clear that the shortest path π from X to Y constrained to pass through these segments in order will have to pass through some endpoints of these segments, at which it will generally form unequal incoming and outgoing angles. If this is the case, and if the endpoints through which π must pass are known, then the path-finding problem reduces to a collection of subproblems, each of which calls for the computation of the shortest path between some pair of points, which is constrained to pass through a specified sequence of lines. The solution of each such subproblem can be obtained by the methods outlined above.

Since in general the shortest path from X to Y will be a concatenation of subpaths, each connecting a pair of points (each of which is either X, Y, or a nonconvex wall corner) and constrained to pass through a sequence of wall edges, we will consider the path-finding problem as essentially solved (up to numerical or symbolic calculations of the sort discussed above) if we can specify the sequence of wall edges and corners through which the desired shortest path must pass. In this setting the problem is reduced to a purely combinatorial one, which we will refer to as the *combinatorial shortest path* problem. It is noteworthy that this combinatorial problem is at least solvable in finite time.

PROPOSITION 3.5. *The combinatorial shortest path problem is solvable in doubly exponential time by precise, symbolic calculations. If one is allowed to use numerical analysis techniques, then the problem can be solved in $O(n^n)$ steps, each step consisting of finding a shortest path constrained to pass through some sequence of wall edges and corners.*

Proof. Note first that it suffices to consider only a finite number of possible sequences of wall edges and corners through which the shortest path from X to Y can pass, because the shortest path will not pass twice through the same wall corner or edge. This makes it plain that the number of such sequences that need be considered is at most $O(n^n)$. For each pair of such sequences ξ, η, apply the procedure outlined above which uses symbolic computations, to find whether the shortest path constrained to pass through the elements of ξ is shorter than the path constrained to pass through the elements of η. The sequence for which the length of the corresponding path is smallest is then the solution to our combinatorial problem. Note that the basic step

that this method employs is comparison between two algebraic numbers, each of which is specified in terms of the unique real roots $r_1, \cdots, r_k$ of some system of k polynomials in k variables, for some $k \leqq n$ (these roots are the points of contact of one of the shortest paths with the wall edges through which it is constrained to pass). Such a comparison can be performed in precise terms using Collins' cylindrical algebraic decomposition technique for analyzing semi-algebraic sets defined by a system of p polynomial equalities and inequalities in k variables, having maximum degree m ([Co]; see also [SS]). Collins' technique runs in $O((mp)^{3^k})$ time. In our case each polynomial equation is quartic, and p, $k \leqq n$. It follows that the overall cost of the combinatorial shortest path problem is doubly exponential in n. When numerical methods are allowed in the evaluation of each of the shortest paths constrained to pass through some sequence of wall edges and corners, the problem can plainly be solved in $O(n^n)$ such numerical evaluations. Q.E.D.

It is currently an open problem whether faster procedures than the straightforward one just sketched exist for solving the combinatorial shortest path problem.

4. Shortest paths along a convex polyhedron. In this section we analyze the problem of calculating the shortest path between two points along the surface of a convex polyhedron in 3-space. This special case is favorable because of various properties of geodesic paths along a convex polyhedron. These properties will be analyzed in this section; an algorithm for the calculation of such shortest paths will be presented in the following section.

Let K be a given convex polyhedron, and let S denote its boundary. Let X and Y be two points on S. The problem that we consider is to calculate the shortest path from X to Y constrained to lie along S. It will be more convenient to consider a somewhat more general problem, namely—given a point X on S, we wish to preprocess K so that later, for any desired destination point Y on S, the shortest path from X to Y can be easily and quickly calculated. To simplify the foregoing analysis we will assume, without real loss of generality, that the representation of K is nondegenerate, in the sense that no two faces of K are coplanar. (Otherwise, we can repeatedly combine pairs of adjacent coplanar faces into single faces, until the above property holds for K.) Let n be the number of vertices of K, so that K has also $O(n)$ edges and faces.

DEFINITION. (a) A point $Z \in S$ is called a *ridge* point if there exist at least two shortest paths from X to Z along S (cf. Fig. 4.1.) We denote by R the set of all ridge points in S.

(b) For each point $Z \in S - R$, let $\pi(Z)$ denote the unique shortest path from X to Z along S.

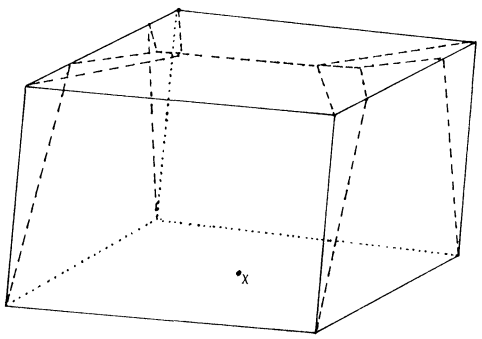


FIG. 4.1. *Ridge points along a convex polyhedron.*

We will first consider the following subproblem: Let $Z \in S$, and suppose that the sequence $\xi = (\xi_1, \cdots, \xi_n)$ of edges of K through which $\pi(Z)$ passes has already been found (we will shortly see in Lemma 4.1 that $\pi(Z)$ cannot cross a vertex of K, so that it meets each of the edges $\xi_1, \cdots, \xi_n$ in an interior point of that edge). Then we wish to find the points of contact between $\pi(Z)$ and each of the edges ξ_i. In the general three-dimensional case treated in the previous section, the solution of this subproblem had been rather complicated, and involved solution of high-degree algebraic equations, mainly because any two adjacent edges ξ_{i-1} and ξ_i in ξ could be skew to one another. However, in the special case which concerns us here this cannot happen, thereby making the problem immediately solvable as follows.

Let π consist of the segments $\pi_0, \cdots, \pi_n$. Recall that Lemma 3.1 implies that for each $i = 1, \cdots, n$, the angles that π_{i-1} and π_i subtend at ξ_i are equal. This suggests the following simple algorithm for the calculation of the points of contact. Since we know the sequence of edges through which $\pi(Z)$ passes, we also know the corresponding sequence $f_0, f_1, \cdots, f_n$ of faces of K through which $\pi(Z)$ passes, where the face f_i contains the two edges ξ_i and ξ_{i+1}, for $i = 1, \cdots, n-1$, and where f_0 contains X and ξ_1 and f_n contains ξ_n and Z.

We then unfold the collection of faces $f_0, \cdots, f_n$ so as to make them all lie in the same plane L. This is done iteratively. That is, initially we place f_0 in L, letting X coincide with the origin. Suppose that we have already unfolded and placed in L all faces up to f_{i-1}. We then unfold f_i about ξ_i until it becomes coplanar with f_{i-1} (but lies on the other side of ξ_i). In practice, we compute for each face f_i the displacement a_i and orientation θ_i defining its position in L relative to some standard and fixed plane representation of this face. We can then compute from a_n and θ_n the position of Z in L. The required path $\pi(Z)$, unfolded to L, is then simply the straight segment XZ. The points of intersection of this segment with the unfolded edges $\xi_1, \cdots, \xi_n$ are then readily determined, and can be easily transformed back to the original polyhedron. For further reference, let us call this process as the *planar unfolding of K relative to* $\xi_1, \cdots, \xi_n$; we also refer to the pair (a_n, θ_n) as the position of f_n in that planar unfolding. (See also Alexandrov [Al] for an analysis of the unfolded planar structure of K; the above observations have also been made by Franklin et al. [FAV], [FA] although they have not developed them into a polynomial-time algorithm).

Hence, as in the general 3-D case, the main problem which needs to be solved is that of calculating the sequence $\xi_1, \cdots, \xi_n$ of edges through which the path $\pi(Y)$ passes.

To this end, we will partition S into at most n vertex-free connected regions, called *peels*, such that the interiors of these regions do not contain any ridge point, and such that for each such region p, the path $\pi(Z)$ to any $Z \in p$ is wholly contained in p. Since p is vertex-free, the sequence of edges through which $\pi(Z)$ passes will be easy to calculate, as will be shown below.

To obtain this partitioning, we begin with analysis of several properties of ridge points.

LEMMA 4.1. *A shortest path $\pi(Z)$ cannot pass through a vertex of K.*

Proof. (We are indebted to R. Pollack for suggesting this simplified proof.) Suppose the contrary, and let U be a vertex of K lying on a shortest path $\pi(Z)$ from X to some $Z \in S$. Suppose first that K is incident to exactly three faces of K. Let A and B be two points on $\pi(Z)$ lying on the two straight subsegments of $\pi(Z)$ adjacent to U, with A lying before U and B after U along that path. Let f_A, f_B be the faces containing A and B respectively, and let f be the third face of K containing U. Instead of the planar unfolding of K in which $\pi(Z)$ is a straight segment (call this the "straight" unfolding of $\pi(Z)$), we construct another unfolding as follows. First unfold faces of

K into the plane as in the planar unfolding of $\pi(Z)$, until U is reached. Then, instead of unfolding f_B past f_A (as is done in the straight unfolding of $\pi(Z)$), unfold first f past f_A along their common edge, and then unfold f_B past f along their common edge; then continue to unfold as in the remainder of the straight unfolding of $\pi(Z)$ (see Fig. 4.2).

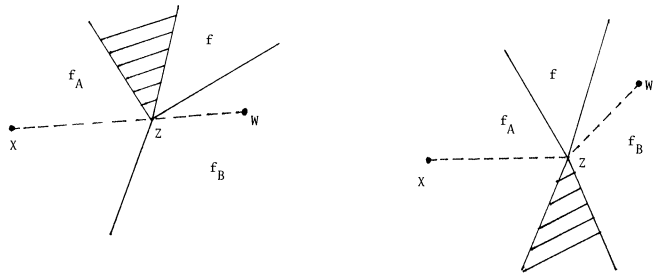


FIG. 4.2. *Proof of Lemma* 4.1.

In this new unfolding the path $\pi(Z)$ appears as a broken line at U, and it is easily seen that $\pi(Z)$ can be shortcut near U, yielding a shorter path which is also contained within this new unfolding. This contradiction establishes the lemma. Essentially the same argument also applies in case U is incident to more than three faces. Q.E.D.

LEMMA 4.2. *A shortest path $\pi(Z)$ cannot pass through a ridge point.*

Proof. Suppose that $\pi(Z)$ does pass through a ridge point W. Then the initial portion π_1 of $\pi(Z)$ up to W is one of several shortest paths from X to W. Let π_2 be another such shortest path, and without loss of generality assume that π_1 and π_2 are transversal to one another at W (this will be the case if we choose W to be the first ridge point along $\pi(Z)$, which is always well defined, because the set of ridge points along $\pi(Z)$ is closed, as can be easily verified). Let π' denote the path obtained by replacing π_1 by π_2 in $\pi(Z)$; note that π' is also a shortest path to Z. However, if W is interior to some face of K, then π' cannot be a shortest path to W, since it bends at an interior point of a face of K. On the other hand, if W lies on an edge of K, then π' cannot be a shortest path to W since it forms unequal angles with the edge containing W (note that W cannot be a vertex by Lemma 4.1). Q.E.D.

LEMMA 4.3. *The set R of ridge points is the union of finitely many straight segments.*

Proof. With any point Z of R we can associate the face of K containing Z, and the two sequences of edges of K through which the two shortest paths from X to Z pass. Since there are only finitely many values that each of these three parameters can assume (because a shortest path to a point cannot pass through an edge of K more than once), it suffices to show that the locus of all ridge points Z lying on a fixed face of K, for which the two shortest paths from X to Z pass through two fixed sequences of edges of K, is a straight segment.

Therefore let f be a fixed face of K, and let $\boldsymbol{\xi} = (\xi_1, \cdots, \xi_n)$, $\boldsymbol{\eta} = (\eta_1, \cdots, \eta_m)$ be two fixed sequences of edges of K, such that any two adjacent edges in either sequence lie on a common face, and such that ξ_1 and η_1 bound the face containing X, while ξ_n and η_m bound f. Assume that there exists at least one point $Z \in R$ having these values as its associated parameters. Let (a_ξ, θ_ξ), (a_η, θ_η) be the positions of f in the planar unfoldings of K relative to the two sequences $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ respectively, where a_ξ and a_η both give the position of Z in the corresponding planar unfolding. Let $W \in R$ be another point having the same parameters as Z, and write $ZW = \mathbf{w}$ in the standard Cartesian representation of f. Then we have (where R_ξ (resp. R_η) denotes the rotation

of the plane by the angle θ_ξ (resp. θ_η)):

$$|a_\xi + R_\xi \mathbf{w}| = |a_\eta + R_\eta \mathbf{w}|$$

which states the equality of the lengths of the two shortest paths from X to W, constrained to pass respectively through the sequences $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ of edges of K. Squaring out the above equation, and using the fact that $|R_\xi \mathbf{w}| = |R_\eta \mathbf{w}|$, and that $|a_\xi| = |a_\eta|$, we obtain

$$a_\xi \cdot R_\xi \mathbf{w} = a_\eta \cdot R_\eta \mathbf{w}$$

which is the equation of a straight line passing through Z. Q.E.D.

Remarks. (1) The preceding equation is nondegenerate provided that $a_\xi R_\xi \neq a_\eta R_\eta$. But if these two quantities were equal, then the two corresponding shortest paths to W would be such that their terminal segments along f coincide. This however implies that these two paths pass through another ridge point (e.g. any interior point of these coincident segments), which is impossible, by Lemma 4.2.

(2) The equation defining the ridge segment given above can be rewritten as

$$(a_\xi R_\xi - a_\eta R_\eta) \cdot \mathbf{w} = 0$$

and thus be given the following geometric interpretation:

Perform the two planar unfoldings relative to $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$. First rotate each unfolding in clockwise direction by the respective angle θ_ξ, θ_η (note that these rotations cause the final face f in each of the unfoldings to have the same orientation as the standard representation of f). Next translate (without rotating) the two resulting copies of f so that they both coincide with the standard representation of f. Note that this will have moved the starting point X to the planar positions $X_\xi = a_\xi R_\xi$ and $X_\eta = a_\eta R_\eta$ respectively. Then the required ridge segment (in the standard representation of f) is contained in the perpendicular bisector to the segment $X_\xi X_\eta$ (see Fig. 4.3).

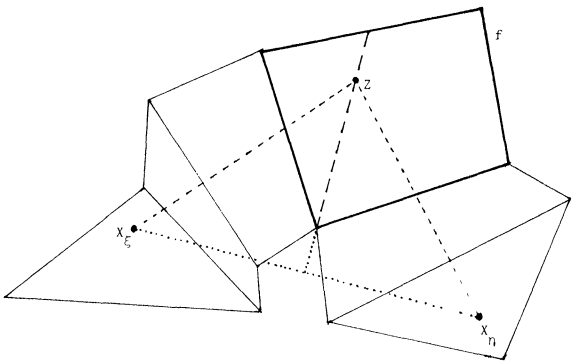


FIG. 4.3. *Construction of ridge segments.*

(3) This construction of ridge segments makes it clear that if Z lies on a ridge segment e defined by two edge sequences $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ and Z is not a vertex of K, then for points W lying on the line containing e in a sufficiently small neighborhood of Z the following properties hold: There exist two geodesic (but not necessarily shortest) paths to W having equal lengths and passing respectively through the sequences $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$; moreover the starting orientation of each of these paths is a monotone and continuous function of W. (Note that these geodesics will be shortest paths for W lying on at least one side of Z.)

To continue our analysis of the structure of R, we first introduce the following notation.

DEFINITION. (a) For each planar orientation θ define a polygonal path $p = p(\theta)$ from X along S as follows: p starts at X in the direction of θ (relative to the standard representation of the face f_0 containing X). Whenever p reaches an edge e of K it bends over it to the adjacent face so that the two segments of p adjacent to e form with it equal angles (as in Lemma 3.1). p will terminate as soon as it reaches a vertex of K.

(b) For each orientation θ and each $Z \in p(\theta)$ we define $p(\theta, Z)$ to be the initial portion of $p(\theta)$ between X and Z.

With some exceptions noted below, we will consider the path $p(\theta)$ to terminate after the first time it reaches either a vertex of K or a ridge point. (Note that this must occur for each θ, because otherwise either the length of p would increase without bound, while p still being the shortest path to any of its points, which is plainly impossible, or else p would reach X again, in which case it is clear that p contains a ridge point.) Let $r(\theta)$ denote the endpoint of $p(\theta)$.

LEMMA 4.4. (a) *Let Z_n be a sequence of points on S converging to some $Z \in S$ as $n \to \infty$. For each $n \geqq 1$ let π_n be a shortest path from X to Z_n, and suppose that the paths π_n converge in the Hausdorff topology of sets to a path π. Then π is a shortest path to Z.*

(b) *The function $r(\theta)$ is continuous.*

Proof. (a) Suppose the contrary, and let π' be a path from X to Z which is shorter than π. Since the length of π is the limit of the lengths of the paths π_n, it follows that if n is sufficiently large, by appending to π' a short path connecting Z to Z_n, we can obtain a path to Z_n which is shorter than π_n, a contradiction.

(b) Let $\theta_n \to \theta$, and suppose that $Z_n = r(\theta_n)$ converges to some point Z, and that for each n the point $r(\theta_n)$ is a ridge point (if the latter property cannot be achieved, then from a certain n on, $r(\theta_n)$ is a vertex, which implies that the sequence θ_n is constant too, in which case there is nothing to prove). It follows that for each n there exists another orientation ϕ_n such that $r(\theta_n) = r(\phi_n) = Z_n$. Passing to subsequences if necessary, we can assume that ϕ_n converge to some ϕ, and that the paths $p(\theta_n, Z_n)$ converge to $p(\theta, Z)$ and similarly the paths $p(\phi_n, Z_n)$ converge to $p(\phi, Z)$. Let ξ^n, η^n be the two sequences of edges of K through which the two paths $p(\theta_n, Z_n)$ and $p(\phi_n, Z_n)$ pass. Passing again to a subsequence if necessary, we can assume that the sequences ξ^n and η^n are both constant (necessarily distinct from one another). Two cases can arise:

(i) If $\theta \neq \phi$ then by (a) the two distinct paths $p(\theta, Z)$ and $p(\phi, Z)$ are shortest paths to Z, so that Z is a ridge point, and $r(\theta) = Z$, because $p(\theta, Z)$ does not pass through a vertex of K or a ridge point (other than Z).

(ii) If $\theta = \phi$ then Z must be a vertex of K. Indeed, if this were not the case, then (by Lemma 4.1) there would exist a sufficiently small neighborhood U of $p(\theta, Z)$ which contains no vertex of K. But then for n sufficiently large the two paths $p(\theta_n, Z_n)$, $p(\phi_n, Z_n)$ would be wholly contained in U, and thus will cross the same sequence of edges of K, contrary to assumption. The same argument as in (i) above now implies that $Z = r(\theta)$.

This proves part (b) of the lemma. Q.E.D.

COROLLARY. *The set R^* consisting of all vertices of K and ridge points is a closed connected set.*

Proof. It suffices to show that the map r defined above is onto R^*, for then R^* will be the continuous image of the unit circle of starting orientations θ. To show that r is onto, let $Z \in R^*$, and let $\pi(Z)$ be one of the shortest paths to Z. Then, arguing as

in the preceding proof, $Z = r(\theta)$, where θ is the starting orientation of $\pi(Z)$ (as follows from Lemmas 4.1 and 4.2 and from the definition of r). Q.E.D.

In view of Lemma 4.3, the set R^* can be regarded as a graph whose edges are (portions of) the straight segments yielded by the proof of Lemma 4.3. The vertices of this graph are either vertices of K, or points at which such a segment intersects an edge of K, or points at which two such segments intersect. The *degree* of each vertex u of R^* is defined to be the number of edges of R^* incident to u. More information on the structure of the vertices of R is obtained from the following lemmas.

LEMMA 4.5. (a) *Each vertex of R^* having degree 1 is a vertex of K.*

(b) *Each vertex of R^* having degree 2 is an intersection of R with the interior of some edge of K.*

(c) *R^* does not contain any closed path.*

Proof. (a) Suppose that Z is a vertex of R^* of degree 1 which is not a vertex of K. Then Z must be a ridge point. Suppose first that there are exactly two shortest paths π and π' reaching Z. By Lemma 4.3 Z lies on a segment e which is determined by the two sequences $\boldsymbol{\xi}, \boldsymbol{\eta}$ of edges of K through which π and π' respectively pass. If Z is an endpoint of e (and of no other ridge segment) then for each point W on the line containing e lying near Z on the other side of e there must exist a unique shortest path $\pi(W)$ to W. However, this path passes through a sequence $\boldsymbol{\zeta}$ of edges of K which is distinct from both $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$. For suppose that $\pi(W)$ passes through the sequence $\boldsymbol{\xi}$; then by definition of e there would exist another path from X to W having the same length as $\pi(W)$ and passing through the sequence $\boldsymbol{\eta}$ of edges (cf. Remark (3) following Lemma 4.3). Hence, letting $W \to Z$ along this line, we would obtain a third shortest path to Z passing through the sequence $\boldsymbol{\zeta}$, contrary to assumption. This argument can be generalized to the case in which more than two shortest paths reach Z, thus proving (a).

(b) Suppose the contrary, and let $Z \in R$ be an interior point of some face f of K, which is a common endpoint of exactly two ridge segments e_1 and e_2. Let ξ^i, η^i be two distinct pairs of sequences of edges of K which define e_i, $i = 1, 2$. Suppose for definiteness that $\xi^1, \eta^1 \neq \xi^2$. Then there are at least three shortest paths reaching Z and passing respectively through the sequences ξ^1, ξ^2 and η^1 (see Fig. 4.4). Consider the perpendicular bisector l to the segment $X_{\xi^1}X_{\xi^2}$ as in the ridge construction procedure described above. l is distinct from the lines containing e_1 and e_2, it passes through Z, and contains no ridge point in a small neighborhood of Z, except Z itself. Thus for each $W \in l$ near Z there exists a unique shortest path $\pi(W)$ passing through a sequence $\zeta(W)$ of edges of K which is distinct from ξ^i, η^i, $i = 1, 2$, as can be seen from Remark (3) following Lemma 4.3. But then, using arguments similar to those used in the preceding paragraph, we obtain a contradiction which concludes the proof of (b).

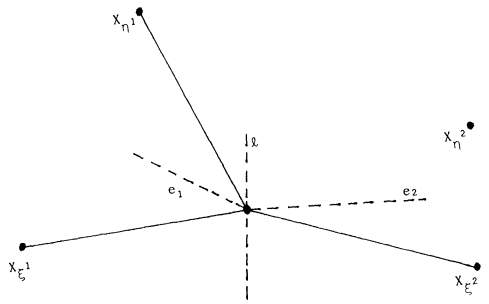


FIG. 4.4. *Illustrating the proof that ridge segments cannot "bend" at a point interior to a face.*

(c) Suppose the contrary, and let C be a simple closed path in R^*. By the Jordan curve theorem, C divides S into two disjoint regions S_1, S_2, so that one of them, say S_1 contains X. Let Z be an arbitrary point in S_2. Then the shortest path $\pi(Z)$ from X to Z will have to intersect C, which contradicts either Lemma 4.1 or Lemma 4.2. Q.E.D.

COROLLARY. *R^* is a tree having (some of) the vertices of K as leaves.*

Proof. Immediate by the preceding lemma.

Suppose that we have managed to construct R^*. We can then use it for calculating shortest paths from X to arbitrary points $Y \in S$ as follows: Define Q to be the union of R with the shortest paths from X to every vertex of K. Q partitions S into disjoint connected regions (which we call *peels*), the interiors of which do not contain any vertex or ridge point. Let p be one of these peels. Using arguments similar to the preceding ones, we can show that the (unique) shortest path $\pi(Z)$ to any interior point Z of p is wholly contained in p. Also, there exists an open interval $I = I(p)$ of orientations such that $p = \bigcup_{\theta \in I} p(\theta, r(\theta))$, and such that for each $\theta \in I$, the endpoint $r(\theta)$ does not reach a vertex. Moreover, the two end orientations θ_1 and θ_2 of I are such that both $p(\theta_1)$ and $p(\theta_2)$ terminate at a vertex of K (otherwise I and p could be increased with all the other properties of p still valid). Finally, the intervals $I(p)$ are pairwise disjoint, and the union of their closures cover the whole angular space $[0, 2\pi]$.

Let J_p denote the set of edges of K which intersect p. It follows that the edges J_p can be ordered by their adjacency in p. That is, we say that e precede e' in J_p if there exists a shortest path in p passing first through e and later through e'. To see that this order is well defined, and to gain more insight into the structure of peels, we have the following lemmas.

LEMMA 4.6. *Suppose that a peel p is unfolded into the plane by unfolding each of the geodesic paths $p(\theta)$ in p into the ray from X at orientation θ. Then the resulting image of p is convex.*

Proof. Let P be the unfolded image of p. P is plainly contained in the angular sector consisting of rays at orientations $\theta \in (\theta_1, \theta_2)$ and is star shaped with respect to X. Hence if P is not convex there must exist a concave corner Z on its boundary but not on either of the two rays $\theta = \theta_1$ or $\theta = \theta_2$. Hence Z is a ridge point. Let the two (ridge) edges meeting at Z be e_1 and e_2, and let l be the line containing e_1. Then points $W \in l$ near Z but on the other side of e_1 will be inside P, so that the shortest path to such a W is unique, is contained in p, and passes through the same sequence of edges as the path which reaches Z from within p. But then Remark (3) following the construction of ridge segments given above implies that there also exists another geodesic path to W whose length is equal to the length of the geodesic path to W through p. This contradiction proves the assertion. Q.E.D.

LEMMA 4.7. *Let p be a peel and let f be a face of K such that $p \cap f \neq \emptyset$. Then there exists a unique sequence ξ of edges of K such that for each $Z \in p \cap f$ the shortest path $\pi(Z)$ to Z passes through the sequence ξ.*

Proof. Suppose the contrary, and let Z_1, Z_2 be two points in $p \cap f$ for which the sequences ξ, η of edges of K through which the paths $\pi(Z_1)$, $\pi(Z_2)$ respectively pass are distinct. By Lemma 4.6 the segment Z_1Z_2 is contained in $p \cap f$, and by continuity there would have to exist a ridge point $Z \in Z_1Z_2$ (hence in p), a contradiction which proves our claim. Q.E.D.

Hence each peel p defines at most $O(n)$ distinct sequences of edges of K, one for each face of K which p intersects, such that the shortest path to any $Z \in p$ passes through one of these sequences. These sequences can be arranged in an auxiliary tree

associated with p so that ξ is an ancestor of η in this tree if and only if ξ is a prefix of η. Given a peel p, it is straightforward to compute all these sequences, and we omit details of this easy construction.

These observations lead to the following proposition.

PROPOSITION 4.8. (a) *There are n peels.*

(b) *There are $O(n^2)$ edges in R^*.*

Proof. (a) Immediate from the preceding considerations.

(b) We will show that each face of K contains $O(n)$ segments of R^*. Let f be a face of K, and let $\xi^1, \cdots, \xi^l$ denote the collection of all sequences of edges of K traversed by shortest paths from X to points on f. Each such sequence corresponds to a unique peel, so that, by Lemma 4.7, $t \leqq n$. For each such sequence ξ^i let $(a_{\xi^i}, \theta_{\xi^i})$ be the parameters describing the position of f in the planar unfolding corresponding to the sequence ξ^i, and let $X_{\xi^i} = a_{\xi^i} R_{\theta_{\xi^i}}$ denote the planar position of the point X when this unfolding is moved so that f coincides with its standard planar representation.

In a manner quite similar to the analysis of Voronoi diagrams [Sh] we can define the *dual* graph of $R^* \cap f$ to consist of the points X_{ξ^i} as its nodes, such that X_{ξ^i} and X_{ξ^j} are connected by an edge if the two sequences ξ^i and ξ^j define an edge of $R^* \cap f$. As in [Sh], one can show that this dual graph is planar by embedding it into the plane as follows. Map each node X_{ξ^i} to its plane position, and map each edge (X_{ξ^i}, X_{ξ^j}) to the union of two segments connecting respectively X_{ξ^i}, X_{ξ^j} to a point on the common edge of $R^* \cap f$ which these two sequences define. Since this graph is planar, and since Lemma 4.5(b) implies that it has no multiple edges, it follows that it has at most $O(n)$ edges, thus proving (b). (A similar connection between the partitioning of S by the set of ridge points and Voronoi diagrams is also noted in [FAV].) Q.E.D.

Remark. The slicing of the surface S of K into peels has the following geometric implication. If we apply the planar unfolding procedure to all the peels, from the same planar position of X, we obtain a planar layout which we denote as $U(K)$ which has the following properties:

(1) No two unfolded peels overlap in the plane (except at points lying on an unfolded geodesic path connecting X to a vertex of K). Indeed, if Z_1 is a point in one peel and Z_2 is a point in another, then the segments XZ_1 and XZ_2 have different orientations, unless the exceptional condition noted above holds.

(2) $U(K)$ is star shaped with respect to X.

(3) If $V(K)$ is another planar unfolding of the whole surface of K having properties (1) and (2), then the smallest disc about X containing $U(K)$ has radius smaller than or equal to that of the corresponding such disc containing $V(K)$.

The partitioning of S into peels by Q enables us to find the shortest path from X to any $Y \in S$ in the following simple manner:

(a) Find the peel p containing Y. For simplicity assume that Y is an interior point of p.

(b) Find the sequence ξ of edges in J_p through which $\pi(Y)$ passes.

(c) Apply the planar unfolding procedure relative to ξ, as described above, to obtain the required shortest path $\pi(Y)$.

Concerning the complexity of the path-finding procedure just outlined, assume that the face f of K containing Y is specified too. Each such f is partitioned by Q into at most $O(n)$ subregions. It is then straightforward to locate in time $O(n)$ the subregion of f containing Y, and thus the peel p containing Y (for example, pass a straight line through Y and compute its intersections with $Q \cap f$, from which the region containing Y can be readily calculated). Given p, we obtain ξ in $O(n)$ time from the precalculated associated tree, and the planar unfolding procedure can then be applied to ξ also in time $O(n)$.

COROLLARY. *After preprocessing, the shortest path from a fixed point X to any point $Y \in S$ can be computed in time $O(n)$, using a data-structure whose size is $O(n^2)$.*

The main problem that we still face is that of computing Q. This will be done in the following section.

5. The peels construction algorithm. In this section we present an algorithm for partitioning the surface of K into peels, which runs in time $O(n^3 \log n)$. The algorithm constructs a tree T, called the *slice tree* as follows. Let us define a slice σ in terms of two "starting orientations" $\theta_1 < \theta_2$ and a "terminal face" f, which are assumed to have the following properties:

(i) Each of $p(\theta_1)$ and $p(\theta_2)$ reaches a vertex of K either before reaching f or on f itself.

(ii) For each $\theta \in (\theta_1, \theta_2)$ the path $p(\theta)$ reaches f and does not meet a vertex before its first exit from f.

(iii) For each $\theta \in (\theta_1, \theta_2)$ let $Z_\theta \in p(\theta)$ be the point at which this path leaves f (for the first time); then all the points Z_θ lie on the same edge of f (which we call the *terminal edge* of σ).

The corresponding slice, denoted as $\sigma(\theta_1, \theta_2, f)$ is defined to be the union of all paths $p(\theta, Z_\theta)$, for $\theta \in (\theta_1, \theta_2)$. Note that the set of points at which geodesic paths in $\sigma(\theta_1, \theta_2, f)$ enter f is also a subsegment of some edge of f. Note also that all the geodesic paths $p(\theta, Z_\theta)$ in this slice pass through the same sequence of edges of K, and that if σ is unfolded into the plane along this sequence of edges, its image is a triangle bounded by the two rays emerging from the origin (X) at orientations θ_1 and θ_2 and by the terminal edge of σ (cf. Fig. 5.1).

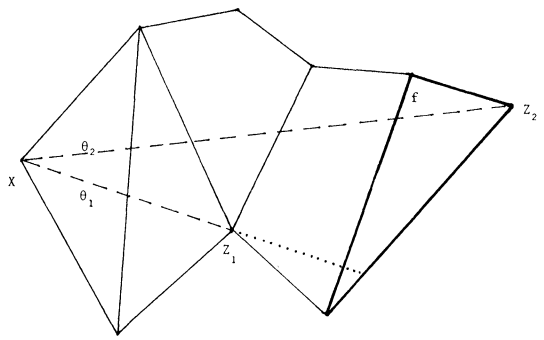


FIG. 5.1. *Planar layout of a slice.*

A slice is roughly meant to correspond to some portion of a peel, which is "encoded" implicitly by the range of starting orientations and the terminal face f. The correspondence is such that for each "true" slice $\sigma = \sigma(\theta_1, \theta_2, f)$ the sequence of edges that it meets is one of the sequences associated with some peel p (in the manner described in the preceding section), and the range (θ_1, θ_2) is contained in the corresponding range $I(p)$. Note that the above definition does not ensure that the geodesics $p(\theta)$ within σ for $\theta \in (\theta_1, \theta_2)$ are actually shortest paths. However, the algorithm to be described below will make sure that each slice σ that it will construct will correspond to some peel p in the manner just described, so that at least one geodesic path passing through the sequence of edges defined by σ is indeed a shortest path.

The slice tree that our algorithm will construct is defined as follows. Each node of T except for its root is a slice. The root is a dummy slice and its sons are the slices $\sigma(\theta_i, \theta_{i+1}, f_0)$, $i = 1, \cdots, s$, where f_0 is the initial face of K containing X and where $\theta_1, \cdots, \theta_s$ are the orientations of the segments connecting X to the vertices of f_0,

ordered in angular order about X. Let $\sigma = \sigma(\theta_1, \theta_2, f)$ be a node of T. Its sons are obtained by extending σ one face past f. Specifically, let e be the terminal edge of σ, i.e. the edge of f containing all the terminal points Z_θ for $\theta \in (\theta_1, \theta_2)$, and let f' be the face of K adjacent to f at e. If the geodesic paths in σ have already passed through f' (before reaching f) then σ is called a *terminal* slice, and remains a leaf of T. Otherwise we extend all paths $p(\theta)$ for $\theta \in (\theta_1, \theta_2)$ into f', and let τ denote the set of points at which these paths leave f'. Plainly τ is a connected portion of the boundary of f' and thus is a union of portions $\tau_1, \cdots, \tau_q$ of edges of f' (actually each τ_j for $1 < j < q$ is a full edge). For each $1 \leqq j < q$ let $\psi_j \in (\theta_1, \theta_2)$ denote the orientation of the geodesic path in σ which reaches the vertex at which τ_j and τ_{j+1} meet. Put also $\psi_0 = \theta_1$ and $\psi_q = \theta_2$. Then slices that are candidates for the sons of σ in T are the slices $\sigma(\psi_{i-1}, \psi_i, f')$, $i = 1, \cdots, q$ (see Fig. 5.2).

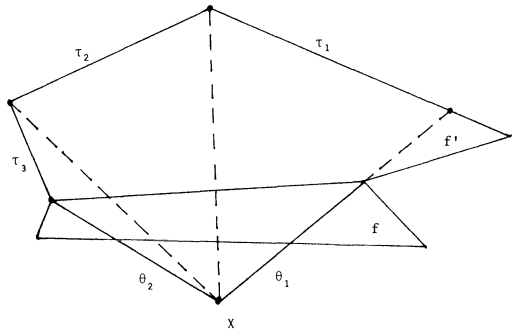


FIG. 5.2. *Extending a slice past an edge.*

The problem with extending the tree in this somewhat uncontrolled manner is that this might cause us to add to the tree slices for which the sequences of edges of K which they define may be such that no shortest path passes through it. Note that the number of *true* slices (i.e. slices for which there exists at least one shortest path passing through the sequence of edges which they define) is $O(n^2)$, in virtue of Proposition 4.8, but that we have no obvious way to estimate the total number of possible "false" slices that might be introduced by the construction in the preceding paragraph.

We therefore design our algorithm so that it always adds only true slices to the tree T. More specifically, the algorithm will maintain the following two invariants:

(a) The tree constructed by the algorithm contains only true slices (in the above sense).

(b) With each slice $\sigma = \sigma(\theta_1, \theta_2, f)$ we associate a *terminal portion* σ_f which is a polygonal subregion of $\sigma \cap f$. The terminal portions of slices in the tree will be pairwise disjoint and each terminal slice portion will contain the set of all points Z in f for which the shortest path to Z within σ is shorter than that any path to Z contained with one of the slices present in T.

To maintain the invariant (b), the algorithm will have to store in the tree that it constructs additional information concerning the terminal portion of each slice. Terminal slice portions that are proper subsets of their corresponding slice portions $\sigma \cap f$ can arise due to the overlapping of such a $\sigma \cap f$ with other slices already present in the tree (and reaching the same terminal face). This implies that as the tree is being constructed, these trimmed portions may be trimmed again and again as new slices are added to the tree until they reach their final value at the end of the algorithm. We will show below that proper assembly of the terminal portions of the slices in the final tree gives us all the peels that we seek.

Each iteration step of the algorithm adds a new slice to the tree. Let T_k denote the slice-tree after the kth iteration of the algorithm. For each starting orientation θ let Z_θ denote the farthest point on $p(\theta)$ such that $p(\theta, Z_\theta)$ is wholly contained within a slice of T_k. Define the *front* $F(T_k)$ of T_k to be the set of all points Z_θ which lie on edges of K. Let $W(T_k)$ denote a point $Z_\theta \in F(T_k)$ for which the length of $p(\theta, Z_\theta)$ is smallest, and let $m(T_k)$ denote this smallest length.

Initially, as defined above, T_0 consists of all the slices contained in the face f_0 containing X, all being children of a common dummy root.

At the $(k+1)$st step, the algorithm picks $Z_\theta = W(T_k)$, and obtains the slice σ in T_k containing Z_θ. Suppose for the moment that Z_θ lies in just one such slice. Let f be the terminal face of σ, let e be the edge of K containing Z_θ, and let f' be the other face of K adjacent to f at e.

Let U_θ denote the farthest point along $p(\theta) \cap f'$ (lying on another edge of f'), and let σ' be the slice which extends σ past e and which contains $p(\theta, U_\theta)$ (again we suppose for the moment that only one such slice exists). Then we have the next lemma.

LEMMA 5.1. *σ' is a true slice, in the sense that there exists at least one shortest path passing through the sequence of edges which σ' defines.*

Proof. What we have to show is the existence of a starting orientation θ' within the range of starting orientations of σ' and a point $U \in p(\theta') \cap f'$ for which $p(\theta', U)$ is a shortest path to U. We take θ' to be θ, and U to be a point in $p(\theta) \cap f'$ sufficiently near Z_θ. Suppose to the contrary that $p(\theta, U)$ is not one of the shortest paths to U. Let ψ be the starting orientation of the shortest path to U. Two cases can arise:

(i) The point U belongs to some slice in T_k. But if this is the case for all points U sufficiently close to Z_θ, then Z_θ must be a ridge point and lie on the boundary of more than one slice in T_k, contrary to assumption.

(ii) Hence U does not belong to any slice in T_k, so that $p(\psi)$ must reach the front $F(T_k)$ at some point V before reaching U. But then the length of $p(\psi, V)$ is at least that of $p(\theta, Z)$, so that if U is chosen sufficiently near Z_θ the path $p(\psi, U)$ cannot be a shortest path to U. (To show this note that the additional length of $p(\theta)$ between Z_θ and U will be strictly smaller than the length of $p(\psi)$ between V and U if one chooses U sufficiently near Z_θ, for otherwise an impossible situation as in case (i) above would occur.) This proves the lemma. Q.E.D.

The cases in which Z_θ belongs to two slices in T_k, or in which there are two slices which extend σ and contain $p(\theta, U_\theta)$ deserves a slightly modified treatment. It can be shown that, except for some rare cases which can be detected by a purely local analysis of the structure of geodesic paths near Z_θ, one can extend either of the slices containing Z_θ in the first case, or use any of the extended slices obtained in the second case, and Lemma 5.1 will still hold for this extended new slice, although its proof will have to be somewhat modified.

The algorithm will then add the slice σ' as a son of σ to the slice tree, thus maintaining property (a). To maintain (b) the algorithm will have to trim the terminal portion of σ' to keep it disjoint from the terminal portions of other slices reaching the same face, and possibly also trim these other terminal portions.

To understand how such a trimming is to be done, suppose first that we are given just two slices, both reaching the same terminal face f and overlapping one another on f. Then the following basic procedure is applicable.

Slice trimming procedure. Let $\sigma_1 = \sigma(\theta_1, \theta_1', f)$, $\sigma_2 = \sigma(\theta_2, \theta_2', f)$ be two slices with the same terminal face f. Apply the planar unfolding procedure to σ, σ' to obtain planar triangular layouts of these two slices. Move these two layouts in the plane so that the two copies of the face f in these layouts coincide with the standard plane

representation of f. Let $X_{\sigma_1}, X_{\sigma_2}$ be the positions of the point X in these two new layouts, and let l be the perpendicular bisector of $X_{\sigma_1}X_{\sigma_2}$. Let f_{12} (resp. f_{21}) denote the portion of f lying on the X_{σ_1}-side (resp. the X_{σ_2}-side) of l. We then replace $\sigma_1 \cap f$ (resp. $\sigma_2 \cap f$) by $\sigma_1 \cap f - \sigma_1 \cap \sigma_2 \cap f_{21}$ (resp. $\sigma_2 \cap f - \sigma_1 \cap \sigma_2 \cap f_{12}$). See Fig. 5.3.

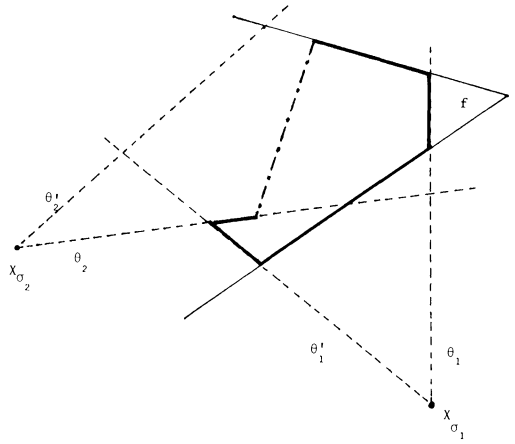


FIG. 5.3. *The slice trimming procedure.*

In other words, for points Z which can be reached by two distinct geodesic paths, lying in two different slices, the slice trimming procedure determines which of these two paths is shorter, and removes the point Z from the other slice.

Let us consider in more detail the computational aspects of this procedure. As the slice tree is being built, we can store with each slice σ the position X_σ of the point X in the planar lyout of σ in which f lies in its standard plane position (X_σ will be represented by quantities a_σ, θ_σ as in the preceding section, which can be easily updated as we extend σ to an adjacent face).

Given two slices $\sigma_1 = \sigma(\theta_1, \theta_1', f)$ and $\sigma_2 = \sigma(\theta_2, \theta_2', f)$ meeting at the same terminal face f, identify σ_j with its specific planar unfolding used in the slice trimming procedure above, $j = 1, 2$. Then it is easily seen (cf. Fig. 5.3) that $\sigma_1 - \sigma_1 \cap \sigma_2 \cap f_{21}$ is a bounded polygonal region which is star shaped with respect to X_{σ_1}, and which is bounded by at most six segments, including the two rays emanating from X_{σ_1}. We regard the boundary of this region, excluding the two rays from X_{σ_1}, as the graph of a function expressing the distance ρ from X_{σ_1} as a function of the orientation $\theta \in (\theta_1, \theta_1')$ and write it as $\rho_{\sigma_1;\sigma_2}(\theta)$.

Now suppose that at a certain time during its construction, the tree T contains slices $\sigma_1, \cdots, \sigma_t$ all reaching the same terminal face f. Then to enforce property (b), the terminal portion of each σ_i will have to be replaced by $\nu(\sigma_i) \cap f$, where

$$(*) \qquad \nu(\sigma_i) = \sigma_i - \bigcup_{j \neq i} \sigma_i \cap \sigma_j \cap f_{ji}.$$

Note that $\nu(\sigma_i)$ is a star-shaped region (with respect to X_{σ_i}) whose boundary, except for the two rays emanating from X_{σ_i}, is represented by the function

$$\rho_{\sigma_i}(\theta) = \min_{j \neq i} \rho_{\sigma_i;\sigma_j}(\theta), \qquad \theta \in (\theta_i, \theta_i').$$

In other words, ρ_{σ_i} is the "lower envelope" of $t-1$ polygonal lines, each consisting of at most four segments, as these lines are viewed from the point X_{σ_i}.

We claim that the following property holds.

LEMMA 5.2. *If ρ_{σ_i} consists of t_i segments, then $\sum_i t_i = O(t)$.*

Proof. To prove this claim, we proceed in a manner quite similar to the estimate of the size of Voronoi diagrams used in [Sh]. That is, we define an undirected graph G whose nodes are the points X_{σ_i}, and in which an edge connects X_{σ_i} to X_{σ_j} if the trimmed portions of σ_i and σ_j have an edge in common, and if this edge is not contained in any of the four rays emanating from X_{σ_i} and X_{σ_j}. It is easy to show that G is a planar graph by mapping each edge $(X_{\sigma_i}, X_{\sigma_j})$ of G to the union of two segments connecting X_{σ_i} and X_{σ_j} to a point on the corresponding common edge. Moreover, the graph G has no multiple edges, a fact which can be established as in the proof of Lemma 4.5. It therefore follows by Euler's formula that G has $O(t)$ edges. Finally, each of the edges obtained by the trimming process which is not recorded in G lies on a ray bounding some slice σ_i, and plainly there can be at most $O(t)$ such edges. These observations establish our claim. Q.E.D.

Now by the results of the preceding section, $t \leqq n$, since the total number of "true" slices reaching the same face is at most n. This means that the total number of edges separating trimmed terminal portions of slices within a given face is at most $O(n)$ during each stage of the algorithm. Moreover, as we add a new slice σ to the tree we need to update the function ρ_{σ_i} for all slices σ_i reaching the same terminal face f as σ. Suppose as before that ρ_{σ_i} consists of t_i segments. Then its updated value due to the appearance of σ can be computed in a straightforward manner using (*) in time $O(t_i)$, so that updating of all these functions can be accomplished in time $O(n)$. Moreover, each new edge added to any of these functions, and only such edges, will also appear in the graph of the new function ρ_σ, and it is an easy matter to assemble all these edges and thus obtain ρ_σ in time $O(n)$.

Remark. Lemma 4.6 implies that the final trimmed value of a slice is convex (when properly unfolded). The procedure sketched above might however temporarily lead to nonconvex trimmed slices. We do not know whether nonconvex intermediate slices can actually be obtained.

We therefore conclude that one can maintain property (b) in time $O(n)$ per each step of the algorithm.

The algorithm also needs to update the values of $W(T_k)$ and $m(T_k)$ in view of the addition of σ' to the tree. Note that the new front of the tree is obtained from the previous front by deletion of the "entering" segment of σ', by addition of the terminal edge of σ', and by possibly shortening other edges of the front due to the trimming procedure described above. Note that the deletion of the entering edge of σ' will in general have split (a portion of) the terminal edge of σ into two subsegments which still belong to the front, and that similar splits or replacements of edges will result from the trimming process. This suggests that we represent the front of the tree as a union of subsegments of edges of K. For each segment e in the front we can easily compute the point $W(e)$ on e for which the geodesic path to $W(e)$ through the slice bounded by e is shorter than that to any other point on e, and also the length $m(e)$ of this path. We then maintain a priority queue containing all the relevant segments e constituting the front, ordered by the value of $m(e)$. At each addition of a new slice σ' to the tree we delete from the priority queue the terminal segment of the ancestor σ of σ', add back the two subsegments of this segment still in the front, and add the terminal edge of σ'. Since the trimming step can result in a new terminal slice portion having no terminal edge, and since the terminal edges of existing slices may also be trimmed by that procedure, the priority queue of front edges will have to be updated in an appropriate manner. Since the incremental trimming procedure described above will produce at each step only $O(n)$ slice-bounding segments, it follows that at most

$O(n)$ updates of the priority queue will be required at each stage. Once these updates are performed, the updated values $W(T_k)$ and $m(T_k)$ are easily available and can be retrieved in the next iteration of the algorithm.

The algorithm terminates when the priority queue representing the front of the tree becomes empty, i.e. when the front itself becomes empty.

When this happens, we still need a final phase that will construct the peels of K from the slice tree. To this end let $\sigma = \sigma(\theta_1, \theta_2, f)$ be a leaf of T. The final trimmed value of σ is defined as

$$\mu(\sigma) \equiv \bigcup_{\sigma'} \sigma \cap \tau(\sigma')$$

where σ' ranges over all slices on the path in T to σ, and where $\tau(\sigma')$ denotes the final trimmed terminal portion of σ'. That is, we collect all trimmed terminal portions of slices lying along the path to σ in T, but restrict each such portion to the wedge between the two starting orientations defining σ.

Note that $\mu(\sigma)$ need not be a full peel. In fact, a necessary and sufficient condition for $\mu(\sigma)$ to be a peel is that the two bounding geodesics $p(\theta_1)$ and $p(\theta_2)$ are not trimmed, and still reach vertices of K within (the closure of) $\mu(\sigma)$. If, say, $p(\theta_1)$ has been trimmed, let σ_1 be the slice whose range of starting orientations is adjacent to that of σ at θ_1. Then it is easy to see that the peel containing $\mu(\sigma)$ also contains $\mu(\sigma_1)$. These observations make it clear that we can construct all peels by a depth-first traversal of T, visiting sons of a slice in, say, counterclockwise order of their ranges of starting orientations.

To estimate the time required by the algorithm we note that the maintenance of property (b) is the costliest part of the algorithm, in which the updating of the priority queue representing the front of T may require $O(n \log n)$ time for each step (the trimming procedure itself requiring only $O(n)$ time). Since the algorithm adds at most $O(n^2)$ slices to the tree, it follows that the algorithm will run in time $O(n^3 \log n)$.

The correctness of the algorithm follows from the following considerations. First note that the peels as constructed by the algorithm are pairwise disjoint by construction. We also claim that they cover the whole surface of K. To show this, it suffices to prove that every true slice is constructed by the algorithm. Indeed, if this latter property is known to hold, then for each point Z on the surface of K let $p(\theta, Z_\theta)$ be a shortest path to Z. The sequence of edges and faces through which this path passes defines a true slice σ which will appear in the final tree T. It is easy to verify that Z will belong to the final trimmed terminal portion of σ.

To see that all true slices are constructed by the algorithm, we note that once the algorithm has added a slice to the tree with a nonempty terminal edge which connects that slice to another true slice, then this edge will not be wholly deleted by the trimming procedure, and eventually the following slice will also be picked up by the algorithm and added to the tree. This implies in a straightforward inductive manner that all true slices are added to the tree.

Hence the peels produced by the algorithm are pairwise disjoint and cover the whole surface of K. It is now a simple matter to prove, arguing as in the proof of Lemma 5.1, that each such peel is one of the peels defined in the preceding section, and thus to conclude the proof of correctness of the algorithm.

In view of the discussion at the preceding section, we thus have the following summary theorem.

THEOREM 5.2. *Given a convex polyhedron K with n vertices and a point X on its surface, one can preprocess K by a procedure which runs in $O(n^3 \log n)$ time. This procedure produces a data structure of size $O(n^2)$, with the aid of which one can find in $O(n)$ time the shortest path along the surface of K from X to any other specified point.*

Remark. It seems quite likely that the algorithm developed in this section is not optimal, as it requires $O(n^3 \log n)$ time to construct a quadratic data structure (and then search the structure in only linear time to find a shortest path). After the original submission of this paper, Mount [Mo] has recently proposed an improved approach, in which the data structure only maintains points of intersections of slices with the edges of K (rather than with its faces, as done here). Thus his data structure is a collection of disjoint intervals on the edges of K, rather than a collection of disjoint polygons on the faces of K, making it much easier to maintain this structure, and thereby reducing the running time to $O(n^2 \log n)$.

Note added in proof. (1) Recently the running time of the 2-D shortest path algorithm has been improved to $O(n^2)$ by Welzl and by Guibas, using more efficient constructions of the visibility graph. (2) The running time of the slice-construction algorithm in § 5 can be reduced to $O(n^3)$ using a new data structure, known as F-heap and due to Fredman and Tarjan, to represent the priority queue needed there (this however is pre-empted by Mount's result mentioned above).

REFERENCES

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

[Al] A. D. ALEXANDROV, *Konvexe Polyeder* (translated from Russian), Akademie-Verlag, Berlin, 1958.

[Ch] B. CHAZELLE, *A theorem on polygon cutting with applications*, Proc. 23rd IEEE Symposium on Foundations of Computer Science, 1982, pp. 339–349.

[Co] G. E. COLLINS, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, in Second GI Conference on Automata Theory and Formal Languages, Lecture Notes in Computer Science 33, Springer-Verlag, Berlin, pp. 134–183.

[FAV] W. R. FRANKLIN, V. AKMAN AND C. VERRILLI, *Voronoi diagrams with barriers and on polyhedra*, Tech. Rept., Electrical, Computer and Systems Engineering Dept., Rensselaer Polytechnic Institute, Troy, NY, November 1983.

[FA] W. R. FRANKLIN AND V. AKMAN, *Minimal paths between two points on/around convex polyhedra*, Tech. Rept., Electrical, Computer and Systems Engineering Dept., Rensselaer Polytechnic Institute, Troy, NY, May 1984.

[Ga] M. GARDNER, *The 2nd Scientific American Book of Mathematical Puzzles and Diversions*, Simon and Schuster, New York, 1961.

[LP] D. T. LEE AND F. P. PREPARATA, *Euclidean shortest paths in the presence of rectilinear barriers*, Networks, 14 (1984), pp. 393–410.

[LW] T. LOZANO-PEREZ AND M. A. WESLEY, *An algorithm for planning collision-free paths among polyhedral obstacles*, Comm. ACM, 22 (1979), pp. 560–570.

[Mo] D. M. MOUNT, *On finding shortest paths on convex polyhedra*, Tech. Rept., Computer Science Dept., Univ. Maryland, College Park, October 1984.

[NP] J. NIEVERGELT AND F. P. PREPARATA, *Plane sweeping algorithms for intersecting geometric figures*, Comm. ACM, 25 (1982), pp. 739–747.

[OSB] J. O'ROURKE, S. SURI AND H. BOOTH, *Shortest paths on polyhedral surfaces*, extended abstract, Dept. Electrical Engineering and Computer Science, Johns Hopkins Univ., Baltimore, September 1984.

[Pa] C. H. PAPADIMITRIOU, *An algorithm for shortest-path motion in three dimensions, manuscript*, Computer Science Dept., Stanford Univ., Stanford, CA, July 1984.

[Re] J. REIF, *Minimum s-t cut of a planar undirected network in $O(n \log^2 n)$ time*, this Journal, 12 (1983), pp. 71–81.

[SS] J. T. SCHWARTZ AND M. SHARIR, *On the piano movers' problem*: II. *General techniques for computing topological properties of real algebraic manifolds*, Adv. Appl. Math., 4 (1983), pp. 298–351.

[Sh] M. I. SHAMOS, *Computational Geometry*, Ph.D. Dissertation, Yale Univ., New Haven, CT, 1975.

[To] M. TOMPA, *An optimal solution to a wire routing problem*, J. Comp. Syst. Sci., 23 (1981), pp. 127–150.

[Wa] B. L. VAN DER WAERDEN, *Algebra*, 5th Edition, Springer-Verlag, Berlin, 1960.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986

© 1986 Society for Industrial and Applied Mathematics

AN EFFICIENT ALGORITHM FOR GENERATING LINEAR TRANSFORMATIONS IN A SHUFFLE-EXCHANGE NETWORK*

T. ETZION† AND A. LEMPEL†

Abstract. This paper presents an algorithm for generating all the permutations defined by linear transformations on a shuffle-exchange network of 2^n processors in $2n-1$ passes. The proposed algorithm generates any such permutation in $O(n \log^2 n)$ elementary steps. The subclass of bit-permutations is generated in $O(n)$ steps.

Key words. algorithm, complexity, linear transformations, permutations, shuffle-exchange network

1. Introduction. The shuffle-exchange (SE) network is an efficient tool for implementing various types of parallel processes [2], [6]. The SE network is composed of $N=2^n$ processors, where each processor is represented by a binary n-tuple $(x_1, x_2, \cdots, x_n)$. In the SHUFFLE-operation processor $(x_1, x_2, \cdots, x_n)$ transfers information to processor $(x_2, \cdots, x_n, x_1)$. In the EXCHANGE-operation processors $(x_1, x_2, \cdots, x_{n-1}, 0)$ and $(x_1, x_2, \cdots, x_{n-1}, 1)$ may exchange information, independent of other pairs of this form.

One SHUFFLE followed by one EXCHANGE is called a *pass.* Between the SHUFFLE phase and the EXCHANGE phase of a pass there is a computational phase during which the active pairs of the upcoming EXCHANGE are determined. Prior to the first pass there is normally a preprocessing stage. The overall procedure consisting of the preprocessing stage and all the passes is often referred to as the *routing algorithm.*

An important problem in this context is the design of efficient routing algorithms that implement permutations in a SE network in a minimal number of passes. In general, a transformation on a SE network associates with each processor a destination processor for the purpose of information transfer. This paper deals with the realization of nonsingular linear transformations, i.e., permutations for which each bit of the destination processor is a linear combination of the bits of the origin processor. It is well known ([3] and [4]) that such permutations can be realized in $2n-1$ passes, using a routing algorithm of $O(n^2)$ steps.

In § 2 we show how to realize these permutations in $2n-1$ passes, using a routing algorithm of $O(n \log^2 n)$ steps. In § 3 we show that if the permutation is merely a bit permutation, then only $O(n)$ steps are required.

Following Linial and Tarsi [3], we employ the combinatorial model described below.

DEFINITION 1[3]. A 0-1 matrix $\mathbf{A}$, of order $N \times m$, $N=2^n$, $m \geqq n$, is said to be *balanced* if all the rows in any n consecutive columns of $\mathbf{A}$ are distinct.

DEFINITION 2. The *standard* matrix is an $N \times n$ matrix $\mathbf{D}$ whose ith row is the base-2 representation of i, $0 \leqq i \leqq N-1$.

In terms of these definitions our problem can be stated as follows [3]: Given a balanced $N \times n$ matrix $\mathbf{A}$ find a (possibly empty) matrix $\mathbf{X}$ such that the matrix $[\mathbf{D} \mid \mathbf{X} \mid \mathbf{A}]$ is balanced.

2. Realization of linear transformations. In this section we show how to realize linear transformations on a SE network in $2n-1$ passes using a routing algorithm of $O(n \log^2 n)$ steps.

* Received by the editors May 9, 1984, and in final form November 9, 1984.

† Computer Science Department, Technion, Israel Institute of Technology, Haifa, Israel.

In the sequel all the arithmetic is over $GF(2)$, all the vectors are column vectors with n elements, $\mathbf{I}=[I(1)I(2)\cdots I(n)]$ denotes the identity matrix of order n, and $\mathbf{T}$ denotes a nonsingular matrix of order n.

PROPOSITION 1[3]. *Let* $\mathbf{A}$ *be a matrix of order* $N\times n$. *Then* $\mathbf{AT}$ *is balanced if and only if* $\mathbf{A}$ *is.*

DEFINITION 3. A matrix $\mathbf{R}$ of order $n\times m$, $n\leqq m$, is said to be *n-regular* if every n consecutive columns of $\mathbf{R}$ are linearly independent.

It follows readily from Proposition 1 that if $\mathbf{R}$ is n-regular then $\mathbf{DR}$ is balanced. In what follows we consider n-regular matrices of the form $\mathbf{R}=[\mathbf{I}\,\vdots\,\mathbf{Y}\,\vdots\,\mathbf{T}]$ and propose a method of finding a suitable matrix $\mathbf{Y}$ of $n-1$ columns when given the matrix $\mathbf{T}$.

Consider a $n\times n$ binary matrix $\mathbf{B}=[B(1)B(2)\cdots B(n)]$, where each column $B(i)$ has either one or two nonzero entries. $\mathbf{B}$ can be viewed as the incidence matrix of the (undirected) graph $G(\mathbf{B})$ defined as follows:

DEFINITION 4. $G(\mathbf{B})$ has $n+1$ vertices $0,1,\cdots,n$ and n edges $e(1)$, $e(2),\cdots,e(n)$, where $e(k)$ joins vertices $i>0$ and $j>0$ if $B(k)$ has nonzero entries in rows i and j, and $e(k)$ joins vertices $i>0$ and 0 if $B(k)$ is nonzero in row i only.

LEMMA 1[5]. *The vectors* $B(1), B(2),\cdots, B(n)$ *are linearly independent if and only if* $G(B)$ *is a tree.*

LEMMA 2. *Let* $B(1), B(2),\cdots, B(n)$ *be linearly independent vectors. Then there exists an integer* k, $1\leqq k\leqq n$, *and binary coefficients* b_j, $1\leqq j\leqq n-1$, *such that*

$$I(k)=B(n)+\sum_{j=1}^{n-1} b_jB(j). \tag{1}$$

Proof. The matrix $\mathbf{B}=[B(1)B(2)\cdots B(n)]$ is nonsingular. Hence, there exists a matrix $\mathbf{Q}=[Q(1)Q(2)\cdots Q(n)]$ such that $\mathbf{BQ}=\mathbf{I}$. Since $\mathbf{Q}$ is nonsingular, there exists at least one k such that the last entry of $Q(k)$ equals 1. Q.E.D.

LEMMA 3. *Let* $B(1), B(2),\cdots, B(n)$ *be linearly independent vectors and let* k, $1\leqq k\leqq n$ *be an integer satisfying Lemma* 2. *Then* $B(0), B(1), B(2),\cdots, B(n-1)$ *are linearly independent, where*

$$B(0)=I(k)+\sum_{j=1}^{n-1} c_jB(j), \qquad c_j\in\{0,1\}. \tag{2}$$

Proof. Assume, the contrary, that $B(0), B(1), B(2),\cdots, B(n-1)$ are linearly dependent. Then, since the last $n-1$ vectors are linearly independent, there exist d_j, $1\leqq j\leqq n-1$, such that

$$B(0)=\sum_{j=1}^{n-1} d_jB(j). \tag{3}$$

From (1), (2), and (3), we obtain

$$B(n)=\sum_{j=1}^{n-1}(b_j+c_j+d_j)B(j)$$

which contradicts the linear independence of the $B(j)$, $1\leqq j\leqq n$. Q.E.D.

Based on Lemmas 1 and 3, we propose the following construction of $\mathbf{Y}=[Y(1)Y(2)\cdots Y(n-1)]$ such that $[\mathbf{I}\,\vdots\,\mathbf{Y}\,\vdots\,\mathbf{T}]$ be n-regular for a given $\mathbf{T}=[T(1)T(2)\cdots T(n)]$.

Construction 1. Let $\mathbf{B}_0=\mathbf{T}$ and let $\mathbf{B}_m=[Y(n-m)\cdots Y(n-1)T(1)\cdots T(n-m)]$, $1\leqq m\leqq n-1$. Given $\mathbf{B}_m$, $0\leqq m<n-1$, construct $Y(n-m-1)$ as follows.

(i) If $k=n-m-1$ satisfies Lemma 2, set $Y(n-m-1)=I(n-m-1)$.

(ii) If $k = n - m - 1$ does not satisfy Lemma 2, find an integer q which does satisfy Lemma 2 and set $Y(n-m-1) = I(n-m-1) + I(q)$.

LEMMA 4. *The matrix* $[I(1) \cdots I(n) Y(1) \cdots Y(n-1) T(1) \cdots T(n)]$ *obtained via Construction* 1 *is n-regular.*

Proof. The n-regularity of $[Y(1) \cdots Y(n-1) T(1) \cdots T(n)]$ follows directly from Lemma 3. To complete the proof, it suffices to show that $[J(1) \cdots I(n) Y(1) \cdots Y(n-1)]$ is n-regular. Let $\mathbf{C}_1 = \mathbf{I}$ and let $\mathbf{C}_m = [I(m) \cdots I(n) Y(1) \cdots Y(m-1)]$, $1 < m \leqq n$. We will show that linear independence among the columns of $\mathbf{C}_m$, $1 \leqq m < n$ implies the same for $\mathbf{C}_{m+1}$. Clearly, the columns of $\mathbf{C}_1$ are linearly independent. Suppose $\mathbf{C}_m$, $m \geqq 1$, is nonsingular and consider $\mathbf{C}_{m+1} = [I(m+1) \cdots I(n) Y(1) \cdots Y(m)]$. By Construction 1, either $Y(m) = I(m)$ or $Y(m) = I(m) + I(q)$ for some $q \neq m$. In the first case it is clear that $\mathbf{C}_{m+1}$ is nonsingular. In the latter case noting that $\mathbf{C}_r$, $1 \leqq r < n$, has at most two nonzero entries in every column, we can view $\mathbf{C}_r$ as the incidence matrix of the graph $G(\mathbf{C}_r)$ according to Definition 4. By Lemma 1, since $\mathbf{C}_m$ is nonsingular $G(\mathbf{C}_m)$ is a tree. $G(\mathbf{C}_{m+1})$ is obtained from $G(\mathbf{C}_m)$ by deleting the edge $(0, m)$ (corresponding to the column $I(m)$) and inserting the edge (m, q) (corresponding to the column $I(m) + I(q)$). If $G(\mathbf{C}_{m+1})$ contains a cycle then, since $Y(1), \cdots, Y(n-1)$ are linearly independent, the cycle must include the vertex 0. Since $G(\mathbf{C}_m)$ is a tree, deleting the edge $(0, m)$ from $G(\mathbf{C}_m)$ leaves a graph with no path between the vertices 0 and m. Hence, inserting the edge (m, q) cannot generate a cycle that contains the vertex 0. Hence $G(\mathbf{C}_{m+1})$ is a tree, and $\mathbf{C}_{m+1}$ is nonsingular. Q.E.D.

In order to find an integer k that satisfies Lemma 2, we need an efficient algorithm to invert a matrix. To this end we use the algorithm proposed by Csanky [1] who showed how to invert a matrix of order n, in $O(\log^2 n)$ steps using a polynomial number of processors. Csanky's algorithm utilizes a model that has an arbitrary number of identical processors with independent control and an arbitrarily large shared memory with unrestricted access. In this model, each processor is capable of taking its operands from the shared memory performing any one of the binary operations $+, -, *, /$ and storing the result in memory in one step.

Based on Lemma 2, Csanky's algorithm, and Construction 1, we propose a procedure to realize linear transformations. In this procedure each processor has at each stage the following information:

(1) An $(n-1)$-tuple $U = (u(1), \cdots, u(n-1))$, where $u(j) = k$ if $Y(j) = I(j) + I(k)$ and $u(j) = 0$ if $Y(j) = I(j)$.

(2) Two n-tuples, S and $F = S\mathbf{T}$, whose initial values represent, respectively, the ID of the said processor and that of the destination processor as defined by the given linear transformation. In the SHUFFLE and the EXCHANGE operations that follow each processor transfers its current S and F and receives new values for S and F.

Procedure 1. Given the linear transformation defined by a nonsingular matrix $\mathbf{T} = [T(1) T(2) \cdots T(n)]$ let $\mathbf{B}_0 = T$ and let $\mathbf{B}_m = [Y(n-m) \cdots Y(n-1) T(1) \cdots T(n-m)]$. Having computed $\mathbf{B}_r = [Y(n-r) \cdots Y(n-1) T(1) \cdots T(n-r)]$, $r \geqq 0$, apply the Csanky algorithm to generate the inverse $\mathbf{Q} = [Q(1) \cdots Q(n)]$ of $\mathbf{B}_r$. If the last entry of $Q(n-1)$ equals 1, set $Y(n-r-1) = I(n-r-1)$ and $u(n-r-1) = 0$; otherwise, find an integer k such that the last entry of $Q(k)$ equals 1 and set $Y(n-r-1) = I(n-r-1) + I(k)$ and $u(n-r-1) = k$. After $u(1), \cdots, u(n-1)$ are generated they are transferred to each of the N processors of the SE network. With reference to the n-tuples $S = (s(1), \cdots, s(n))$ and $F = (f(1), \cdots, f(n))$, stored with each processor, perform the following:

```
s(0) := 0;
for i := 1 to n-1 do
begin
  SHUFFLE;
  if s(u(i)) ≠ 0 then EXCHANGE;
end;
SHUFFLE;
if s(n) ≠ f(1) then EXCHANGE;
for i := 1 to n-1 do
begin
  SHUFFLE;
  if f(i+1) ≠ s(i)+s(u(i)) then EXCHANGE;
end;
```

Note that SHUFFLE and EXCHANGE are executed in parallel by all the processors.

THEOREM 1. *Procedure* 1 *realizes a linear transformation in* $2n-1$ *passes using a routing algorithm of* $O(n \log^2 n)$ *steps.*

Proof. In order to show that Procedure 1 realizes the linear transformation associated with the matrix $\mathbf{T}$, it suffices to show that it implements the moves implied by the balanced matrix $\mathbf{DB} = \mathbf{D}[\mathbf{I} \mid Y(1) \cdots Y(n-1) \mid \mathbf{T}] = [\mathbf{D} \mid \mathbf{D}Y(1) \cdots \mathbf{D}Y(n-1) \mid \mathbf{DT}]$. That is, for a given processor $S = (s(1), \cdots, s(n))$ and its destination processor $F = (f(1), \cdots, f(n))$, the path in the SE network via which the transformation $F = S\mathbf{T}$ is implemented by Procedure 1 is given by the sequence of processors corresponding to successive n-tuples from the row $S\mathbf{B} = s(1), \cdots, s(n)$, $(s(1)+s(u(1)))$, $(s(2)+s(u(2)))$, $\cdots$, $(s(n-1)+s(u(n-1)))$, $f(1), \cdots, f(n)$. To this end, note that for each row $S\mathbf{B}$, Procedure 1 performs an EXCHANGE if and only if the leading bit of the current processor differs from the last bit of the succeeding processor.

The claimed complexity of Procedure 1 is obtained as follows. The n-regular matrix $\mathbf{B} = [\mathbf{I} \mid Y(1) \cdots Y(n-1) \mid \mathbf{T}]$ is generated by $n-1$ applications of the Csanky algorithm. Therefore, this part consists of $O(n \log^2 n)$ steps. The $(n-1)$-tuple $U = (u(1), \cdots, u(n-1))$ is transferred to each of the N processors of the SE network on a bus in $O(n)$ steps. The $2n-1$ passes correspond to last $2n-1$ columns of $\mathbf{DB}$ and each pass is executed in constant time. Thus, the overall complexity of the procedure is $O(n \log^2 n)$. Q.E.D.

3. Realization of bit-permutations. In this section we show how to realize the linear transformation associated with a permutation matrix $\mathbf{T}$ in $O(n)$ steps.

DEFINITION 5. $\mathbf{T} = [T(1)T(2) \cdots T(n)]$ is called a permutation matrix if $T(j) = I(p(j))$, $j = 1, 2, \cdots, n$, where $p(1), p(2), \cdots, p(n)$ is an arbitrary permutation on the integers $1, 2, \cdots, n$.

Based on Lemma 1, we propose the following construction of $\mathbf{Y} = [Y(1)Y(2) \cdots Y(n-1)]$ such that $[\mathbf{I} \mid \mathbf{Y} \mid \mathbf{T}]$ is n-regular for a given permutation matrix $\mathbf{T} = [I(p(1))I(p(2)) \cdots I(p(n))]$.

Construction 2. Let $\mathbf{B}_0 = \mathbf{T}$ and let $\mathbf{B}_m = [Y(n-m) \cdots Y(n-1)I(p(1)) \cdots I(p(n-m))]$, $1 \leqq m \leqq n-1$. Along with the columns of $\mathbf{Y}$ we construct a sequence of graphs G_i, $0 \leqq i \leqq n-1$. G_0 is the edgeless graph of n isolated vertices $1, 2, \cdots, n$, and given $\mathbf{B}_m$ and G_m, $0 \leqq m < n-1$, construct $Y(n-m-1)$ and G_{m+1} as follows.

If the addition of edge $(n-m-1, p(n-m))$ to G_m creates a cycle, set $Y(n-m-1) = I(n-m-1)$ and $G_{m+1} = G_m$; otherwise, set $Y(n-m-1) = I(n-m-1) + I(p(n-m))$ and obtain G_{m+1} by adding the edge $(n-m-1, p(n-m))$ to G_m.

LEMMA 5. *The matrix* $[I(1)\cdots I(n)Y(1)\cdots Y(n-1)I(p(1))\cdots I(p(n))]$ *obtained via Construction* 2 *is n-regular.*

Proof. First, observe that every column of the matrix $\mathbf{B}_r$, $0 \leqq r \leqq n-1$, has at most two nonzero entries and, thus, can be viewed as the incidence matrix of the graph $G(\mathbf{B}_r)$ defined in § 2. Note that G_r, as defined by Construction 2, can be obtained from $G(\mathbf{B}_r)$ by deleting from the latter the vertex 0 and all the edges incident with this vertex. Note further that $G(\mathbf{B}_{m+1})$ is obtained from $G(\mathbf{B}_m)$ by:

(i) Deletion of edge $(0, p(n-m))$.

(ii) Addition of either edge $(0, n-m-1)$, or edge $(p(n-m), n-m-1)$.

Assume that $G(\mathbf{B}_m)$, $m \geqq 0$, is a tree. Then, operation (i) results in two pieces of $G(\mathbf{B}_m)$, with no path between vertices 0 and $P(n-m)$. Hence, if at this stage connecting vertex $p(n-m)$ to vertex $n-m-1$ creates a cycle, it follows that operation (i) leaves vertex $n-m-1$ in the same piece with vertex $p(n-m)$, namely with no path between vertex 0 and vertex $n-m-1$. Therefore, in this case, the graph $G(\mathbf{B}_{m+1})$ obtained in operation (ii) by adding the edge $(0, n-m-1)$ is a tree.

If, on the other hand, connecting vertex $p(n-m)$ to vertex $n-m-1$, after operation (i), does not create a cycle in the piece containing vertex $p(n-m)$, it certainly does not create a cycle with vertex 0 and the resulting graph is again a tree.

Since $G(\mathbf{B}_0)$ is a tree it follows that $G(\mathbf{B}_m)$ is a tree for all $0 \leqq m \leqq n-1$, which implies that the matrix $[Y(1)\cdots Y(n-1)I(p(1))\cdots I(p(n))]$ is n-regular.

The n-regularity of the matrix $[I(1)\cdots I(n)Y(1)\cdots Y(n-1)]$ follows in the same manner as in the proof of Lemma 4. Q.E.D.

The reader can readily verify that the result of Construction 2 could be obtained via Construction 1 through an appropriate choice of the parameter k of Lemma 2.

Construction 2 leads to Procedure 2, given below for realizing bit-permutations. In this procedure, which is simpler than Procedure 1, each processor has at each stage the following information:

(1) an $(n-1)$-tuple $U = (u(1), \cdots, u(n-1))$ as in Procedure 1;

(2) an n-tuple S as in Procedure 1;

(3) the permutation $P = (p(1), \cdots, p(n))$.

Procedure 2.

```
Part 1
for i := 1 to n-1 do
begin
  u(i) := p(i+1);
  check (i) := false;
end;
check (n) := false;
for i := 1 to n-1 do
begin
  cycle := false;
  current := i;
  while ((cycle = false) and (current < n) and (check(current) = false)) do
  begin
    check (current) := true;
    if u(current) ≠ i then current := u (current)
    else cycle := true;
  end;
  if cycle = true then u(i) := 0;
end;
```

Part 2
$s(0) := 0$;
for $i := 1$ to $n-1$ do
begin
  SHUFFLE;
  if $s(u(i)) \neq 0$ then EXCHANGE;
end;
SHUFFLE;
if $s(n) \neq s(p(1))$ then EXCHANGE;
for $i := 1$ to $n-1$ do
begin
  SHUFFLE;
  if $s(p(i+1)) \neq s(i) + s(u(i))$ then EXCHANGE;
end;

THEOREM 2. *Procedure 2 realizes a bit-permutation in* $2n-1$ *passes and* $O(n)$ *steps.*

Proof. In Part 1 of Procedure 2 each processor computes the $(n-1)$-tuple $U = (u(1), \cdots, u(n-1))$. Initially $u(n-m-1)$ is set to $p(n-m)$ which corresponds to setting $Y(n-m-1)$ to $I(n-m-1) + I(p(n-m))$. Then, $u(n-m)$ is set to 0 if the insertion of the edge $(n-m-1, p(n-m))$ creates a cycle in the corresponding graph G_m. Part 2 of Procedure 2 is identical to Procedure 1, with $s(p(i))$ substituting for $f(i)$.

The claimed complexity of Procedure 2 is obtained as follows. Part 1 consists of $O(n)$ steps since the variables *check*(i), $1 \leqq i \leqq n$, insure that for each i, the variable *current* takes the value i at most once in the while loop. As in Procedure 1, each of the $2n-1$ passes is executed in constant time. Thus, the overall complexity of the procedure is $O(n)$. Q.E.D.

Acknowledgment. The authors wish to thank Yossi Shiloach for presenting the problem and for many helpful discussions.

REFERENCES

[1] L. CSANKY, *Fast parallel matrix inversion algorithms*, this Journal, 4 (1976), pp. 618-623.
[2] D. H. LAWRIE, *Access and alignment of data in an array processor*, IEEE Trans. Comput., C-25 (1976), pp. 55-65.
[3] N. LINIAL AND M. TARSI, *Efficient generation of permutations with the shuffle exchange networks*, submitted for publication.
[4] M. PEASE, *The indirect binary n-cube microprocessor*, IEEE Trans. Comput., C-26 (1977), pp. 122-131.
[5] S. SESHU AND M. B. REED, *Linear Graphs and Electrical Networks*, Addison-Wesley, Reading, MA, 1961.
[6] H. S. STONE, *Parallel processing with the perfect shuffle*, IEEE Trans. Comput., C-20 (1971), pp. 153-161.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986

© 1986 Society for Industrial and Applied Mathematics

VARIABLE SIZED BIN PACKING*

D. K. FRIESEN† AND M. A. LANGSTON‡

Abstract. In the classical bin packing problem one seeks to pack a list of pieces in the minimum space using unit capacity bins. This paper addresses the more general problem in which a fixed collection of bin sizes is allowed. Three efficient approximation algorithms are described and analyzed. They guarantee asymptotic worst-case performance bounds of 2, 3/2 and 4/3.

Key words. bin packing, approximation algorithms, worst-case analysis

1. Introduction. Consider a finite collection of bin sizes and an inexhaustible supply of bins of each size. The problem we investigate is that of packing a list of pieces into bins so as to minimize the total space used in the packing. If the cost of a bin is proportional to its size, then our objective corresponds to minimizing the total cost of the packing. This problem arises in a variety of interpretations from computer storage allocation to stock cutting.

This variable sized bin packing problem is *NP*-hard, reducing to the classical bin packing problem when only one bin size is permitted. Hence we focus our efforts on efficient approximation algorithms to ensure near-optimal packings. We seek to establish an algorithm's worst-case performance bound, which is a ceiling on the ratio of the space used in that algorithm's packing of a given list to the minimum space required for any packing of the same list. Of course the problem could be solved exactly, given enough time. An attractive scheme is described in [GG] which uses a cutting stock interpretation, linear programming techniques and repeated applications of an algorithm for solving the knapsack problem (also *NP*-hard).

The classical bin packing problem and many of its variations are of fundamental importance, reflected in the impressive amount of research reported (see [CGJ] for an updated survey). Yet only a few results have appeared concerning this more general problem in which bins need not be of a single given size (see for example [FL1], [FL2] or [La]). In particular, it is known from [FL2] that if the user is allowed to choose *any single* bin size, then any packing algorithm can be employed iteratively to yield a worst-case performance bound arbitrarily close to its bound for the classical problem.

The problem we consider herein appears to be substantially more difficult and our results are more modest. In § 3, we present two procedures and show that they guarantee tight worst-case performance ratios of 2 and 3/2. Our main result is a proof that a slightly more complex algorithm we name FFDLS, to be described and analyzed in § 4, has a worst-case performance ratio of 4/3. Moreover, this bound is tight.

2. The model. Let k denote the number of distinct bin sizes available. A list of pieces is specified by $L=\{p_1, p_2, \cdots, p_n\}$. Assume that bin and piece sizes are normalized so that the largest bin has size 1 (and hence the largest piece must have a size no greater than 1). We let $B=\langle B_1, B_2, \cdots, B_l\rangle$ denote the ordered set of bins used by a heuristic H. That is, B_1 is the first bin to receive a piece, B_2 is the next bin to receive a piece, and so on. In the same fashion we let $B^*=\langle B_1^*, B_2^*, \cdots, B_m^*\rangle$ denote the ordered set of bins used by some optimal procedure OPT. $|B_i|(|B_i^*|)$ denotes the number of pieces assigned to the ith bin by the heuristic (optimal) packing.

* Received by the editors October 18, 1983, and in revised form December 1, 1984.

† Computer Science Department, Texas A & M University, College Station, Texas 77843.

‡ Computer Science Department, Washington State University, Pullman, Washington 99164. This author's research was partially supported by the National Science Foundation under grant ECS-8403859.

We let $s(p_i)$ denote the size of the ith piece. Similarly we use s to represent bin sizes and a function c to specify the total contents of a bin (clearly $c(B_i) \leqq s(B_i)$). Since we seek a bound on the space required by an algorithm, we define $H(L) \equiv \sum_{i=1}^{l} s(B_i)$ and $\mathrm{OPT}(L) \equiv \sum_{i=1}^{m} s(B_i^*)$. We will establish results of the form $H(L) \leqq R \cdot \mathrm{OPT}(L) + C$ where R and C are the asymptotic worst-case performance ratio and the additive constant, respectively.

3. The algorithms NFL and FFDLR. Guaranteeing a bound of 2 is very easy. Consider the simple heuristic NFL (for Next Fit using Largest bins only) which packs a bin of size 1 until a piece will not fit, and then starts packing a new bin of size 1. NFL is clearly of time complexity $O(n)$, linear in the number of pieces which must be packed.

THEOREM 1. $\mathrm{NFL}(L) < 2\,\mathrm{OPT}(L) + 1$ *for any list* L.

Proof. For $1 \leqq i < l$, $c(B_i) + c(B_{i+1}) > 1$. Therefore $\sum_{i=1}^{l} c(B_i) > (l-1)/2$ and

$$\mathrm{NFL}(L) = (l-1) + 1 < 2 \sum_{i=1}^{l} c(B_i) + 1 = 2 \sum_{i=1}^{m} c(B_i^*) + 1 \leqq 2 \sum_{i=1}^{m} s(B_i^*) + 1$$

$$= 2\,\mathrm{OPT}(L) + 1. \qquad \square$$

Letting ε denote an arbitrary small positive real value, any instance consisting of pieces of size $1/2 + \varepsilon$ and bins of sizes 1 and $1/2 + \varepsilon$ demonstrates that 2 is an asymptotic lower bound as well for NFL's performance.

The major stumbling block to achieving improved performance appears to be the need to repack into smaller bins (the interested reader should have no difficulty devising examples to illustrate that initially packing smaller bins does not help). Moreover, algorithms known to be superior to Next Fit for the classical model (see descriptions in [CGJ], [JDUGG] or [Jo]) also fail unless allowed to repack bins. Hence we consider FFDLR (for First Fit Decreasing using Largest bins, at end Repack to smallest possible bins). A formal description follows:

```
FFDLR: procedure
begin
  sort and reindex L so that s(p_1) ≧ s(p_2) ≧ ··· ≧ s(p_n)
  for i ← 1 to n do
    pack p_i into the first bin of size 1 that has room for it
    end
  for i ← 1 to l do
    repack contents of B_i into the smallest possible empty bin
    end
end.
```

FFDLR is of time complexity $O(n \log n + l \log k)$, comparing favorably with NFL. We now demonstrate that its modest increase in run-time is well worth the effort.

THEOREM 2. $\mathrm{FFDLR}(L) < (3/2)\,\mathrm{OPT}(L) + 1$ *for any list* L.

Proof. The proof is by contradiction. Assume that there exists a set of bin sizes and a list L of pieces such that $\mathrm{FFDLR}(L) \geqq (3/2)\,\mathrm{OPT}(L) + 1$. Let B and B^* be the set of bins used by FFDLR and an optimal algorithm, respectively. Note that any wasted space in B_l can be absorbed in the additive constant. Let i denote the largest index less than l such that $c(B_i) < (2/3)s(B_i)$. Clearly such an i exists. Then $1/2 < c(B_i) < 2/3$. Also, $|B_i| = 1$, else its second piece has size less than $1/3$, $c(B_j) > (2/3)s(B_j)$ for all $j < i$, and the theorem holds with the additive constant absorbing any wasted space in B_i and B_l, since $c(B_i) + c(B_l) > 1$.

We conclude that there is no bin size in $[c(B_i), 3/4]$ and that each B_j, $j \leqq i$, contains a piece whose size is at least $c(B_i)$ and which cannot be packed in a bin of size 1 with any item from $B_{j'}$, $j' > i$. Therefore

$$\text{FFDLR}(L) \leqq i + (3/2) \sum_{j=i+1}^{l} c(B_j) + 1 < (3/2)\left[(3/4)i + \sum_{j=i+1}^{l} c(B_j) \right] + 1$$

$$\leqq (3/2)\,\text{OPT}(L) + 1. \qquad \square$$

To illustrate that this asymptotic bound of 3/2 is tight, consider any instance consisting of an even number of pieces of size $1/3+\varepsilon$ and bins of sizes 1 and $1/3+\varepsilon$.

4. The FFDLS algorithm. Since the single repacking step at the end of FFDLR does not ensure less than 50% wasted space, we consider dynamically shifting bin contents as the packing is constructed. Focusing our attention on a 4/3 performance bound, we design FFDLS (for First Fit Decreasing using Largest bins, but Shifting as necessary). Formally we have:

```
FFDLS: procedure
begin
  sort and reindex L so that s(p_1) ≧ s(p_2) ≧ ··· ≧ s(p_n)
  for i ← 1 to n do
    pack p_i into the first bin B_j of size 1 that has room for it
    if B_j contains a piece whose size exceeds 1/3
      then shift, if possible, the contents of B_j to the smallest empty bin B_j' that
        will hold them where c(B_j) ≧ (3/4)s(B_j')
    end
  for i ← 1 to l do
    repack contents of B_i into the smallest possible empty bin
    end
  end.
```

FFDLS is of time complexity $O(n \log n + n \log k)$. Note that shifting is not performed for bins containing only small pieces (i.e., pieces of size at most 1/3). Roughly speaking this is because such a bin, unless it is B_l, must be filled sufficiently over 3/4 to "balance" bins filled to less than 3/4. To illustrate, suppose we shift anyway and pack $12K$ pieces, $6K$ of size .365 and $6K$ of size .271, using bin sizes 1, .636 and .361. The heuristic requires a space of $3K(1)+6K(.361)$, while the optimal packing needs only $6K(.636)$. (FFDLS would not construct an optimal packing either, but would require a space of only $3K(1)+2K(1)$, wasting less than 33%.)

FFDLS, as described, does not attempt to pack a piece into any "small bin" that is already 3/4 full. That is, first fit packing is done only in bins of size 1. In practice we can allow a first fit packing over all partially filled bins, although this modification can have no effect on the worst-case ratio (it cannot make it worse and does not improve the example mentioned at the end of this section).

We now prove that FFDLS guarantees packings whose asymptotic space requirements never exceed 4/3 the optimal. The major part of the proof relies on "weighting function" arguments (the interested reader is referred to [Co] for an exposition of this technique).

THEOREM 3. $\text{FFDLS}(L) < (4/3)\,\text{OPT}(L) + 3$ *for any list* L.

Proof. The proof is by contradiction. Suppose the existence of a set of bin sizes and a list L of pieces such that $\text{FFDLS}(L) \geqq (4/3)\,\text{OPT}(L) + 3$. Let B and B^* be the set of bins used by FFDLS and an optimal algorithm, respectively. Without loss of

generality, assume this instance is minimal. That is, k is the minimum number of distinct bin sizes for which the statement of the theorem fails to hold and, for this k, n is the minimum number of pieces needed for a counterexample. Let the pieces be indexed such that $s(p_1) \geqq s(p_2) \geqq \cdots \geqq s(p_n)$.

We first show that only pieces of size greater than 1/6 need be considered. Suppose $s(p_n) \leqq 1/6$. Let B_i denote the FFDLS bin containing p_n. Clearly $i \neq l$, else for $1 \leqq j < l$, $c(B_j) \geqq (3/4)s(B_j)$, since either FFDLS shifted the contents to B_j where this property holds or $c(B_j) > 5/6$, implying that $\mathrm{OPT}(L) > (3/4)(\mathrm{FFDLS}(L) - 1)$. From this and the fact p_n was initially packed in a bin of size 1 we conclude that $c(B_i) - s(p_n) > 1/2$. B_i must contain a piece whose size exceeds 1/3, else not only does it hold that, for $1 \leqq j < i$, $c(B_j) \geqq (3/4)s(B_j)$, but also that, for $i \leqq j < l$, B_j contains at least three pieces and thus $c(B_j) > 3/4$ again implying that $\mathrm{OPT}(L) > (3/4)(\mathrm{FFDLS}(L) - 1)$. Since n is the minimum number of pieces required for a counterexample, it must be that p_n affects the packing. Hence there is some bin size b such that $(c(B_i) - s(p_n)) + s(p_n) > b > (4/3)(c(B_i) - s(p_n))$ and we derive $s(p_n) > 1/6$.

We define piece types as follows: p_i is of type X_1 if $s(p_i) > 1/2$, of type X_2 if $1/2 \geqq s(p_i) > 1/3$, of type Y otherwise. We call a bin B "short" if $c(B) < (3/4)s(B)$. Clearly such a bin exists. A bin can be short only under very special conditions. If a short bin, B, contains an X_1 piece, a, then B can contain at most one additional piece, b, where $1/6 < s(b) < 3/4 - s(a) < (1 - s(a))/2$. Such a piece b implies that $s(a) < 3/4 - s(b) < 2/3$. Thus b must be the only Y piece available for B, since any larger piece would have been used instead of b and any piece no larger would have been packed with a and b. If a short bin, B, contains an X_2 piece as its largest piece, then either B contains two X_2 pieces and nothing else or there are no more X_2 pieces available for B. Except for B_l, a short bin's largest piece cannot be of type Y. We thus allow for four "exceptional bins":

1) the last bin which contains one X_1 piece and one Y piece if there are no Y pieces remaining;
2) the last bin of size greater than 8/9 which contains exactly two X_2 pieces;
3) the bin which contains one X_2 piece and one Y piece;
4) B_l.

Note that at most three of these can exist in the same packing, and we can use the additive constant to absorb their wasted space. A nonexceptional short bin must contain either a single X_1 piece or exactly two X_2 pieces. The last nonexceptional short bin will be denoted by BIN in the remainder of the proof.

We divide the analysis into two cases, based on $|\mathrm{BIN}|$. For each case we will construct a weighting function w on L. A piece's weight will depend on its type and its placement in the FFDLS packing. A bin's weight is simply the total weight of all its pieces. By the proper definition of w we will show that

$$(3/4)\left(\sum_{i=1}^{l} s(B_i) - 3\right) \leqq \sum_{i=1}^{l} w(B_i) = \sum_{i=1}^{m} w(B_i^*) \leqq \sum_{i=1}^{m} s(B_i^*).$$

Case I. $|\mathrm{BIN}| = 2$. For this case we claim $s(p_n) > 1/5$. Suppose otherwise. Let B_i denote the FFDLS bin containing p_n. For $1 \leqq j < i$, $c(B_j) \geqq (3/4)s(B_j)$, since either FFDLS shifted the contents to B_j where this property holds or $c(B_j) > 4/5$. Thus $i < l$. Also $c(B_i) - s(p_n) > 3/5$, else for $i < j \leqq l$ either $|B_j| = 1$ or $c(B_j) > (3/4)s(B_j)$ indicating, since $|\mathrm{BIN}| = 2$, that only $B_i = \mathrm{BIN}$ and B_l can be short, which is impossible. There is at least one B_j, $i \leqq j < l$, such that $c(B_j) < (3/4)s(B_j) \leqq 3/4$ and hence B_i must contain a piece whose size exceeds 1/3. Minimality dictates that p_n affect the packing. Thus

there is some bin size b satisfying $(c(B_i)-s(p_n))+s(p_n)>b>(4/3)(c(B_i)-s(p_n))$ which implies $s(p_n)>1/5$.

We next show that every nonexceptional short bin contains two X_2 pieces. The only other possibility is a B_i containing only one piece, z, of type X_1. The size of BIN's second piece exceeds 1/4, though it is less than 3/8, since BIN must be packed after B_i. Hence B_i is not short unless $s(z)>5/8$. But if this is the case, we can construct a smaller counterexample as follows. Delete z from L. If the bin B_j^* in the optimal packing which had contained z has no other piece, halt. Otherwise delete v, the remaining piece of B_j^* (it can have only one such piece since $5/8+2/5>1$) and z', the X_1 piece packed with it in the FFDLS packing (z' must have been packed before z since FFDLS left z alone in B_i though v would have fit). Now examine the optimal bin which had contained z' and so on. This procedure halts in a finite number of steps. L has been shortened. Both packings have lost the same number of bins. Every optimal bin deleted has size $>(4/3)(5/8)>3/4$. Hence we still have a counterexample.

The function w is defined in Table 1. "Exceptional pieces" (that is, pieces from exceptional bins) have a weight of zero. Some of the restrictions deserve comment. If B_i contains an X_1 piece and a Y piece, then the size of the X_1 piece exceeds 5/8 since BIN contains two X_2 pieces, one having size less than 3/8. For this same reason, B_i cannot contain an X_1 piece and two Y pieces. If B_i contains three Y pieces, then the size of each piece exceeds 1/4, since otherwise BIN cannot be short. This observation also rules out the possibility of B_i containing four Y pieces. Should B_i contain one X_2 piece and two Y pieces, we treat it as if all three pieces were of type Y. Note that for any nonexceptional bin B_i, $w(B_i)=(3/4)s(B_i)$. Also, any piece p_i from a nonexceptional bin satisfies $w(p_i)\leqq s(p_i)$, unless p_i is an X_2 piece of weight greater than 1/3.

Therefore the proof of Case I reduces to showing that even if such an X_2 piece is in B_i^*, $w(B_i^*)\leqq s(B_i^*)$.

TABLE 1

The function w for Case I.

Contents of B_i	Piece weights (times $s(B_i)$)	Restrictions
X_1	3/4	$s(X_1)>(3/4)s(B_i)$
X_1, X_2	1/2, 1/4	$s(X_1)>1/2$, $s(X_2)>1/3$
X_1, Y	5/8, 1/8	$s(X_1)>5/8$, $s(Y)>1/5$
X_2	3/4	$s(X_2)\geqq(3/4)s(B_i)$
X_2, X_2	3/8, 3/8	$s(X_2)>1/3$
X_2, X_2, Y	5/16, 5/16, 1/8	$s(X_2)>1/3$, $s(Y)>1/5$
Y, Y, Y	1/4	$s(Y)>1/4$

Suppose $|B_i^*|=1$. Let p denote the single piece of B_i^*. FFDLS could not have packed p as the first piece in a bin, else $w(B_i^*)\leqq 3/8<4/9<(4/3)s(p)<s(B_i^*)$ since p was not shifted. Let B_j be the last short bin containing two X_2 pieces (such a bin must exist and is exceptional). p could not have been packed in a bin preceding B_j, else every X_2 piece available for B_j would have been shifted into a bin of size at most $s(B_i^*)$. Neither can p have been packed in a bin following B_j, else such a bin is short contradicting the definition of B_j or has size less than $c(B_j)<3/4$ and $w(p)<(3/8)(3/4)<1/3<s(p)$. The only remaining possibility is that p comes from B_j in which case $w(p)=0$.

Suppose $|B_i^*|=2$. Suppose the second piece was not shifted into a smaller bin by FFDLS. If this piece is an X_1 piece, then $s(B_i^*)>5/6$ and the contents of the initial FFDLS bin containing the problem X_2 piece would have been repacked into a bin of size no larger than $s(B_i^*)$. Therefore, $w(B_i^*)\leqq 1/2+(3/8)s(B_i^*)<s(B_i^*)$. If the second piece of B_i^* is another X_2 piece, then it must be that $s(B_i^*)<3/4$. But this means that one of the X_2 pieces was packed with an X_1 piece, else the smaller X_2 piece would have been shifted or been exceptional. Thus $w(B_i^*)\leqq 3/8+1/4<2/3<s(B_i^*)$. Hence it can only be that this piece is of type Y. It is packed by FFDLS into a bin containing only Y pieces, else $w(B_i^*)\leqq 3/8+1/8<1/3+1/5<s(B_i^*)$. Since the problem X_2 piece is not exceptional, $2s(X_2)+s(Y)>1$. Thus

$$\begin{aligned} w(B_i^*) &\leqq 3/8+1/4<(3/8)(2s(X_2)+s(Y))+1/4 \\ &< s(B_i^*)-(1/4)s(X_2)-(5/8)s(Y)+1/4 \\ &< s(B_i^*)-(1/4)s(X_2)-(5/8)(1-2s(X_2))+1/4 \\ &= s(B_i^*)+s(X_2)-3/8<s(B_i^*). \end{aligned}$$

Suppose the second piece of B_i^* was shifted when packed in B_j by FFDLS. This piece can only be of type X_1 or X_2, since a nonexceptional Y piece is not shifted if packed only with other Y pieces, and otherwise fills a bin so that there is no distinction between a shift and the repack step. Suppose the piece is of type X_1. $s(\text{this } X_1 \text{ piece})<2/3$ and thus $s(B_j)<8/9$. $2s(X_2)>s(B_j)$, else $w(B_i^*)\leqq w(X_1)+(3/8)(8/9)<s(B_i^*)$. But now we use the fact that $s(X_2)<(3/8)s(B_i^*)$ and find that

$$\begin{aligned} w(B_i^*) &\leqq (3/4)s(B_j)+(3/8)s(B_i^*) \\ &< (3/2)s(X_2)+(3/8)s(B_i^*) \\ &< (3/4)^2 s(B_i^*)+(3/8)s(B_i^*)<s(B_i^*). \end{aligned}$$

Suppose the second piece is of type X_2. The shifted piece is larger than the other, else, since the other cannot be shifted, either $w(B_i^*)\leqq(3/4)(3/8)+3/8<2/3<s(B_i^*)$ or the other piece is exceptional. $s(B_i^*)>8/9$, else $w(B_i^*)\leqq w(X_2)+(3/8)s(B_i^*)\leqq s(X_2)+1/3<s(B_i^*)$. Even so, $w(B_i^*)\leqq w(X_2)+(3/8)s(B_i^*)\leqq 7/8<8/9<s(B_i^*)$.

Suppose $|B_i^*|=3$. If $s(B_i^*)\leqq 3/4$, then it must contain two Y pieces, each of weight at most $1/8$ and $w(B_i^*)\leqq 3/8+2/8<1/3+2/5<s(B_i^*)$. Thus $s(B_i^*)>(4/3)c(\text{BIN})>8/9$, else BIN is not short. B_i^* cannot contain an X_1 nor three X_2 pieces. If it contains two Y pieces, $w(B_i^*)\leqq 3/8+1/2<8/9<s(B_i^*)$. The only remaining possibility is that it contains two X_2 pieces and a Y piece. $w(Y)>1/8$, else $w(B_i^*)\leqq 7/8<8/9<s(B_i^*)$. Thus the Y piece was available when the X_2 pieces were packed and $w(Y)\leqq(1/4)s(B_i^*)$. The smaller X_2 piece must have been shifted, since otherwise the Y piece would fit implying $w(B_i^*)\leqq(3/8)s(B_i^*)+5/16+(1/4)s(B_i^*)<s(B_i^*)$. It must have been shifted alone, else $w(B_i^*)\leqq 2(3/8)s(B_i^*)+(1/4)s(B_i^*)=s(B_i^*)$. In fact, it must have been shifted into a bin whose size exceeds 4/9, else $w(B_i^*)\leqq(3/8)s(B_i^*)+(3/4)(4/9)+(1/4)s(B_i^*)<s(B_i^*)$. But this means that the larger piece of BIN, which has size less than $5/12<4/9$ but was not shifted, must be even smaller than the smaller X_2 piece of B_i^*. Hence the available Y piece would have fit with the two X_2 pieces in BIN, which is impossible.

Suppose $|B_i^*|=4$. It must contain three Y pieces, only one of which may have a weight as large as $1/4$. Hence $w(B_i^*)\leqq 3/8+1/4+2(1/8)<1/3+3(1/5)<s(B_i^*)$.

Case II. $|\text{BIN}|=1$. We first show that there is no FFDLS bin containing a Y piece without an X_1 piece. Suppose otherwise. Then such a bin cannot precede BIN and

$c(\mathrm{BIN}) > 2/3$. Hence we can employ the fact that $s(p_n) > 1/6$ and find a smaller counterexample using the construction described in the previous case.

The function w is defined in Table 2. Exceptional pieces are again assigned weight zero. Some of the entries deserve attention. If an FFDLS bin contains an X_1 piece and two Y pieces, it is the smaller Y piece which receives weight zero. Each piece from a bin B_i with two X_2 pieces has size $\geqq (3/8)s(B_i)$ except for the second X_2 piece assigned to an exceptional bin. Note that $w(B_i) = (3/4)s(B_i)$ if B_i is not exceptional. Also, $w(p_i) \leqq s(p_i)$ unless p_i is an X_1 piece of weight $> 1/2$.

TABLE 2
The function w for Case II.

Contents of B_i	Piece weights (times $s(B_i)$)	Restrictions
X_1	3/4	$s(X_1) > 1/2$
X_1, X_2	1/2, 1/4	$s(X_1) > 1/2$, $s(X_2) > 1/3$
X_1, Y	3/4, 0	$s(X_1) \geqq 2/3$, $s(Y) > 1/6$
	2/3, 1/12	$s(X_1) < 2/3$, $s(Y) > 1/6$
X_1, Y, Y	2/3, 1/12, 0	$s(X_1) < 2/3$, $s(Y) > 1/6$
X_2	3/4	$s(X_2) \geqq (3/4)s(B_i)$
X_2, X_2	3/8, 3/8	$s(X_2) \geqq (3/8)s(B_i)$

The proof of Case II thus reduces to showing that even if such an X_1 piece is in B_i^*, $w(B_i^*) \leqq s(B_i^*)$. We first dispose of the possibility that B_i^* contains an X_2 piece. Suppose such is the case. Let B_j represent the FFDLS bin containing the X_1 piece. B_j cannot contain a second piece as large as the X_2 piece, else we can discard a bin from each packing by deleting the pieces of B_j and moving the X_2 piece, if it remains, from B_i^* to the position in the optimal packing formerly occupied by the second piece of B_j, contradicting minimality since $s(B_i^*) > 5/6 > 3/4$. Nor can B_j contain a second piece smaller than the X_2 piece, else FFDLS has packed the X_2 piece with an earlier X_1 piece whereby our construction again illustrates the violation of minimality. We conclude that FFDLS packs the X_1 piece alone in B_j, though the X_2 piece is available and would fit. Hence $w(X_1) \leqq s(X_1)$, since the X_1 piece must have been shifted.

Suppose $|B_i^*| = 1$. Then clearly minimality requires that FFDLS pack the X_1 piece with another piece. If $s(\text{the } X_1 \text{ piece}) \geqq 2/3$, then $w(B_i^*) \leqq 3/4 < (4/3)(2/3) \leqq (4/3)s(X_1) < s(B_i^*)$ since the X_1 piece was not shifted alone. If $s(X_1) < 2/3$, then similarly $w(B_i^*) \leqq 2/3 = (4/3)(1/2) < (4/3)s(X_1) < s(B_i^*)$.

Suppose $|B_i^*| = 2$. $s(X_1) < 2/3$, else $w(B_i^*) \leqq 3/4 + 1/12 = 2/3 + 1/6 < s(B_i^*)$. Let B_j denote the FFDLS bin containing the X_1 piece. $|B_j| > 1$, else $w(B_i^*) \leqq (3/4)s(B_j) + 1/12 \leqq (3/4)s(B_i^*) + 1/12 < s(B_i^*)$. Let a and b denote the second largest pieces of B_j and B_i^* respectively. It must be that $a > b$, else either $|B_j| = 2$ and $w(B_i^*) \leqq (2/3)s(B_i^*) + 1/12 < s(B_i^*)$ or $|B_j| = 3$ and, lest the X_1 and a be shifted to a bin no larger than $s(B_i^*)$, $w(B_i^*) \leqq 2/3 + 1/12 < (4/3)(2/3) < (4/3)(s(X_1) + s(a)) < s(B_i^*)$. Suppose b is packed in $B_{j'}$, $j' \leqq j$. Then either b is the third piece of $B_{j'}$ or $B_{j'}$ contains an item whose size exceeds $2/3$ since a was not used instead. Thus $w(B_i^*) \leqq 2/3 < s(B_i^*)$. Suppose b is packed in $B_{j'}$, $j < j'$. Let B_i^* denote the optimal bin containing the X_1 piece from $B_{j'}$. $w(B_i^*) \leqq (2/3)s(B_i^*) + 2(1/12)$ since the first two pieces of $B_{j'}$ would have been shifted to a bin no larger than $s(B_i^*)$. This also implies that $w(b) \leqq (1/12)s(B_i^*)$. Furthermore, $2/3 < s(B_i^*)$, else the X_1 piece of $B_{j'}$ would have been

shifted alone. The weights of these two bins now balance, since

$$w(B_i^*)+w(B_{i'}^*)\leqq 2/3+(1/12)s(B_i^*)+(2/3)s(B_{i'}^*)+1/6$$
$$<s(B_{i'}^*)+(3/4)s(B_i^*)+1/6<s(B_{i'}^*)+s(B_i^*).$$

Note that $B_{i'}^*$ is used to balance its weight with that of B_i^* only, since there is but one Y piece in $B_{j'}$.

Suppose $|B_i^*|=3$. Then $s(X_1)<2/3$. FFDLS cannot pack the X_1 piece by itself, else $w(B_i^*)\leqq(3/4)s(B_i^*)+2/12<s(B_i^*)$. Thus $w(B_i^*)\leqq 2/3+2/12=1/2+2/6<s(B_i^*)$. □

The careful reader may suspect that the additive constant is really very pessimistic. We believe that a more detailed analysis will reduce it to 1.

To demonstrate the tightness of the 4/3 asymptotic ratio, choose any packing instance with an even number of pieces of size 1/2 and bins of size 1 and 2/3.

5. Directions for future research. Worst-case results are especially useful here since any attempt at establishing analytical average-case figures will be doubly dependent on the input considered (one probability distribution for bin sizes and a second, possibly distinct, distribution for piece sizes). Nonetheless, we remark that average-case analysis is an active field of research (see for example [OMW]) and techniques may be brought to bear on this problem. It is possible that FFDLR surpasses FFDLS in this respect since FFDLS was constructed solely to establish the 4/3 bound.

A natural research goal is to devise an algorithm whose worst-case ratio is less than 4/3. Some heuristic designed along the line of FFDLS may well succeed. We have found packing the largest bins first to be an attractive approach, thus providing at least a partial answer to a question posed in [La].

Variable sized bins could be permitted in other bin packing models such as multidimensional packing [BCR], dual packing [AJKL] or on-line packing [Br] (note that NFL is on-line with respect to both bins and pieces). From a more theoretical standpoint, the problem may even allow an "approximation scheme" (see for example [KK]).

REFERENCES

[AJKL] S. F. ASSMANN, D. S. JOHNSON, D. J. KLEITMAN AND J. Y-T. LEUNG, *On a dual version of the one-dimensional bin packing problem*, J. Algorithms, to appear.

[BCR] B. S. BAKER, E. G. COFFMAN, JR. AND R. L. RIVEST, *Orthogonal packings in two dimensions*, this Journal, 9 (1980), pp. 846–855.

[Br] D. J. BROWN, *A lower bound for on-line one-dimensional bin packing algorithms*, CSL Tech Rpt R-864 (1979), Univ. Illinois, Urbana, IL.

[CGJ] E. G. COFFMAN, JR., M. R. GAREY AND D. S. JOHNSON, *Approximation algorithms for bin-packing—an updated survey*, to appear.

[Co] E. G. COFFMAN, JR., *An introduction to proof techniques for packing and sequencing algorithms*, in Deterministic and Stochastic Scheduling, M. A. H. Dempster, et al., eds., Reidel Publishing Co., Amsterdam, 1982, pp. 245–270.

[FL1] D. K. FRIESEN AND M. A. LANGSTON, *Bounds for* MULTIFIT *scheduling on uniform processors*, this Journal, 12 (1983), pp. 60–70.

[FL2] ———, *A storage-size selection problem*, Inform. Proc. Letters, 18 (1984), pp. 295–296.

[GG] P. C. GILMORE AND R. E. GORMORY, *A linear programming approach to the cutting stock problem*, Operation Res., 9 (1961), pp. 849–859.

[JDUGG] 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.

[Jo] D. S. JOHNSON, *Fast algorithms for bin packing*, J. Comput. Syst. Sci., 8 (1974), pp. 272–314.

[KK] N. KARMARKAR AND R. M. KARP, *An efficient approximation scheme for the one-dimensional bin packing problem*, Proc. 23rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1982.

[La] M. A. LANGSTON, *Performance of heuristics for a computer resource allocation problem*, SIAM J. Alg. Disc. Meth., 5 (1984), pp. 154–161.

[OMW] H. L. ONG, M. J. MAGAZINE AND T. S. WEE, *Probabilistic analysis of bin packing heuristics*, Operations Res., 32 (1984), pp. 983–998.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics

LOGARITHMIC DEPTH CIRCUITS FOR ALGEBRAIC FUNCTIONS*

JOHN H. REIF†

Abstract. This paper describes circuits for computation of a large class of algebraic functions on polynomials, power series, and integers, for which, it has been a long standing open problem to compute in depth less than $\Omega(\log n)^2$.

Algebraic circuits assume unit cost for elemental addition and multiplication. This paper describes $O(\log n)$ depth algebraic circuits which given as input the coefficients of n degree polynomials (over an appropriate ring), compute the product of $n^{O(1)}$ polynomials, the symmetric functions, as well as division and interpolation of real polynomials. Also described are $O(\log n)$ depth algebraic circuits which are given as input the first n coefficients of a power series (over an appropriate ring) compute the product of $n^{O(1)}$ power series, as well as division, reciprocal and reversion of real power series.

Furthermore this paper describes boolean circuits of depth $O(\log n(\log\log n))$ which, given n-bit binary numbers, compute the product of n numbers and integer division. As corollaries, we get boolean circuits of the same depth for evaluating, within accuracy 2^{-n}, polynomials, power series, and elementary functions such as (fixed) powers, roots, exponentiations, logarithm, sine and cosine.

All these circuits have constant indegree, polynomial size, and may be uniformly constructed by a deterministic Turing machine with space $O(\log n)$.

Key words. algebraic computation, circuit depth, parallel computation, integer division, powering

1. Introduction. Much research is now done on parallel algorithms, although in fact at this time most current computers contain only a single processor. However, these computers do use parallel circuits to implement the most basic and often repeated operations, such as the arithmetic operations: addition, subtraction, multiplication and division. These operations are generally applied to integers with an n bit binary representation, and to floating point reals with relative accuracy 2^{-n}. Other frequently used repeated operations, which certainly would merit special purpose circuits, are the elementary functions such as sine, cosine, arctangent, exponentation, logarithm, square roots, and fixed powers. For practical reasons we require circuits of constant indegree which can be uniformly constructed within $O(\log n)$ deterministic space (and thus deterministic polynomial time).

The *depth* of a circuit is the time for its parallel execution. What is the minimum depth of boolean circuits for these arithmetic operations and elementary functions?

For integer addition, Ofman [62], Krapchenko [67] and Ladner and Fischer, [80] give boolean circuits of depth $O(\log n)$ and size $O(n)$. Subtraction circuits with the same asymptotic depth and size can easily be gotten from these addition circuits. Also Reif [83] has recently given linear size, constant indegree boolean circuits of depth $O(\log\log n)$ for addition and subtraction of random numbers with error probability at most n^{-c}.

For integer multiplication, Ofman [62] and Wallace [64] give boolean circuits of depth $O(\log n)$, and Schönhage and Strassen [71] also achieve depth $O(\log n)$ with simultaneous size $O(n(\log n)\log\log n)$.

The problem of computing division or the elementary functions in better then depth $\Omega(\log n)^2$ has been open for at least 17 years since S. Cook's Ph.D. thesis (Cook [66]) (also see Borodin and Munro [75], and Savage [76]). Wallace [64] first gave a

* Received by the editors, May 5, 1983, and in final revised form October 11, 1984. This work was supported in part by the National Science Foundation under grant NSF-MCS82-00269 and the Office of Naval Research under contract N00014-80-C-0647.

† Aiken Computation Laboratory, Division of Applied Sciences, Harvard University, Cambridge, Massachusetts 02138.

division circuit with depth $\Omega(\log n)^2$. Subsequently, Anderson et al. [67] gave a division circuit of the same depth which was implemented by them on the IBM/360 Model 91 Floating-Point Execution Unit. Knuth [81] and Aho, Hopcroft and Ullman [74] described a division circuit attributed to Steve Cook of depth $(\log n)^2$ and size $O(n \log n \log\log n)$. The best known boolean circuit depth for the elementary functions was $\Omega(\log n)^2$ (Brent [76], Kung [76]). Many of the above mentioned boolean circuits of depth $\Omega(\log n)^2$ for division and elementary functions use a second order Newton iteration with $\Omega(\log n)$ steps, each requiring an n-bit integer multiplication with $\Omega(\log n)$ depth. Alternatively, a reduction is often made to the problem of computing the mth power of a n-bit integer modulo 2^n+1 for $m = O(n)$. This can be computed by $\Omega(\log n)$ steps of repeated squaring, where each square computation requires $\Omega(\log n)$ depth.

By new techniques we achieve depth less than $(\log n)^2$. An essential technique in the construction of our circuits is the use of convolutions, which can be computed in boolean depth $O(\log n)$ by the fast Fourier transforms. This technique was first introduced by Schönhage and Strassen [71] for the multiplication of two integers. Our innovation was to generalize the convolution technique to products of more than two terms.

Section 2 introduces the appropriate mathematical groundwork for the generalized polynomial convolution techniques which we utilize. Also in § 2 we give $O(\log n)$ depth algebraic circuits for various polynomial and power series operations. These algebraic circuits are interesting in the theoretical context of parallel algebraic computation, where arithmetic operations are assumed to be of unit cost.

The last part of this paper is concerned with the possibly more practical construction of boolean circuits, which originally motivated this work. In § 3 we give uniform boolean circuits of nearly logarithmic depth for the problem computing the product of $n^{O(1)}$ integers modulo (2^n+1). In an earlier version of this paper (Reif [83]) we proved our boolean circuits had depth $O(\log n(\log\log n)^2)$. This draft includes an improvement to our construction due to Beame, Cook, and Hoover [84a, b] which reduces the depth by a factor of $\log\log n$ to $O(\log n(\log\log n))$ and gives simultaneously polynomial size. These results imply uniform boolean circuits of depth $O(\log n(\log\log n))$ for the problems of division and computing elementary functions, among others.

This also implies sequential space complexity upper bounds for these and related problems. In particular, Borodin [77] proved that if a function f is computed in uniform boolean circuit depth $D(n) \geqq \log n$, then f can be computed by a deterministic Turing machine with space $D(n)$. Thus for example, division, the elementary functions and the first n bits of π can be computed by deterministic space $O(\log n(\log\log n))$.

2. Circuits for polynomial and power series computations. Our basic techniques are best understood first in the simpler context of polynomials and power series. In fact, this context is interesting in itself. We might envision a special purpose computer designed for algebraic computation. Its data are (coefficients of) polynomials and power series. The arithmetic operations including division of polynomials and power series are elementary operations of our "algebraic computer." Also, frequently applied operations are the composition of power series, revision of a power series, computation of elementary functions applied to power series, and interpolation of polynomials. We give in this section circuits of depth $O(\log n)$ for all these polynomial and power series operations, where each gate of the circuits computes an addition, multiplication, or a division of two elements of the domain.

2.0. Circuit definitions. A *circuit* α_N over a commutative ring $\mathcal{R} = (\mathcal{D}, +, \cdot, 0, 1)$ is an acyclic labeled digraph, with

(i) a list of N distinguished *input nodes* that have no entering edges;
(ii) *constant nodes* with indegree 0 and labeled with constants in $\mathcal{D}$;
(iii) *internal nodes* with indegree two and labelled with the symbols in {"+", "·"};
(iv) a list of N' distinguished *output nodes.*

Given an assignment of the input nodes from domain $\mathcal{D}$, the value of the circuit at the output nodes is gotten by evaluation of the gates in topological order. The circuit α_N thus defines a mapping from $\mathcal{D}^N$ to $\mathcal{D}^{N'}$. A circuit α_N over a field is similarly defined, except the internal nodes can also compute division. Since division may yield an undefined value, a circuit over a field defines in general a partial mapping of inputs to outputs.

Let $\mathcal{R}[x]$ be the polynomials over commutative ring $\mathcal{R}$. Let $\mathcal{R}[[x]]$ be the power series over $\mathcal{R}$.

Let f be a partial function of (the coefficients of) m polynomials $p_1(x), \cdots, p_m(x)$ in $\mathcal{R}[x]$ of degree $n-1$. A circuit α_N for f has $N = mn$ inputs, namely the list of N coefficients in $\mathcal{D}$ of the given polynomials. The output nodes of α_N give the list of coefficients of $f(p_1(x), \cdots, p_m(x))$. If on the other hand f is a function of m power series $p_1(x), \cdots, p_m(x)$ in $\mathcal{R}[[x]]$ each with n given low order coefficients, then the circuit α_N for f also has $N = nm$ inputs, and the outputs nodes of α_N only give some prescribed finite number of the coefficients of (the possibly infinite) power series $f(p_1(x), \cdots, p_m(x))$.

The *depth* of circuit α_N is the length of its longest path. A partial function f over polynomials or power series in $\mathcal{R}$ has simultaneous *depth* $O(D(N))$ and *size* $O(S(N))$ if there exists an infinite family of circuits $\alpha_1, \cdots, \alpha_N, \cdots$ and constants $c_1, c_2 \geqq 1$ such that $\forall N \geqq 1$, α_N has depth not more than $c_1 D(N)$ and size not more than $c_2 S(N)$ and given N input coefficients of the input polynomial or power series, α_N computes f within the prescribed number of coefficients.

Let $\alpha_1, \alpha_2, \cdots$ be a family of circuits over $\mathcal{R} = (\mathcal{D}, +, \cdot, 0, 1)$ where $\mathcal{D}$ is countable. Fix some enumeration $c_1, c_2, \cdots$ of the constants in $\mathcal{D}$. We assume each circuit α_N is encoded by a binary string where the binary representation of i is used to represent each constant symbol c_i labeling a node in α_N. (Thus, for example, the Nth root of unity, if it exists, might be represented by a binary string of length log N.) The circuit family $\alpha_1, \cdots, \alpha_N, \cdots$ is *uniform* in the sense of Borodin [77] if there exists a logarithmic space deterministic Turing machine which given any $N > 0$ in unary outputs for the binary encoding of α_N. All the circuits considered in this paper are uniform in this sense.

2.1. The discrete Fourier transform. Fix a commutative ring $\mathcal{R} = (\mathcal{D}, +, \cdot, 0, 1)$. We assume ω is a principle Nth root of unity in $\mathcal{R}$ and that N has a multiplicative inverse. (For example, $e^{2\pi\sqrt{-1}/N}$ is a principle Nth root of unity in the complex numbers.) Given a vector $a \in \mathcal{D}^N$, the Discrete Fourier Transform is

$$\mathrm{DFT}_N(a) = Aa$$

where $A_{ij} = \omega^{ij}$ for $0 \leqq i, j < N$. Then A^{-1} exists (Aho, Hopcroft and Ullman [74, p. 253]), where $A_{ij}^{-1} = (1/N)\omega^{-ij}$. The inverse Discrete Fourier Transform is $\mathrm{DFT}_N^{-1}(a) = A^{-1}a$ and obviously satisfies $\mathrm{DFT}_N^{-1}(\mathrm{DFT}_N(a)) = a$. (*Note*: given a vector $a \in \mathcal{D}^n$, where $n < N$, $\mathrm{DFT}_N(a)$ we be defined to be $\mathrm{DFT}_N(a^+)$ where a^+ is the vector of length N derived by concatenating a with $N-n$ zeros.) Cooley and Tukey [65] gave the Fast Fourier Transform for which

THEOREM 2.1. DFT_N *and* DFT_N^{-1} *over* $\mathcal{R}$ *have simultaneous depth* $O(\log N)$ *and size* $O(N \log N)$.

Note: the assumption of the nth root of unity is not really essential to our techniques, since in general, our techniques will be applicable whenever a $O(\log n)$ depth circuit exist for the Discrete Fourier Transform. For example, Theorem 2.1 obviously applies to the complex numbers, and since the field operations over complex numbers can be simulated over the reals with only a factor of two depth increase, Theorem 2.1 also applies to the reals.

2.2. Products of polynomials. Suppose we are given m vectors $a_i \in \mathcal{D}^n$ for $i = 1, \cdots, m$. Each vector $a_i = (a_{i,0}, \cdots, a_{i,n-1})^T$ gives the coefficients of a $n-1$ degree polynomial $A_i(x) = \sum_{j=0}^{n-1} a_{i,j}x^j$ in $\mathcal{R}[x]$. Let $N = nm$. We wish to compute the product polynomial $B(x) = \sum_{k=0}^{N-1} b_k x^k$, where $B(x) = \prod_{i=1}^{m} A_i(x)$. (Note that we have $b_k = 0$ for $N-m+1 \leqq k \leqq N-1$.)

In the special case $m = 2$ and $N = 2n$, the convolution vector $b = (b_0, \cdots, b_{N-1})^T = a_1 \circledast a_2$ gives the coefficients of $B(x)$. By the Convolution Theorem:

$$a_1 \circledast a_2 = \mathrm{DFT}_N^{-1}(\mathrm{DFT}_N(a_i) \cdot \mathrm{DFT}_N(a_2)) \text{ where } \cdot \text{ denotes pairwise product.}$$

Hence the well-known result:

THEOREM 2.2. *The product of two polynomials in* $\mathcal{R}[x]$ *of degree* $n-1$ *has simultaneous depth* $O(\log n)$ *and size* $O(n \log n)$.

In the case of general $m \geqq 2$, we wish to compute the coefficient vector

$$b = (b_0, \cdots, b_{N-1})^T = a_1 \circledast \cdots \circledast a_m.$$

By repeated application of the Convolution Theorem we get

LEMMA 2.1. $b = \mathrm{DFT}_N^{-1}(\mathrm{DFT}_N(a_1) \cdots \mathrm{DFT}_N(a_m))$.

First in parallel for $i = 1, \cdots, m$ compute $f_i = \mathrm{DFT}_N(a_i)$, where $f_i = (f_{i,0}, \cdots, f_{i,N-1})^T$. Next we compute in parallel for $j = 10, \cdots, N-1$ the elementary products $F_j = \prod_{i=1}^{m} f_{i,j}$. Finally, we compute $\mathrm{DFT}_N^{-1}((F_0, \cdots, F_{N-1})^T)$. Since the computation of DFT_N, DFT_N^{-1} and the required products F_j, each have depth $O(\log N)$, we have:

THEOREM 2.3. *The product of* m *polynomials in* $\mathcal{R}[x]$ *of degree* $n-1$ *has depth* $O(\log(nm))$.

Note that in contrast, the naive method of repeated producting by Theorem 2.2 has depth $(\log(m)\log(n))$. Also note that since Theorem 2.1 applies to the real polynomials so do Theorems 2.2 and 2.3.

2.3. Modular products of polynomials. Let $B(x) = \prod_{i=1}^{m} a_i(x)$ be the product polynomial considered in the previous section. Here we consider the computation of the *modular product* $D(x) = \sum_{i=0}^{n-1} d_i x^i$ where $D(x) \equiv B(x) \bmod (x^n+1)$.

LEMMA 2.2. *The coefficients of* $D(x)$ *are* $d_i = \sum_{r=0}^{m-1} (-1)^r b_{nr+i}$ *for* $i = 0, \cdots, n-1$.

Proof.

$$B(x) = \sum_{j=0}^{N-1} b_j x^j = \sum_{r=0}^{m-1} \sum_{i=0}^{N-1} b_{nr+i} x^{nr+i}$$

$$\equiv \sum_{r=0}^{m-1} (-1)^r b_{nr+i} x^i \bmod (x^n+1)$$

since $(-1)^r \equiv x^{nr} \bmod (x^n+1)$.

We assume ω is a principal nth root of unity in $\mathcal{R}$ and n has a multiplicative inverse. We also assume there exists an $\psi \in \mathcal{D}$ such that $\psi^2 = \omega$ and $\psi^n = -1$. Let

$\hat{a}_i = (a_{i,0}, \psi a_{i,1}, \cdots, \psi^{n-1} a_{i,n-1})^T$. The *negatively wrapped convolution* of $a_1, \cdots, a_m$ is

$$\hat{d} = (d_0, \psi d_1, \cdots, \psi^{n-1} d_{n-1})^T.$$

LEMMA 2.3. $\hat{d} = \mathrm{DFT}_n^{-1}(\mathrm{DFT}_n(\hat{a}_1) \cdots \mathrm{DFT}_n(\hat{a}_m))$.

Proof. For $i = 1, \cdots, m$ let $\mathrm{DFT}_n(\hat{a}_i) = (g_{i,0}, \cdots, g_{i,n-1})^T$ where

$$g_{i,k} = \sum_{j=0}^{n-1} a_{i,j} \psi^j \omega^{jk}$$

for $k = 0, \cdots, n-1$. Let

$$e_k = \left(\prod_{i=1}^{m} g_{i,k} \right) = \sum_{0 \le j_1, \cdots, j_m < n} \psi^{\sum j_i} \omega^{k(\sum j_i)} \left(\prod_{i=1}^{m} a_{i,j_i} \right).$$

Now let $\mathrm{DFT}_n(\hat{d}) = (e'_0, \cdots, e'_{n-1})^T$. Then for $k = 0, \cdots, n-1$ we get

$$\begin{aligned} e'_k &= \sum_{h=0}^{n-1} d_h \psi^h \omega^{kh} \\ &= \sum_{h=0}^{n-1} \sum_{r=0}^{m-1} \psi^h \omega^{kh} (-1)^r b_{nr+h} \quad \text{by Lemma 2.2} \\ &= \sum_{h=0}^{n-1} \sum_{r=0}^{m-1} \psi^h \omega^{kh} (-1)^r \sum_{\substack{0 \le j_1, \cdots, j_m < n \\ nr+h = \sum j_i}} \prod_{i=1}^{m} a_{i,j_i}. \end{aligned}$$

But if we substitute $h = (\sum_{i=1}^{m} j_i) - nr$ into the above expansion, we get

$$\psi^h \omega^{kh} (-1)^r = \psi^{\sum j_i} \omega^{k(\sum j_i)}$$

since $\psi^{nr} = (-1)^r$ and $\omega^n = 1$. Hence $e'_k = e_k$.

The above Lemmas 2.2, 2.3 and Theorem 2.1 imply:

THEOREM 2.4. *The modular product* $(A_1(x) \cdots A_m(x)) \bmod (x^n + 1)$ *of polynomials* $A_1(x), \cdots, A_m(x)$ *in* $\mathcal{R}[x]$ *of degree* $n-1$ *has simultaneous depth* $O(\log(nm))$ *and size* $O(nm \log(nm))$. *The modular power* $A(x)^m \bmod (x^n + 1)$ *of a single polynomial* $A(x)$ *of degree* $n-1$ *has simultaneous depth* $O(\log(nm))$ *and size* $O(n \log(nm))$.

2.4. Elementary functions of power series. An immediate consequence of Theorem 2.3 is

COROLLARY 2.1. *The composition of two power series in* $\mathcal{R}[[x]]$ *has depth* $O(\log n)$.

The elementary functions $\exp(x)$, $\log(x)$, $\sin(x)$, $\cos(x)$, $\arctan(x)$, and square root (x), etc. all have known Taylor series expansions convergent over given intervals. Thus by Corollary 2.1 we have

COROLLARY 2.2. *The elementary functions on* $\mathcal{R}[[x]]$ *have depth* $O(\log n)$.

For some given $x_1, \cdots, x_N \in \mathcal{D}^N$ it is frequently useful in algebraic computations to determine the polynomial $\prod_{i=1}^{N} (y - x_i) = \sum_{j=0}^{N} (-1)^j p_j y^j$ whose coefficients $p_j = \sum_{i_1 < i_2 < \cdots < i_j} x_{i_1} \cdots x_{i_j}$ are the elementary symmetric functions. It was pointed out to us by Les Valiant that Theorem 2.3 immediately implies

COROLLARY 2.3. *The elementary symmetric functions in* $\mathcal{R}[[x]]$ *have depth* $O(\log N)$.

2.5. Division, interpolation and reversion. Let $A(z) = \sum_{i=0}^{n-1} a_i z^i$ be a real power series where $a_0 = 1$. The reciprocal of $A(z)$ is the power series $I(z) = \sum_{i=0}^{\infty} r_i z^i$ such that $A(z) \cdot I(z) = 1$. $I(z)$ has the infinite series expansion

$$I(z) = \sum_{i=0}^{\infty} (1 - A(z))^i.$$

We wish to compute the first n coefficients of $I(z)$. Since $I(z) = \sum_{i=0}^{n-1}(1-A(z))^i + O(z^n)$, we have by Theorem 2.3:

COROLLARY 2.4. *The first n terms of the reciprocal of a real power series and the division of two real power series can be computed in depth* $O(\log n)$.

An alternative method using the lemma below results in a circuit of depth $O(\log n)$ with smaller circuit size.

LEMMA 2.4. *If*

$$\tilde{I}(z) = \prod_{i=0}^{\lceil \log(n+1)\rceil - 1}(1+(1-A(z))^{2^i})$$

then $I(z) - \tilde{I}(z) = O(z^{n+1})$.

Proof. Let $B(z) = 1 - A(z)$. Then $A(z)\tilde{I}(z) = (1-B(z))\tilde{I}(z) = 1 - B(z)^{\bar{n}} = 1-(1-A(z))^{\bar{n}}$, where $\bar{n} = 2^{\lceil \log(n+1)\rceil}$. So

$$I(z) - \tilde{I}(z) = \frac{(1-A(z))^{\bar{n}}}{A(z)} = O(z^{\bar{n}}) = O(z^{n+1}). \qquad \square$$

COROLLARY 2.5. *Given real polynomials* $a(x)$, $b(x)$ *of degree at most n, we can compute in depth* $O(\log n)$ *the unique polynomials* $q(x)$, $r(x)$ *such that* $a(x) = q(x)b(x) + r(x)$ *and* degree $(r(x)) <$ degree $(b(x))$.

Proof. (Also, see Knuth [81].) Let $n_1 =$ degree $(a(x))$ and $n_2 =$ degree $(b(x))$. The computation is trivial unless $n_1 \geqq n_2 \geqq 1$. Then

$$A(z) = Q(z)B(z) + z^{n_1 - n_2 + 1}R(z)$$

where

$$A(z) = z^{n_1}a\left(\frac{1}{z}\right), \quad B(z) = z^{n_2}b\left(\frac{1}{z}\right), \quad Q(z) = z^{n_1-n_2}q\left(\frac{1}{z}\right)$$

and

$$R(z) = z^{n_2-1}r\left(\frac{1}{z}\right).$$

Thus to compute the coefficients of $q(x)$, $r(x)$ we compute the first $n_1 - n_2 + 1$ coefficients of $A(z)/B(z) = Q(z) + o(z^{n_1-n_2+1})$, then compute the power series $A(z) - B(z)Q(z) = z^{n_1-n_2+1}R(z)$, and finally output the coefficients of $Q(z)$, $R(z)$. $\square$

COROLLARY 2.6. *Interpolation of a real polynomial has depth* $O(\log n)$.

Proof. Suppose we are given real polynomials $p_1(x), \cdots, p_m(x)$ each of degree $n-1$, and real polynomials $q_1(x), \cdots, q_m(x)$ where degree $(q_i(x)) <$ degree $(p_i(x))$ for $i = 1, \cdots, n$. Let $P(x) = \prod_{i=1}^{m} p_i(x)$. The Chinese Remainder Theorem states that there is a unique polynomial $Q(x)$ of degree less than that of $P(x)$ such that $Q(x) \equiv q_i(x) \bmod p_i(x)$ for $i = 1, \cdots, m$, coprimality of the p_i assumed.

The Lagrangian interpolation formula gives

$$Q(x) \equiv \sum_{i=0}^{m} q_i(x)r_i(x)s_i(x) \bmod P(x)$$

where $s_i(x) = P(x)/p_i(x)$ and $r_i(x)$ is the multiplicative inverse of $s_i(x) \bmod p_i(x)$.

Theorem 2.2 and Corollary 2.5 imply that preconditioned Chinese remaindering, with the $r_1(x), \cdots, r_m(x)$ also given, has depth $O(\log n)$.

However, in the special case $p_i(x) = x - a_i$ for $i = 1, \cdots, m$, where the a_i are distinct then each $r_i(x) = 1/s_i(x)$ can be computed in parallel by Theorem 3.3 and Corollary

2.5 in depth $O(\log n)$. In this case the $q_i(x) = b_i$ are constants, since they must have degree less than the $p_i(x)$.

Further note that in this case $Q(x)$ is the unique polynomial such that $Q(a_i) = b_i$ for $i = 1, \cdots, m$. Thus we have proved Corollary 2.6. □

We now show that Theorem 2.3 and Corollary 2.4 imply:

COROLLARY 2.7. *The reversion of a real power series has depth* $O(\log n)$.

Proof. Let $A(x) = \sum_{i=0}^{\infty} a_i x^i$ be a real power series where $a_0 = 0$ and $a_1 = 1$. The *reversion* of $A(x)$ is the power series $R(z) = \sum_{k=0}^{\infty} r_k z^k$ where $z = A(x)$ iff $x = R(z)$. Note that $r_0 = 0$ and $r_1 = 1$. For the kth coefficient, we first compute

$$B(x) = \frac{1}{A(x)^k} = \sum_{i=0}^{\infty} b_{i+1} x^i,$$

and then apply Lagrange's reversion formula (Lagrange, [1768]) $r_k = b_{k-1}/k$ for $k \geqq 2$. Thus Theorem 3.3 implies Corollary 2.7. □

3. Integer computations.

3.0. Boolean circuits. We consider computations over integers given as n bit binary numbers, and reals over $[0, 1]$ given within accuracy 2^{-n}. Our computational model in this section is the *boolean circuit*, defined as usual. The ith input node of α_n takes the ith bit of the encoding of the input integer or real. Each gate of α_n computes a boolean operation $\vee$, $\wedge$, or $\neg$. Each output node provides a bit of the encoding of the computed integer or real. (In the case of reals with floating point representation, we only provide the input and output bits up to some finite prescribed accuracy.)

3.1. The DFT over an integer ring. We assume n and ω are positive powers of two. Let $p = \omega^{n/2} + 1$ and let $\mathbb{Z}_p$ be the ring of integers modulo p.

PROPOSITION 3.1. *In* $\mathbb{Z}_p$, ω *is a principal nth root of unity and n has a multiplicative inverse* modulo p.

Proposition 3.1 implies that DFT_n and DFT_n^{-1} are well defined.

The fast Fourier transform computation of Cooley and Tukey [65] yields an arithmetic circuit α_n of depth $O(\log n)$ and size $O(n \log n)$ computing DFT_n whose elements require:

(i) addition of two $\lceil \log(p) \rceil$-bit integers.

(ii) multiplication of a $\lceil \log(p) \rceil$-bit integer by a power of ω.

We wish to expand α_n into a boolean circuit. Since ω is a power of two, the multiplications can be implemented by the appropriate bit shifts (i.e., the gate connections are shifted by the appropriate amount). The additions can be implemented by Carry-Save Add circuitry of Ofman [62] and Wallace [64] (also see Savage [76]) yielding a boolean circuit of depth $O(\log(n \log p))$ and size $O(n \log p \log(n \log p))$. Thus we have

THEOREM 3.1. DFT_n *and* DFT_n^{-1} *over the ring* $\mathbb{Z}_p$ *have simultaneous boolean depth* $O(\log(n \log p)) = O(\log n)$ *and size* $O(n \log p \log(n \log p))$.

3.2. Products of integers. Schönhage, Strassen [71] have shown:

THEOREM 3.2. *The product of two N-bit integers has simultaneous boolean depth* $O(\log N)$ *and size* $O(N \log N \log\log N)$.

We now prove that for N a power of two, the modulo $2^N + 1$ product of m integers, each of N-bits has boolean depth $O(\log(Nm) \log\log N)$. (Note that the naive method of repeated squaring by Theorem 3.2 results in a boolean circuit of depth $\Omega(\log(m) \log N)$.) We begin with a key lemma which reduces the number of bits of the integers to be produced.

LEMMA 3.1. (DFT reduction). *For N a power of two, mN sufficiently large, and any $m < N^{1/2}$ the product modulo (2^N+1) of m integers each N bits long can be computed in $O(\log mN)$ additional boolean depth and $(mN)^{O(1)}$ additional gates after computing $n = O(mN)^{1/2}$ products modulo (2^n+1) each of m integers each n bits long.* □

Proof. Let $a_1, \cdots, a_m$ be a list of N-bit numbers. We wish to compute $b = \prod_{i=1}^{m} a_i \bmod (2^N+1)$.

(1) Since $N = 2^u$ for some integer u, we can block each N-bit a_i into n (n a power of 2) chunks $a_{i,0}, \cdots, a_{i,n-1}$ of $h = N/n$ bits each so that

$$a_i = \sum_{j=0}^{n-1} a_{i,j} 2^{hj}$$

where $0 \leqq a_{i,j} < 2^h$. Define the associated polynomial

$$A_i(x) = \sum_{j=0}^{n-1} a_{i,j} x^j$$

and observe $a_i = A_i(2^h)$.

(2) We intend to take DFT_n with $\omega = 4$, $\psi = 2$ and $p = \omega^{n/2}+1 = 2^n+1$. Associate with each a_i a coefficient vector $\hat{a}_i$ defined by

$$\hat{a}_i \equiv (a_{i,0}, \psi a_{i,1}, \cdots, \psi^{n-1} a_{i,n-1})^T.$$

(3) Compute in parallel

$$\mathrm{DFT}_n(\hat{a}_i) = \hat{g}_i \equiv (g_{i,0}, \cdots, g_{i,n-1})^T.$$

(4) Compute product

$$e_k \equiv \prod_{i=1}^{m} g_{i,k} \bmod p.$$

(5) Compute

$$\mathrm{DFT}_n^{-1}((e_0, \cdots, e_{n-1})^T) = \hat{b} \equiv (b_0, \psi b_1, \cdots, \psi^{n-1} b_{n-1})^T$$

to obtain the coefficients of the product polynomial

$$B(x) = \sum_{j=0}^{n-1} b_j x^j$$

where by Lemma 2.3, $b = B(2^h)$.

(6) Evaluate $B(2^h)$ to get b.

Since ψ is a power of two, we can easily extract each b_j from $\psi^{j-1} b_j$ by bit shifting. By Theorem 3.1, the DFT_n and DFT_n^{-1} computations have depth $O(\log n)$. Thus all of these computations have depth $O(\log N + \log h + \log n) = O(\log mN)$ except possibly computing the e_k modular product in step (4). Note that we can use the identity $2^n x \equiv (2^n + 1 - x) \bmod 2^n + 1$ to simplify the computation of e_k to the product of at most m numbers, each of n bits. Thus the depth $D(m, N)$ of the resulting circuit satisfies

$$D(m, N) = D(m, N) + O(\log mN).$$

The reduction is correct if n is a power of two and furthermore the coefficients of $B(x)$ are small, that is if $|b_i| < p/2$. Applying Proposition 3.2, we can ensure $|b_i| < 2^{n-1}$ by having $N \geqq 16$, $m \leqq N^{1/2}$ and choosing n to be the largest power of 2 less than $16(mN)^{1/2}$. □

PROPOSITION 3.2. *For each $j = 0, \cdots, n-1$, the magnitude of the coefficients of $B(x)$ is given by $|b_i| < 2^{2m(h+1+\log n)}$.*

Proof. Let $f(i)$ be the maximum magnitude of any coefficient of a polynomial resulting from a product of 2^i of the $A_j(x)$ polynomials taken mod (x^n+1). Clearly $f(0) \leqq 2^h$ and $f(i) \leqq 2nf(i-1)^2$ for $i>0$. The general solution of the recurrence $S_{i+1} = cS_i^2$ is $S_i = c^{2^i-1}S_0^{2^i}$. Setting $S_0 = 2^h$ and $c = 2n$, we have $f(i) \leqq (2n)^{2^i-1}2^{2^i h} \leqq 2^{h2^i+2^i-1+(2^i+1)\log n}$. Hence,

$$f(\lceil \log m \rceil) \leqq s^{2m(h+1)-1+(2m-1)\log n} \leqq 2^{2m(h+1+\log n)}. \qquad \square$$

The key idea of the Theorem 3.3 is that when $m > N^{1/8}$, the a_i are grouped into blocks of size $<m$ and the product circuit is applied to these smaller blocks, thus reducing m relative to N. When $m \leqq N^{1/8}$ our DFT reduction of Lemma 3.1 is applied to decrease N relative to m. In our original construction (Reif [83]) we required $O(\log \log N)$ applications of Lemma 3.1 to accomplish this decrease of N. Beame, Cook, and Hoover [84a] suggested an improvement which requires only a constant number of applications of our DFT reduction to appropriately reduce N. We give this improved version below, with their kind permission.

THEOREM 3.3. *For N a power of two, the product of m N-bit integers* mod 2^N+1 *has boolean depth* $O(\log(m)\log\log N + \log(N))$ *and size* $(mN)^{O(1)}$.

Proof. Given a list of N-bit integers $a_1, \cdots, a_m$ we compute the product $\prod_{i=1}^m a_i \bmod (2^N+1)$. Let the boolean depth and size required to compute this product be $D(m, N)$ and $S(m, N)$ respectively. Let $t(x)$ be the largest power of two less than x. Using this notation, Lemma 3.1 leads to the following recurrences

(i)
$$D(m, N) \leqq D(m, t((mN)^{3/5})) + O(\log mN),$$
$$S(m, N) \leqq (mN)^{3/5} S(m, t((mN)^{3/5})) + (mN)^{O(1)}.$$

(Note: slightly tighter recurrences can be obtained from Lemma 3.1, but this does not significantly affect the asymptotic analysis.)

Reduction of m: When $m > N^{1/8}$ group the m input integers into blocks of size at most $\lceil N^{1/8} \rceil$ and compute the products for each block. Then compute the product of all the $\lceil m/N^{1/8} \rceil$ blocks. To avoid worrying about the ceiling function in describing the number of integers in each of these products, first perform a single multiplication of two integers mod 2^N+1 to reduce this number by one. Thus,

(ii)
$$D(m, N) \leqq D(N^{1/8}, N) + D\left(\frac{m}{N^{1/8}}, N\right) + O(\log mN),$$
$$S(m, N) \leqq \left\lceil \frac{m}{N^{1/8}} \right\rceil S(N^{1/8}, N) + S\left(\frac{m}{N^{1/8}}, N\right) + (mN)^{O(1)}.$$

Continuing this process recursively results in an $N^{1/8}$-ary tree of multiplication nodes; so the desired product may be computed using sub-circuits which compute products of only $N^{1/8}$ integers. This tree has depth of at most $\lceil 8 \log m / \log N \rceil$ and certainly has fewer than m nodes. It follows that

(iii)
$$D(m, N) \leqq \left\lceil \frac{8 \log m}{\log N} \right\rceil D(N^{1/8}, N) + O(\log mN),$$
$$S(m, N) \leqq mS(N^{1/8}, N) + (mN)^{O(1)}.$$

It is now possible to consider the problem for $m \leqq N^{1/8}$. The solution for arbitrary m clearly follows from this solution via a single application of reduction (ii). The method of attack is to use reductions (i) and (ii) alternatively to reduce the problem to a smaller one of the same type.

Reduction of N: Apply the DFT reduction (i) twice and then reduction (ii). Thus

$$\text{(iv)} \qquad D(N^{1/8}, N) \leqq D(N^{1/8}, t(N^{1/2})) + O(\log N)$$
$$\leqq 2D((N^{1/2})^{1/8}, t(N^{1/2})) + O(\log N)$$

and

$$S(N^{1/8}, N) \leqq N^{5/4} S(N^{1/8}, t(N^{1/2})) + N^{O(1)}$$
$$\leqq N^{3/2} S((N^{1/2})^{1/8}, t(N^{1/2})) + N^{O(1)}.$$

So for sufficiently large N and some fixed c, d

$$\text{(v)} \qquad D(N^{1/8}, N) \leqq 2D((N^{1/2})^{1/8}, t(N^{1/2})) + c \log N,$$
$$S(N^{1/8}, N) \leqq N^{3/2} S((N^{1/2})^{1/8}, t(N^{1/2})) + N^d.$$

The original problem of size N has been reduced to problems of size $N^{1/2}$. These reductions must be applied $\leqq \log\log N$ times until the problems are of constant size. Analysing (v) carefully by expanding out terms, we get

$$\text{(vi)} \qquad D(N^{1/8}, N) \leqq c \log N + 2c \log N^{1/2} + 2^2 c \log N^{1/4} + \cdots$$
$$+ 2^{\log\log N} c \log N^{2^{-\log\log N}},$$
$$S(N^{1/8}, N) \leqq N^d + N^{3/2+d} + N^{3/2+3/2(1/2)+d} + \cdots$$
$$+ N^{3/2+3/2(1/2)+\cdots+3/2(2^{-\log\log N})+d},$$

where the last term in each expression is the cost of the depth $\log N$ in constant size problems. These last terms are bounded by $O(\log n)$ and N^{3+d}, respectively. Summing the $\log\log N$ terms in each expression of (vi) we get

$$\text{(vii)} \qquad D(N^{1/8}, N) \leqq (c+1) \log N \log\log N,$$
$$S(N^{1/8}, N) \leqq N^{3+d} \log\log N.$$

Substituting (vii) into (iii), we get

$$\text{(viii)} \qquad D(m, N) \leqq O(\log(m) \log\log N + \log mN),$$
$$S(m, N) = (mN)^{O(1)},$$

and the theorem is proven. □

3.3. Multiprecision evaluation of polynomials and power series. Let $p(x)$ be a real polynomial or a real power series with $n-1$ given rational coefficients of magnitude $<2^n$. We wish to evaluate $p(x)$ at a floating point real x_0 within accuracy $o(2^{-n})$. Theorem 3.3 implies

COROLLARY 3.1. *The evaluation of $p(x)$ at a given x_0 to accuracy $o(2^{-n})$ has boolean depth $O(\log n(\log\log n))$ and size $n^{O(1)}$.*

The elementary functions $\exp(x)$, $\log(x)$, $\sin(x)$, $\cos(x)$, $\arctan(x)$, square root (x), etc. have Taylor series expansions convergent within accuracy $o(2^{-n})$ over fixed intervals.

COROLLARY 3.2. *The evaluation of an elementary function over a fixed interval with a Taylor series expansion convergent to accuracy $o(2^{-n})$ has boolean depth $O(\log n(\log\log n))$ and size $n^{O(1)}$.*

COROLLARY 3.3. *The elementary symmetric functions (see § 2.4) over the reals have boolean depth $O(\log n(\log\log n))$ and size $n^{O(1)}$.*

3.4. Reciprocals and division of integers. Let a be an integer within bounds $2^{n-1} \leqq a < 2^n$. Then a has a binary representation $\sum_{i=0}^{n-1} a_i 2^i$ where $a_{n-1} = 1$. The reciprocal of a is $2^{-(n-1)}r$, where $r = \sum_{i=0}^{\infty} r_i 2^{-i}$. We wish to compute the first n bits $r_0, \cdots, r_{n-1}$. For this, we can use the product form of Anderson et al. [67] and Savage [76, p. 256].

LEMMA 3.3. *If*

$$\tilde{r} = \prod_{i=0}^{\lceil \log(n+1)-1 \rceil} (1+(1-2^{-n}a)^{2^i})$$

then $|r - \tilde{r}| = o(2^{-n})$.

By Theorem 3.3 and the above lemma,

COROLLARY 3.4. *The reciprocal can be computed within accuracy* $o(2^{-n})$ *by a boolean circuit of depth* $O(\log n(\log\log n))$ *and size* $n^{O(1)}$.

COROLLARY 3.5. *Given integers* a, b *with binary respresentation containing* n *bits, we can compute in boolean depth* $O(\log n(\log\log n))$ *the division quotient* q *and remainder* r *integers such that* $a = qb + r$ *and* $0 \leqq r < b$.

Further work and open problems. A subsequent paper of Beame, Cook, and Hoover [84b] gives $O(\log n)$ depth boolean circuits for taking the product of n integers and integer division. These circuits are nonuniform, in the sense of Borodin [77] since their construction requires more than logarithmic space.

It remains an open problem to find a uniform circuit of $O(\log n)$ depth for integer division.

Also, the circuit depth complexity of the following problems remain open: given integers a, b, p such that $0 < a, b < p < 2^n$,

(1) compute $a^b \bmod p$;

(2) compute the greatest common divisor of a and b;

(3) compute the multiplicative inverse of a, for a relatively prime to p.

The obvious circuits for these problems have $\Omega(n \log n)$ depth. If we use our improved techniques for integer products described in this paper, this depth bound is reduced by a factor of $O((\log\log n)/\log n)$.

NC circuits (see Cook [81]) are uniform boolean circuits of constant degree, $n^{O(1)}$ size and $(\log n)^{O(1)}$ depth. RNC circuits are NC circuits with, in addition, a source of truly random bits. We conjecture that no RNC circuits exist for the above problems (1)-(3). Reif and Tygar [84] show that this conjecture for problem (3) would have an interesting surprising consequence, namely an efficient method for parallel pseudo random number generation. In particular, they give for any $\varepsilon > 0$ and $c \geqq 1$ a NC circuit of depth $O(\log n \log\log n)$ for generating n^c pseudo random bits from only n^ε truly random bits. They show these pseudo random bits cannot be distinguished from truly random bits by any RNC circuit, assuming there is no RNC circuit for problem (3) for infinitely many n.

Acknowledgments. I am grateful to F. Bragdon who first taught me to divide and encouraged me to experiment with faster methods for long division.

The division algorithm of S. A. Cook's Ph.D. thesis stimulated this research L. Valiant gave some useful criticism of preliminary attempts to develop my integer division circuit.

I would like to thank the referees for their useful comments which significantly improved the presentation of this paper.

Also, we would like to thank P. W. Beame, S. A. Cook, and H. J. Hoover for permission to describe their improvement to our boolean circuit for integer product.

REFERENCES

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

S. F. ANDERSON, J. G. EARLE, R. E. GOLDSCHMIDT AND D. M. POWERS, *The IBM system* 360 *Model* 91: *floating point multiplication unit*, IBM J. Res. Dev., 11 (1967), pp. 34-53.

P. W. BEAME, S. A. COOK AND H. J. HOOVER, personal communication, February, 1984. [84a].

———, *Small depth circuits of integer products, powers, and division*, 25th Annual Symposium on Foundations of Computer Science, Singer Island, FL, 1984, pp. 1-11. [84b].

A. BORODIN, J. VON ZUR GATHEN AND J. HOPCROFT, *Fast parallel matrix and* GCD *computations*, 23rd Annual Symposium on Foundations of Computer Sciences, Chicago, IL, 1982, pp. 65-71.

A. BORODIN, *On relating time and space to size and depth*, this Journal, 6 (1977), pp. 733-744.

A. BORODIN AND I. MUNRO, *The Computation Complexity of Algebraic and Numeric Problems*, American Elsevier, New York, 1975, pp. 77-147.

R. P. BRENT, *Fast multiple-precision evaluation of elementary functions*, J. Assoc. Comput. Mach., 23 (1976), pp. 242-251.

S. A. COOK, Ph.D. thesis, Harvard Univ., Cambridge, MA, 1966.

———, *Towards a complexity theory of synchronous parallel computation*, Extrait de l'enseignement mathématique, T XXVII, fase 1-2, 1981.

J. W. COOLEY AND J. TUKEY, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297-301.

A. N. KRAPCHENKO, *Asymptotic estimation of addition time of a parallel adder*, Mat. Zametki, 9 (1967), pp. 35-40; Syst. Theory Res., 19 (1970), pp. 105-122.

D. E. KNUTH, *The Art of Computer Programming*: *Vol.* II, *Seminumerical Algorithms*, 2nd edition, Addison-Wesley, Reading, MA, 1981.

H. T. KUNG, *New algorithms and lower bounds for the parallel evaluation of certain rational expressions and recurrences*, J. Assoc. Comput. Mach., 23 (1976), pp. 252-261.

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

LAGRANGE, Mémoires Acad. Royale des Sciences et Belles-Lettres de Berlin, 24 (1768), pp. 251-326.

J. D. LIPSON, *Chinese remainder and interpolation algorithms*, Proc. 2nd Symposium on Symbolic and Algebraic Manipulation, Association for Computing Machinery, New York, 1971, pp. 372-391.

Y. OFMAN, *On the algorithmic complexity of discrete functions*, Dokl. Akad. Nauk SSSR, 195 (1962), pp. 48-51; Sov. Phys. Dokl., 7 (1963), pp. 589-591.

J. POLLARD, *The fast Fourier transform in a finite field*, Math. Comp., 25 (1963), pp. 365-374.

J. H. REIF, *Probabilistic parallel prefix computation*, TR-08-83, Aiken Computation Laboratory, Harvard Univ., Cambridge, MA, 1983.

———, *Logarithmic depth circuits for algebraic functions*, 24th Annual Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 138-145.

J. H. REIF AND J. D. TYGAR, *Efficient parallel pseudo-random number generation*, TR-07-84, Aiken Computation Laboratory, Harvard Univ., Cambridge, MA, 1984.

J. E. SAVAGE, *The Complexity of Computing*, John Wiley, New York, 1976, pp. 237-260.

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

C. S. WALLACE, *A suggestion for a fast multiplier*, IEEE Trans. Electronic Computing, EC-13 (1964), pp. 14-17.

S. WINOGRAD, *On the time to perform multiplication*, J. Assoc. Comput. Mach., 14 (1967), pp. 793-802.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986

© 1986 Society for Industrial and Applied Mathematics

ALGEBRAIC COMPUTATIONS OF SCALED PADÉ FRACTIONS*

STANLEY CABAY† AND DONG-KOO CHOI‡

Abstract. Two companion algorithms are developed for constructing Padé fractions along an off-diagonal path of the Padé table for a function $-A(z)/B(z)$, where $A(z)$ and $B(z)$ are formal power series over a field.

One of the algorithms computes the first n Padé fractions along the off-diagonal in time $O(n^2)$. When $A(z)$ and $B(z)$ are finite power series (i.e., polynomials), it is shown that the algorithm is equivalent to Euclid's extended algorithm for computing greatest common divisors.

The other algorithm, a generalization of the first, proceeds along the off-diagonal in quadratic steps, and is of complexity $O(n \log^2 n)$. When $A(z)$ and $B(z)$ are polynomials, the second algorithm becomes a fast Euclid's extended algorithm for computing greatest common divisors. The algorithm is of the same complexity as other fast greatest common divisor methods, but its iterative nature provides a practical advantage during implementation.

The algorithms may also be used for computing Padé fractions along an anti-diagonal path of the Padé table. The fast algorithm is of the same complexity as other fast algorithms for anti-diagonal computations. However, it has the advantage of being able to determine easily any specific Padé fraction along the anti-diagonal.

Key words. Padé approximants, power series, algebraic manipulation, greatest common divisor, Euclidean algorithm

1. Introduction.

The Padé table of a formal power series

$$A(z) = \sum_{i=0}^{\infty} a_i z^i \tag{1.1}$$

is a doubly infinite array of rational functions

$$\frac{U_{mn}(z)}{V_{mn}(z)} = \frac{\sum_{i=0}^{m} u_i z^i}{\sum_{i=0}^{n} v_i z^i} \tag{1.2}$$

determined in such a manner that the Maclaurin expansion of $U_{mn}(z)/V_{mn}(z)$ agrees with $A(z)$ as far as possible. The power series $A(z)$ is said to be normal if, for each pair (m, n), this agreement is exact through the power z^{m+n}. The foundation for the development of Padé theory was laid by Cauchy (1821) in his famous "Cour d'Analyse". Later, Frobenius (1881) developed the basic algorithmic aspects of the theory, and Padé [15] treated in detail certain abnormal cases.

Since Padé's time, Padé tables have become a classical tool of analysis. Their analytical properties have been studied in great depth and are surveyed, for example, by Gragg [11] and by Baker [2]. Traditionally, it is assumed that the coefficients in (1.1) and (1.2) lie in the field of complex numbers, and that the power series and the rational functions are to be evaluated at certain points in the complex plane.

Although the results obtained in this paper are likely to have an impact in an analytical (or numerical) setting, the effects of this impact are not examined. The issues addressed are strictly algebraic ones; that is, no consideration is given to the goodness of the approximation of (1.2) to (1.1). Instead, the objective is to provide an effective tool to algebraically manipulate rational functions as truncated power series (for which the cost of operations is relatively cheap), and to transform back to rational form on request. It is assumed that the coefficients lie in an arbitrary field.

* Received by the editors March 1, 1984, and in final form November 28, 1984.

† Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada, T6G 2H1.

‡ Department of Computer Science, University of North Dakota, Grand Forks, North Dakota 58201.

Various relationships are known to exist between neighboring elements in the Padé table. These relationships have been used to derive numerous $O(n^2)$ methods for computing a sequence of elements in the Padé table. A survey and comparison of these methods are given in Brezinski [5], Claessens [8] and Wynn [19]. All these methods have a major flaw; they may fail in the abnormal case. In § 3, a new relationship between elements lying along an off-diagonal path in the Padé table is derived. This leads to yet another $O(n^2)$ method; however, the new method succeeds in the abnormal case. Furthermore, if the coefficient field has an appropriate nth root of unity (which permits fast multiplication and division of polynomials), the asymptotic complexity of the algorithm becomes $O(n \log^2 n)$.

The new algorithm can be applied to the quotient of two power series. Then, in particular, it can be applied to the quotient of two finite power series (i.e. the quotient of two polynomials). In § 4, it is shown that if all elements along a specific off-diagonal of the Padé table are computed, then the new algorithm is equivalent to Euclid's extended algorithm for computing greatest common divisors. Furthermore, if fast polynomial operations can be performed, the new algorithm can compute the greatest common divisor of two polynomials in $O(n \log^2 n)$ arithmetic operations. The algorithm has three advantages over the other fast methods (Moenck [14], Aho et al. [1], and Brent et al. [4]) for computing greatest common divisors. It is basically an iterative algorithm rather than a recursive one, and consequently, significant cost savings can result during implementation. Secondly, it produces intermediate polynomial remainder sequences as a by-product, which is a valuable feature for some applications. Finally, various details about the nature of its behavior are easier to comprehend.

The algorithm can be applied to the quotient of the reciprocals of two truncated power series (polynomials). It is shown in § 4.3 that this yields successive elements along an anti-diagonal path of the Padé table.

2. Preliminary discussion. The class **P** of formal power series over a field **F** consists of expressions of the form

$$A(z) = \sum_{i=0}^{\infty} a_i z^i$$

with coefficients $a_i \in \mathbf{F}$. We denote the units of **P** by

$$\mathbf{U} = \left\{ A(z) = \sum_{i=0}^{\infty} a_i z^i \,\middle|\, a_0 \neq 0,\ A(z) \in \mathbf{P} \right\}.$$

Associated with each such unit is a set of rational functions defined as follows:

DEFINITION. Let $A(z) \in \mathbf{U}$, and let m and n be nonnegative integers. The rational form

$$\frac{U_{mn}(z)}{V_{mn}(z)} = \frac{u_0 + u_1 z + \cdots + u_m z^m}{v_0 + v_1 z + \cdots + v_n z^n} \tag{2.1}$$

is called a *Padé form* of type (m, n) for $A(z)$ if

$$\text{(a)} \quad V_{mn}(z) \not\equiv 0, \quad \text{and} \tag{2.2}$$

$$\text{(b)} \quad A(z) \cdot V_{mn}(z) - U_{mn}(z) = O(z^{m+n+1}). \tag{2.3}$$

The (algebraic) O-symbol indicates that the right side is a power series beginning with the power $z^{m+n+k+1}$, $0 \leqq k \leqq \infty$; $k = +\infty$ means that $A(z) \cdot V_{mn}(z) - U_{mn}(z) = 0$.

THEOREM 2.1 (Frobenius). *Padé forms of type* (m, n) *for* $A(z) \in \mathbf{U}$ *always exist.*

Proof. See Gragg [11], for example. □

However, Padé forms are not unique (see Gragg [11], for example). To overcome this drawback, we introduce

DEFINITION. The rational function $\gamma_{mn}(z) = S_{mn}(z)/T_{mn}(z)$, where $T_{mn}(z) \neq 0$ is the scaled Padé fraction of type (m, n) for $A(z)$ if

$$\text{(I)} \quad \min\{m - \partial(S_{mn}), n - \partial(T_{mn})\} = 0,^1 \tag{2.4}$$

$$\text{(II)} \quad \mathrm{GCD}(S_{mn}, T_{mn}) = z^{\lambda_{mn}}, \quad \text{for some integer } \lambda_{mn} \geqq 0, \text{ and} \tag{2.5}$$

$$\text{(III)} \quad A(z) \cdot T_{mn}(z) - S_{mn}(z) = O(z^{m+n+1}). \tag{2.6}$$

THEOREM 2.2. *Scaled Padé fractions exist, and are unique up to a multiplicative constant.*

Proof. From Theorem 2.1, there exist $U_{mn}(z)$ and $V_{mn}(z)$ such that

$$\partial(U_{mn}) \leqq m, \partial(V_{mn}) \leqq n, \quad \text{and}$$
$$A(z)V_{mn}(z) - U_{mn}(z) = O(z^{m+n+1}).$$

Let $D(z) = \mathrm{GCD}(U_{mn}, V_{mn})$, and

$$P_{mn}(z) = U_{mn}(z)/D(z), \tag{2.7}$$

$$Q_{mn}(z) = V_{mn}(z)/D(z). \tag{2.8}$$

Define

$$S_{mn}(z) = z^{\lambda_{mn}} \cdot P_{mn}(z), \tag{2.9}$$

$$T_{mn}(z) = z^{\lambda_{mn}} \cdot Q_{mn}(z), \tag{2.10}$$

where

$$\lambda_{mn} = \min\{m - \partial(P_{mn}), n - \partial(Q_{mn})\}.$$

Then

$$\mathrm{GCD}(S_{mn}, T_{mn}) = z^{\lambda_{mn}},$$

and

$$\partial(S_{mn}) = \lambda_{mn} + \partial(P_{mn}) \leqq m - \partial(P_{mn}) + \partial(P_{mn}) \leqq m,$$

and similarly

$$\partial(T_{mn}) \leqq n.$$

Moreover, because $\lambda_{mn} = \min\{m - \partial(P_{mn}), n - \partial(Q_{mn})\}$, either $\partial(S_{mn}) = m$, or $\partial(T_{mn}) = n$, or both. Finally, property III of scaled Padé fractions is satisfied since

$$A(z)T_{mn}(z) - S_{mn}(z) = z^{\lambda_{mn}}\{A(z)V_{mn}(z) - U_{mn}(z)\}/D(z) = O(z^{m+n+1}),$$

where $\partial(D) \leqq \lambda_{mn}$.

To show uniqueness, let $\bar{S}_{mn}(z)/\bar{T}_{mn}(z)$ be another scaled Padé fraction of type (m, n) for $A(z)$. Then

$$T_{mn}(z)(A(z)\bar{T}_{mn}(z) - \bar{S}_{mn}(z)) = O(z^{m+n+1}),$$
$$\bar{T}_{mn}(z)(A(z)T_{mn}(z) - S_{mn}(z)) = O(z^{m+n+1}).$$

[1] $\partial(p)$ denotes the degree of the polynomial $p(z)$.

Thus,

$$S_{mn}(z)\bar{T}_{mn}(z) - \bar{S}_{mn}(z)T_{mn}(z) = O(z^{m+n+1}).$$

Since $\partial(S_{mn}) + \partial(\bar{T}_{mn}) \leqq m+n$, and $\partial(\bar{S}_{mn}) + \partial(T_{mn}) \leqq m+n$, it follows that

$$S_{mn}(z)\bar{T}_{mn}(z) - \bar{S}_{mn}(z)T_{mn}(z) = 0;$$

that is,

$$\frac{S_{mn}(z)}{T_{mn}(z)} = \frac{\bar{S}_{mn}(z)}{\bar{T}_{mn}(z)}.$$

Therefore, from conditions I and II, $S_{mn}(z) = \bar{S}_{mn}(z)$ and $T_{mn}(z) = \bar{T}_{mn}(z)$. □

Traditionally, a Padé fraction of type (m, n) for $A(z)$ (see Gragg [11]) is obtained by determining the cofactors of any given Padé form of type (m, n). In terms of a scaled Padé fraction, the Padé fraction $P_{mn}(z)/Q_{mn}(z)$ of type (m, n) for $A(z)$ so computed is defined as

$$P_{mn}(z) = z^{-\lambda_{mn}} S_{mn}(z), \qquad Q_{mn}(z) = z^{-\lambda_{mn}} T_{mn}(z).$$

Thus, Padé fractions also exist, and are unique up to multiplicative constant; however, they may no longer satisfy the order condition (2.3) or (2.6). For purposes of subsequent algorithm development and analysis, scaled Padé fractions have a tremendous advantage over the other definitions.

3. Computation of off-diagonal scaled Padé fractions.

3.1. Preliminary results. The scaled Padé fractions can be arranged in a doubly infinite array as follows:

DEFINITION. The collection of all scaled Padé fractions of type (m, n) for $A(z) \in \mathbf{U}$, given by

$$\Gamma(A) = \begin{vmatrix} \gamma_{-1,-1} & \gamma_{-1,0} & \gamma_{-1,1} & \cdots & \gamma_{-1,n} & \cdots \\ \gamma_{0,-1} & \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,n} & \cdots \\ \gamma_{1,-1} & \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,n} & \cdots \\ \vdots & \vdots & \vdots & & & \\ \gamma_{m,-1} & \gamma_{m,0} & \gamma_{m,1} & \cdots & \gamma_{m,n} & \cdots \\ \vdots & \vdots & \vdots & & \vdots & \end{vmatrix} \tag{3.1}$$

is called the (extended) *scaled Padé table*[2] for $A(z)$. The modifier extended denotes the inclusion of the first row and column in the table which are defined as follows:

$$S_{m,-1}(z) = -z^m \quad \text{and } T_{m,-1}(z) = 0 \quad \text{for } m \geqq -1, \quad \text{and}$$

$$S_{-1,n}(z) = 0 \quad \text{and } T_{-1,n}(z) = -z^n \quad \text{for } n \geqq 0.$$

For computational purposes, define the N-truncated, scaled Padé table for $A(z)$ to be

$$\Gamma_N(A) = \begin{vmatrix} \gamma_{-1,-1} & \gamma_{-1,0} & \gamma_{-1,1} & \cdots & \gamma_{-1,N} \\ \gamma_{0,-1} & \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,N} \\ \gamma_{1,-1} & \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,N} \\ \vdots & \vdots & \vdots & & \vdots \end{vmatrix} \tag{3.2}$$

[2] Note that the (extended) scaled Padé table corresponds to the usual definition of the Padé table given, for example, in Gragg [11] with the exception that the scaling factor z^{λ} in (2.5) does not appear.

Let m and n be nonnegative integers such that $n > N$. The next few results are concerned with the construction of the scaled Padé fraction $\gamma_{mn}(z)$, given that $\Gamma_N(A)$ already exists. Without loss of generality, assume that $m \geqq n$ (otherwise, the same arguments can be applied to $1/A(z) \in \mathbf{U}$). Let

$$M = N + (m - n). \tag{3.3}$$

Then, (m, n) and (M, N) both lie along the $(m-n)$th off-diagonal path of the scaled Padé table $\Gamma(A)$ (see Fig. 3.1).

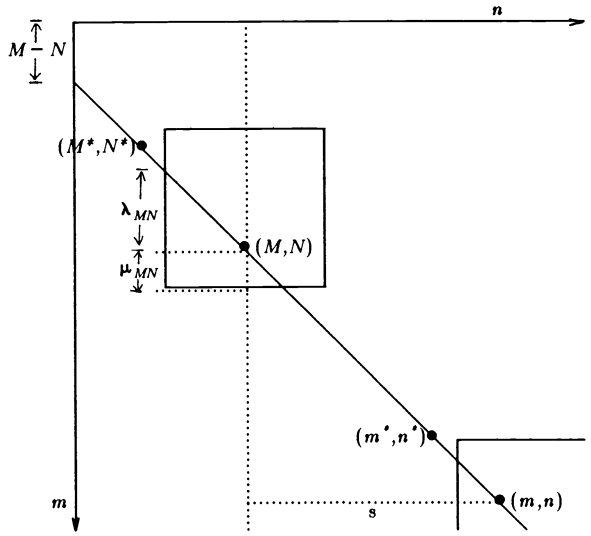


FIG. 3.1.

For the scaled Padé fraction of type (M, N) for $A(z)$, (2.6) becomes

$$A(z) \cdot T_{MN}(z) - S_{MN}(z) = z^{M+N+\mu_{MN}+1} R_1(z), \tag{3.4}$$

where $\mu_{MN} \geqq 0$, and $R_1(0) \neq 0$ if $\mu_{MN} < \infty$. Let

$$z^{\lambda_{MN}} = \mathrm{GCD}\,(S_{MN}, T_{MN}). \tag{3.5}$$

To construct $\gamma_{mn}(z)$, two separate cases, $\mu_{MN} \geqq (n - N)$ and $\mu_{MN} < (n - N)$, arise. These two cases are considered separately in Theorem 3.1 and Theorem 3.4.

THEOREM 3.1. *If* $\mu_{MN} \geqq (n - N)$ *in* (3.4), *then the scaled Padé fraction* $\gamma_{mn}(z) = S_{mn}(z)/T_{mn}(z)$ *is given by*

$$S_{mn}(z) = z^{n-N} S_{MN}(z), \tag{3.6}$$

$$T_{mn}(z) = z^{n-N} T_{MN}(z). \tag{3.7}$$

Proof. Clearly from relation (3.3),

$$\partial(S_{mn}) = n - N + \partial(S_{MN}) \leqq n + M - N = m.$$

Similarly, $\partial(T_{mn}) \leqq n$. Furthermore, it is clear that $\min\{m - \partial(S_{mn}), n - \partial(T_{mn})\} = 0$ since

$\min\{M-\partial(S_{MN}), N-\partial(T_{MN})\}=0$. Using the fact that $(n-N)\leqq\mu_{MN}$,

$$\begin{aligned} A(z)T_{mn}(z)-S_{mn}(z) &= z^{n-N}(A(z)T_{MN}(z)-S_{MN}(z)) \\ &= z^{n-N}(z^{M+N+\mu_{MN}+1}R_1(z)) \\ &= z^{m+N+\mu_{MN}+1}R_1(z) \\ &= O(z^{m+n+1}). \end{aligned}$$

Finally, GCD $(S_{mn}, T_{mn}) = z^{n-N}$ GCD $(S_{MN}, T_{MN}) = z^{n-N+\lambda_{MN}}$.

Consequently, $\gamma_{mn}(z)=S_{mn}(z)/T_{mn}(z)$ given by (3.6) and (3.7) is the scaled Padé fraction of type (m, n) for $A(z)$. □

For the case $\mu_{MN}<(n-N)$, let

$$M^*=M-\lambda_{MN}-1, \tag{3.8}$$

$$N^*=N-\lambda_{MN}-1 \tag{3.9}$$

(see Fig. 3.1). Clearly, the scaled fraction $\gamma_{M^*N^*}(z)$ for $A(z)$ satisfies

$$A(z)\cdot T_{M^*N^*}(z)-S_{M^*N^*}(z)=z^{M^*+N^*+1}R_0(z), \tag{3.10}$$

where $R_0(0)\neq 0$. In addition, by Theorem 2.2 and by the definition of M^* and N^*, it follows that

$$S_{MN}(z)/T_{MN}(z)\neq S_{M^*N^*}(z)/T_{M^*N^*}(z). \tag{3.11}$$

From the two unit power series, $R_1(z)$ and $R_0(z)$ given by (3.4) and (3.10), respectively, construct the unique power series

$$\bar{A}(z)=R_0(z)/R_1(z). \tag{3.12}$$

Associated with $\bar{A}(z)\in\mathbf{U}$, let

$$\bar{m}=n-N+\lambda_{MN}, \tag{3.13}$$

$$\bar{n}=n-N-\mu_{MN}-1. \tag{3.14}$$

Now assume that the $\bar{n}$-truncated scaled Padé table, $\Gamma_{\bar{n}}(\bar{A})$, for $\bar{A}(z)$ has been constructed as well. That is, assume that the $(\bar{m}, \bar{n})$ scaled Padé fraction, $\bar{\gamma}_{\bar{m}\bar{n}}(z)=\bar{S}_{\bar{m}\bar{n}}(z)/\bar{T}_{\bar{m}\bar{n}}(z)$, such that

$$\bar{A}(z)\cdot\bar{T}_{\bar{m}\bar{n}}(z)-\bar{S}_{\bar{m}\bar{n}}(z)=O(z^{\bar{m}+\bar{n}+1}) \tag{3.15}$$

is available.

The scaled Padé fractions $\gamma_{MN}(z)$, $\gamma_{M^*N^*}(z)$ and $\bar{\gamma}_{\bar{m}\bar{n}}(z)$ provide sufficient information to obtain directly a Padé form of type (m, n) for $A(z)$. This Padé form is constructed in Lemma 3.2 below. It is shown, later in Theorem 3.4, that this form is also the scaled Padé fraction of type (m, n) for $A(z)$.

LEMMA 3.2. *Let* $\mu_{MN}<(n-N)$ *in* (3.4), *and let*

$$S_{mn}(z)=z^{-\lambda_{MN}}S_{MN}(z)\bar{S}_{\bar{m}\bar{n}}(z)-z^{\alpha}S_{M^*N^*}(z)\bar{T}_{\bar{m}\bar{n}}(z), \tag{3.16}$$

$$T_{mn}(z)=z^{-\lambda_{MN}}T_{MN}(z)\bar{S}_{\bar{m}\bar{n}}(z)-z^{\alpha}T_{M^*N^*}(z)\bar{T}_{\bar{m}\bar{n}}(z), \tag{3.17}$$

where $\alpha=\lambda_{MN}+\mu_{MN}+2$. *Then* $S_{mn}(z)/T_{mn}(z)$ *is a Padé form of type* (m, n) *for* $A(z)$.

Proof.

$$\begin{aligned}\partial(S_{mn}) &= \max\{\partial(z^{-\lambda_{MN}}S_{MN}\bar{S}_{\bar{m}\bar{n}}), \partial(z^{\alpha}S_{M^*N^*}\bar{T}_{\bar{m}\bar{n}})\}\\ &\leqq \max\{-\lambda_{MN}+M+\bar{m}, \alpha+M^*+\bar{n}\}\\ &= \max\{-\lambda_{MN}+M+(n-N+\lambda_{MN}),\\ &\qquad (\lambda_{MN}+\mu_{MN}+2)+(M-\lambda_{MN}-1)+(n-N-\mu_{MN}-1)\}\\ &= \max\{m, m\}\\ &= m.\end{aligned}$$

Similarly, $\partial(T_{mn}) \leqq n$. Moreover, the fact that $S_{MN}(z)/T_{MN}(z)$, $S_{M^*N^*}(z)/T_{M^*N^*}(z)$ and $\bar{S}_{\bar{m}\bar{n}}(z)/\bar{T}_{\bar{m}\bar{n}}(z)$ are the scaled Padé fractions yields immediately that

$$\min\{m-\partial(S_{mn}), n-\partial(T_{mn})\} = 0.$$

Furthermore, from (3.4), (3.10), (3.12) and (3.15), it follows that

$$\begin{aligned}A(z)\cdot T_{mn}(z)-S_{mn}(z) &= A(z)\{z^{-\lambda_{MN}}T_{MN}(z)\bar{S}_{\bar{m}\bar{n}}(z)-z^{\alpha}T_{M^*N^*}(z)\bar{T}_{\bar{m}\bar{n}}(z)\}\\ &\quad -\{z^{-\lambda_{MN}}S_{MN}(z)\bar{S}_{\bar{m}\bar{n}}(z)-z^{\alpha}S_{M^*N^*}(z)\bar{T}_{\bar{m}\bar{n}}(z)\}\\ &= z^{-\lambda_{MN}}\bar{S}_{\bar{m}\bar{n}}(z)\{A(z)T_{MN}(z)-S_{MN}(z)\}\\ &\quad -z^{\alpha}\bar{T}_{\bar{m}\bar{n}}(z)\{A(z)T_{M^*N^*}(z)-S_{M^*N^*}(z)\}\\ &= z^{-\lambda_{MN}}\bar{S}_{\bar{m}\bar{n}}(z)\{R_1(z)z^{M+N+\mu_{MN}+1}\}-z^{\alpha}\bar{T}_{\bar{m}\bar{n}}(z)\{R_0(z)z^{M^*+N^*+1}\}\\ &= z^{M+N+\mu_{MN}-\lambda_{MN}+1}\{R_1(z)\bar{S}_{\bar{m}\bar{n}}(z)-R_0(z)\bar{T}_{\bar{m}\bar{n}}(z)\}\\ &= -R_1(z)z^{M+N+\mu_{MN}-\lambda_{MN}+1}\{\bar{T}_{\bar{m}\bar{n}}(z)R_0(z)/R_1(z)-\bar{S}_{\bar{m}\bar{n}}(z)\}\\ &= -R_1(z)z^{M+N+\mu_{MN}-\lambda_{MN}+1}O(z^{\bar{m}+\bar{n}+1})\\ &= O(z^{M+N+\mu_{MN}-\lambda_{MN}+1+\bar{m}+\bar{n}+1})\\ &= O(z^{m+n+1}).\end{aligned}$$

□

In order that S_{mn}/T_{mn} in (3.16) and (3.17) be a scaled Padé fraction of type (m, n), it remains to show that GCD $(S_{mn}, T_{mn}) = z^{\lambda_{mn}}$ for some λ_{mn}. With this intent, consider again the $(\bar{m}, \bar{n})$th entry of $\Gamma_{\bar{n}}(\bar{A})$, and let

$$\bar{m}^* = \bar{m} - \bar{\lambda}_{\bar{m}\bar{n}} - 1, \tag{3.18}$$

$$\bar{n}^* = \bar{m} - \bar{\lambda}_{\bar{m}\bar{n}} - 1, \tag{3.19}$$

where

$$z^{\bar{\lambda}_{\bar{m}\bar{n}}} = \text{GCD}\,(\bar{S}_{\bar{m}\bar{n}}, \bar{T}_{\bar{m}\bar{n}}). \tag{3.20}$$

The scaled Padé fraction $\bar{\gamma}_{\bar{m}^*,\bar{n}^*}(z)$ for $\bar{A}(z)$ satisfies

$$\bar{A}\bar{T}_{\bar{m}^*\bar{n}^*}-\bar{S}_{\bar{m}^*\bar{n}^*} = O_E(z^{\bar{m}^*+\bar{n}^*+1}) = O_E(z^{\bar{m}+\bar{n}-2\bar{\lambda}_{\bar{m}\bar{n}}-1}),^{3} \tag{3.21}$$

and, in addition,

$$\frac{\bar{S}_{\bar{m}\bar{n}}(z)}{\bar{T}_{\bar{m}\bar{n}}(z)} \neq \frac{\bar{S}_{\bar{m}^*\bar{n}^*}(z)}{\bar{T}_{\bar{m}^*\bar{n}^*}(z)}. \tag{3.22}$$

[3] $R(z) = O_E(z^k)$ means that $R(z)$ is a power series whose first nonzero coefficient is the coefficient of z^k, exactly.

LEMMA 3.3. *Let*

$$S_{m^*n^*}(z) = z^{-\lambda_{MN}} S_{MN}(z)\bar{S}_{\bar{m}^*\bar{n}^*}(z) - z^{\alpha} S_{M^*N^*}(z)\bar{T}_{\bar{m}^*\bar{n}^*}(z), \tag{3.23}$$

$$T_{m^*n^*}(z) = z^{-\lambda_{MN}} T_{MN}(z)\bar{S}_{\bar{m}^*\bar{n}^*}(z) - z^{\alpha} T_{M^*N^*}(z)\bar{T}_{\bar{m}^*\bar{n}^*}(z) \tag{3.24}$$

where $\alpha = \lambda_{MN} + \mu_{MN} + 2$. *Then* $\gamma_{m^*n^*}(z) = S_{m^*n^*}(z)/T_{m^*n^*}(z)$ *is a Padé form of type* (m^*, n^*) *for* $A(z)$, *where*

$$m^* = m - \bar{\lambda}_{\bar{m}\bar{n}} - 1, \tag{3.25}$$

$$n^* = n - \bar{\lambda}_{\bar{m}\bar{n}} - 1. \tag{3.26}$$

Proof. The proof is identical to the proof of Lemma 3.2. □

THEOREM 3.4. *Let* $\mu_{MN} < (n - N)$ *in* (3.4), *and let*

$$\gamma_{mn}(z) = S_{mn}(z)/T_{mn}(z)$$

be defined by (3.16) *and* (3.17). *Then* $\gamma_{mn}(z)$ *is a scaled Padé fraction of type* (m, n) *for* $A(z)$.

Proof. Let

$$G_{mn}(z) = \mathrm{GCD}\,(S_{mn}, T_{mn}).$$

We first show that $\partial(G_{mn}) \leqq \bar{\lambda}_{\bar{m}\bar{n}}$, where $\bar{\lambda}_{\bar{m}\bar{n}}$ is given by (3.20). Suppose that $\partial(G_{mn}) > \bar{\lambda}_{\bar{m}\bar{n}}$, and proceed by contradiction. Let

$$U_{m^*n^*}(z) = z^{\partial(G_{mn}) - \bar{\lambda}_{\bar{m}\bar{n}} - 1} S_{mn}(z)/G_{mn}(z),$$

$$V_{m^*n^*}(z) = z^{\partial(G_{mn}) - \bar{\lambda}_{\bar{m}\bar{n}} - 1} T_{mn}(z)/G_{mn}(z),$$

where m^* and n^* are given by (3.25) and (3.26). Then $\partial(U_{m^*n^*}) \leqq m^*$, $\partial(V_{m^*n^*}) \leqq n^*$, and

$$\begin{aligned} A(z)\cdot V_{m^*n^*}(z) - U_{m^*n^*}(z) &= z^{\partial(G_{mn}) - \bar{\lambda}_{\bar{m}\bar{n}} - 1}\{A(z)T_{mn}(z) - S_{mn}(z)\}/G_{mn}(z) \\ &= O(z^{(-\bar{\lambda}_{\bar{m}\bar{n}} - 1) + (m+n+1)}) \\ &= O(z^{m^*+n^*+\bar{\lambda}_{\bar{m}\bar{n}}+1}). \end{aligned}$$

Thus, $U_{m^*+n^*}(z)/V_{m^*n^*}(z)$ is a Padé form of type (m^*, n^*) for $A(z)$.

But $S_{m^*n^*}(z)/T_{m^*n^*}(z)$, given by Lemma 3.3, is also a Padé form of type (m^*, n^*). Then,

$$S_{m^*n^*}(z)/T_{m^*n^*}(z) = U_{m^*n^*}(z)/V_{m^*n^*}(z) = S_{mn}(z)/T_{mn}(z),$$

or, equivalently,

$$S_{m^*n^*}(z)T_{mn}(z) - S_{mn}(z)T_{m^*n^*}(z) = 0. \tag{3.27}$$

Replacing $S_{mn}(z)/T_{mn}(z)$ and $S_{m^*n^*}/T_{m^*n^*}$ in (3.27) by the expanded forms (3.16), (3.17), (3.23) and (3.24), it follows that

$$\begin{aligned} &z^{\alpha} z^{-\lambda_{MN}}\{S_{MN}(z)T_{M^*N^*}(z) - S_{M^*N^*}(z)T_{MN}(z)\} \\ &\quad\cdot\{\bar{S}_{\bar{m}\bar{n}}(z)\bar{T}_{\bar{m}^*\bar{n}^*}(z) - \bar{S}_{\bar{m}^*\bar{n}^*}(z)\bar{T}_{\bar{m}^*\bar{n}^*}(z)\} = 0. \end{aligned}$$

Thus, either $S_{MN}(z)/T_{MN}(z) = S_{M^*N^*}(z)/T_{M^*N^*}(z)$, or $\bar{S}_{\bar{m}\bar{n}}(z)/\bar{T}_{\bar{m}\bar{n}}(z) = \bar{S}_{\bar{m}^*\bar{n}^*}(z)/\bar{T}_{\bar{m}\bar{n}}(z)$, which contradicts (3.11) and (3.22).

Thus, $\partial(G_{mn}) \leqq \bar{\lambda}_{\bar{m}\bar{n}}$. But,

$$z^{\bar{\lambda}_{\bar{m}\bar{n}}} = \mathrm{GCD}\,(\bar{S}_{\bar{m}\bar{n}}, \bar{T}_{\bar{m}\bar{n}}),$$

which implies that $z^{\bar{\lambda}_{\bar{m}\bar{n}}}$ divides both $S_{mn}(z)$ and $T_{mn}(z)$ in (3.16) and (3.17). That is,

$z^{\bar{\lambda}_{\bar{m}\bar{n}}}$ divides $G_{mn}(z)$, and consequently

$$G_{mn}(z) = z^{\bar{\lambda}_{\bar{m}\bar{n}}}.$$

Thus, $S_{mn}(z)/T_{mn}(z)$ given in Lemma 3.2 is not only a Padé form of type (m, n) for $A(z)$, but also the scaled Padé fraction of type (m, n) for $A(z)$, where

$$\text{GCD}\,(S_{mn}, T_{mn}) = z^{\lambda_{mn}}$$

and $\lambda_{mn} = \bar{\lambda}_{\bar{m}\bar{n}}$. □

THEOREM 3.5. *Let $\mu_{MN} < (n - N)$ in* (3.4) *and let $\gamma_{mn}(z) = S_{mn}(z)/T_{mn}(z)$ be the scaled Padé fraction of type (m, n) for $A(z)$. If*

$$m^* = m - \lambda_{mn} - 1 \tag{3.28}$$

and

$$n^* = n - \lambda_{mn} - 1, \tag{3.29}$$

where

$$z^{\lambda_{mn}} = \text{GCD}\,(S_{mn}, T_{mn}), \tag{3.30}$$

then the scaled Padé fraction of type (m^*, n^*) for $A(z)$ is $\gamma_{m^*n^*}(z) = S_{m^*n^*}(z)/T_{m^*n^*}(z)$, where $S_{m^*n^*}(z)$ and $T_{m^*n^*}(z)$ are given by (3.23) *and* (3.24).

Proof. The theorem follows using arguments identical to those of proof of Theorem 3.4, and using the results of Lemma 3.3. □

A simple example for the off-diagonal computation is presented.

Example. Let $A(z) = 1 + z^4 + z^5 + z^9 + z^{10} + 2z^{15} + \cdots$. This example constructs the scaled Padé fraction $\gamma_{7,6}(z) = S_{7,6}(z)/T_{7,6}(z)$ of type (7, 6) for $A(z)$. Since $m - n = 1$, the construction proceeds along the 1st off-diagonal path of the scaled Padé table $\Gamma(A)$.

Assume that $\Gamma_3(A)$ is already available, from which it can be determined that the scaled Padé fraction $\gamma_{4,3}(z)$ of type $(M, N) = (4, 3)$ is given by

$$\gamma_{4,3}(z) = \frac{S_{4,3}(z)}{T_{4,3}(z)} = \frac{1 - z + z^2 - z^3 + z^4}{1 - z + z^2 - z^3}. \tag{3.31}$$

From (3.4), the residual for $S_{4,3}(z)/T_{4,3}(z)$ is given by

$$A(z)T_{4,3}(z) - S_{4,3}(z) = R_1(z)z^{4+3+1}, \tag{3.32}$$

where $R_1(z) = -1 + z - z^5 + 2z^7 + \cdots$.

Consequently, (3.32) yields that $\mu_{4,3} = 0$. Since $\mu_{4,3} < n - N$, Theorem 3.4 is applicable. Observe that

$$z^{\lambda_{MN}} = z^0 = \text{GCD}\,(S_{4,3}, T_{4,3}),$$

and consequently the predecessor of $\gamma_{4,3}(z)$ along the first off-diagonal path is $\gamma_{M^*N^*}(z) = \gamma_{3,2}(z)$. Therefore, $\gamma_{3,2}(z)$ is contained in $\Gamma_3(A)$, and is found to be

$$\gamma_{3,2}(z) = \frac{S_{3,2}(z)}{T_{3,2}(z)} = \frac{z^2}{z^2}.$$

The residual for $S_{3,2}(z)/T_{3,2}(z)$ is given from

$$A(z)T_{3,2}(z) - S_{3,2}(z) = R_0(z)z^{3+2+1}, \tag{3.33}$$

where $R_0(z) = 1 + z + z^5 + z^6 + \cdots$.

The two residuals $R_1(z)$ and $R_0(z)$ give the residual power series $\bar{A}(z) \in \mathbf{U}$, where

$$\bar{A}(z) = R_0(z)/R_1(z) = -1 - 2z - 2z^2 - 2z^3 - \cdots. \tag{3.34}$$

Using (3.13) and (3.14),

$$\bar{m} = n - N + \lambda_{MN} = 3,$$

$$\bar{n} = n - N - \mu_{MN} - 1 = 2,$$

and in order to apply Theorem 3.4, it is therefore required to obtain the scaled Padé fraction $\bar{\gamma}_{3,2}(z)$ of type $(3, 2)$ and its predecessor for $\bar{A}(z)$. Assuming that $\mathbf{\Gamma}_3(\bar{A})$ is available, from it can be determined that

$$\bar{\gamma}_{3,2}(z) = \frac{\bar{S}_{3,2}(z)}{\bar{T}_{3,2}(z)} = \frac{-z - z^2}{z - z^2}. \tag{3.35}$$

Similarly, the predecessor of $\bar{\gamma}_{3,2}(z)$ along the first off-diagonal path of $\mathbf{\Gamma}_3(\bar{A})$ is given by $\bar{\gamma}_{1,0}(z) = \bar{S}_{1,0}(z)/\bar{T}_{1,0}(z) = (-1-2z)/1$.

Now by applying the formulae (3.16) and (3.17), the $(7, 6)$ entry of $\mathbf{\Gamma}(A)$ is computed by

$$S_{7,6}(z) = (1 - z + z^2 - z^3 + z^4) \cdot (-z - z^2) - z^{\alpha}(z^2) \cdot (z - z^2),$$

$$T_{7,6}(z) = (1 - z + z^2 - z^3) \cdot (-z - z^2) - z^{\alpha}(z^2) \cdot (z - z^2),$$

where $\alpha = \lambda_{4,3} + \mu_{4,3} + 2 = 2$. Thus,

$$\gamma_{7,6}(z) = S_{7,6}(z)/T_{7,6}(z) = z(-1 - z^4)/z(-1 + z^5). \tag{3.36}$$

Note that $\bar{\gamma}_{1,0}(z)$ is not used for computing $\gamma_{7,6}(z)$. It is used instead in formulae (3.23) and (3.24) for computing the predecessor of $\gamma_{7,6}(z)$. Since

$$z^{\lambda_{mn}} = z^{\bar{\lambda}_{\bar{m}\bar{n}}} = z^1 = \mathrm{GCD}\,(\bar{S}_{3,2}, \bar{T}_{3,2}),$$

it is known that the predecessor is $\gamma_{5,4}(z)$, which is determined by (3.23) and (3.24) to be

$$\gamma_{5,4}(z) = \frac{S_{5,4}(z)}{T_{5,4}(z)} = \frac{(-1 - z + z^2 - z^3 - 2z^5)}{(-1 - z + z^2 - z^3 + z^4)}. \qquad \square$$

3.2. Fast off-diagonal algorithm for a single power series.

3.2.1. The algorithm. The algorithm given in this section constructs the scaled Padé fraction γ_{mn} of type (m, n) for $A(z)$ in a quadratic fashion. The iteration assumes the existence of

$$\gamma_{MN}(z) = S_{MN}(z)/T_{MN}(z), \tag{3.37}$$

where $M = N + (m - n)$, and where for notational convenience we set

$$S_1 = S_{MN}(z) \tag{3.38}$$

and

$$T_1 = T_{MN}(z). \tag{3.39}$$

Also assumed to exist is the predecessor

$$\gamma_{M^*N^*}(z) = S_{M^*N^*}(z)/T_{M^*N^*}(z) \tag{3.40}$$

of $\gamma_{MN}(z)$ on the $(m-n)$th off-diagonal path of $\Gamma(A)$, where

$$M^* = M - \lambda_{MN} - 1,$$

$$N^* = N - \lambda_{MN} - 1$$

and

$$z^{\lambda_{MN}} = \text{GCD}\,(S_{MN}, T_{MN}).$$

Again for notational convenience, we set

$$S_0 = S_{M^*N^*}(z) \tag{3.41}$$

and

$$T_0 = T_{M^*N^*}(z). \tag{3.42}$$

To advance the solution from N to $N+s$ (that is, to construct $\gamma_{M+s,N+s}(z)$), where s is the step size, the algorithm first computes μ_{MN} such that

$$AT_1 - S_1 \bmod z^{M+N+2s+\lambda_{MN}+1} = z^{M+N+\mu_{MN}+1} R_1,$$

where

$$z^{\lambda_{MN}} = \text{GCD}\,(S_1, T_1)$$

and $R_1(0) \neq 0$ if $\mu_{MN} < 2s + \lambda_{MN}$.

If $\mu_{MN} \geqq s$, then $\gamma_{M+s,N+s}(z)$ is constructed trivially by means of Theorem 3.1. Otherwise, Theorems 3.4 and 3.5 are applied.

ALGORITHM 1: **OFFDIAG**

INPUT: A, m, n, where

(1) m and n are nonnegative integers with $m \geqq n$, and

(2) A is a unit power series. (Note that only $A \bmod z^{m+n+1}$ is required.)

OUTPUT: $\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix}$, where

(1) S_1/T_1 is the scaled Padé fraction of type (m, n) for A, and

(2) S_0/T_0 is the scaled Padé fraction of type $(m-\lambda_{mn}-1, n-\lambda_{mn}-1)$ for A, given that

$$z^{\lambda_{mn}} = \text{GCD}\,(S_1, T_1).$$

Step 1: # Initialization #

$$i \leftarrow -1$$

$$M \leftarrow (m-n)$$

$$N \leftarrow 0$$

$$\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \leftarrow \begin{bmatrix} A \bmod z^{M+1} & -z^{M-1} \\ 1 & 0 \end{bmatrix}$$

Step 2: # Calculation of step-size #

$$i \leftarrow i+1$$

$$s \leftarrow \min\{2^i - N, n - N\}$$

Step 3: # Termination criterion #

If $s = 0$ then exit

Step 4: # Calculation of scaling factor for S_1/T_1 #

Determine λ_{MN} such that $z^{\lambda_{MN}} = \text{GCD}\,(S_1, T_1)$

Step 5: # Computation of residual for $\gamma_{MN} = S_1/T_1$ #
Compute μ_{MN} and R_1 such that
$(AT_1 - S_1) \bmod z^{M+N+2s+\lambda_{MN}+1} = z^{M+N+\mu_{MN}+1}R_1$,
where $R_1(0) \neq 0$ if $\mu_{MN} < 2s + \lambda_{MN}$

Step 6: # Identification of Cases #
if $\mu_{MN} \geqq s$
then # Case of Theorem 3.1 #

$$\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \leftarrow \begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \begin{bmatrix} z^s & 0 \\ 0 & 1 \end{bmatrix}$$

go to step 12
else # Case of Theorem 3.4 and 3.5 #
go to step 7

Step 7: # Calculation of degrees for residual scaled Padé fractions #
$\bar{m} \leftarrow s + \lambda_{MN}$
$\bar{n} \leftarrow s - \mu_{MN} - 1$

Step 8: # Computation of residual for $\gamma_{M^*N^*}(z) = S_0/T_0$ #
Compute R_0 such that
$(A \cdot T_0 - S_0) \bmod z^{M+N+\bar{m}+\bar{n}-2\lambda_{MN}} = z^{M+N-2\lambda_{MN}-1}R_0$,
where $R_0(0) \neq 0$

Step 9: # Computation of residual power series #
$\bar{A} \leftarrow R_0/R_1 \bmod z^{\bar{m}+\bar{n}+1}$

Step 10: # Computation of residual scaled Padé fractions #

$$\begin{bmatrix} \bar{S}_1 & \bar{S}_0 \\ \bar{T}_1 & \bar{T}_0 \end{bmatrix} \leftarrow \textbf{OFFDIAG}\,(\bar{A}(z), \bar{m}, \bar{n})$$

Step 11: # Advancement of scaled Padé fraction computation #

$$\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \leftarrow \begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \begin{bmatrix} z^{-\lambda_{MN}} & 0 \\ 0 & -z^{\lambda_{MN}+\mu_{MN}+2} \end{bmatrix} \begin{bmatrix} \bar{S}_1 & \bar{S}_0 \\ \bar{T}_1 & \bar{T}_0 \end{bmatrix}$$

Step 12: # Calculation of degrees of S_1/T_1 #
$N \leftarrow N + s$
$M \leftarrow M + s$
go to step 2

3.2.2. Proof of algorithm validity. Let $\gamma_{mn}^{(A)}(z)$ be the scaled Padé fraction of type (m, n) for $A(z)$, where $m \geqq n \geqq -1$. As in (3.2), define

$$\mathbf{\Gamma}_N(A) = \{\gamma_{mn}^{(A)}(z) | m \geqq n,\ N \geqq n \geqq -1\} \tag{3.43}$$

to be the N-truncated Padé table for $A(z)$, with the additional provision that $m \geqq n$.
Also, define

$$\mathbf{\Omega}_N = \{\mathbf{\Gamma}_N(A) \text{ for all } A \in \mathbf{U}\} \tag{3.44}$$

for $N = 0, 1, \cdots$.

The proof proceeds by induction on $\mathbf{\Omega}_{2^i}$ for $i = 0, 1, 2, \cdots$. Clearly the algorithm is correct for $\mathbf{\Omega}_0$. It is shown that if the algorithm is correct for $\mathbf{\Omega}_N$, then it is also

correct for $\mathbf{\Omega}_{N+L}$, where

$$L=\begin{cases}1 & \text{if } N=0,\\ N & \text{if } N=2^i,\quad i\geqq 0.\end{cases} \tag{3.45}$$

Let $A(z)\in\mathbf{U}$ be an arbitrary unit power series. It is shown that the algorithm correctly computes $\gamma_{mn}^{(A)}(z)$, where $m\geqq n$ and $-1\leqq n\leqq N+L$, given that it is correct for $\mathbf{\Omega}_N$.

If $-1\leqq n\leqq N$, then $\gamma_{mn}^{(A)}(z)\in\mathbf{\Gamma}_N(A)\in\mathbf{\Omega}_N$, and by the inductive hypothesis the algorithm computes it correctly.

If $N<n\leqq N+L$, let $M=N+(m-n)$. By the inductive hypothesis, the algorithm (using $\log_2 N$ iterations) correctly computes $\gamma_{MN}^{(A)}(z)=S_{MN}^{(A)}(z)/T_{MN}^{(A)}(z)\in\mathbf{\Gamma}_N(A)\in\mathbf{\Omega}_N$, and its predecessor $\gamma_{M^*N^*}^{(A)}(z)$. It remains to show that one more iteration correctly gives $\gamma_{mn}^{(A)}(z)$ and its predecessor $\gamma_{m^*n^*}^{(A)}(z)$.

Let $s=n-N$ be given by step 2 of the algorithm and let $z^{\lambda_{MN}}=\mathrm{GCD}\,(S_{MN}^{(A)},\,T_{MN}^{(A)})$ be given by step 4. Then step 5 yields $\mu_{MN}\geqq 0$ and a polynomial $R_1(z)$ such that

$$(A(z)T_{MN}^{(A)}(z)-S_{MN}^{(A)}(z)) \bmod z^{M+N+2s+\lambda_{MN}+1}=R_1(z)z^{M+N+\mu_{MN}+1}, \tag{3.46}$$

where $\mu_{MN}\geqq 0$, and $R_1(0)\neq 0$ if $\mu_{MN}<2s+\lambda_{MN}$. Then two cases arise.

If $\mu_{MN}\geqq s$, then by Theorem 3.1, step 5 correctly yields $\gamma_{mn}^{(A)}(z)$. Furthermore, from (3.6) and (3.7),

$$z^{\lambda_{mn}}=\mathrm{GCD}\,(S_{mn}^{(A)}(z),\,T_{mn}^{(A)}(z))=z^{\lambda_{MN}+s}.$$

Consequently, the predecessor of $\gamma_{mn}^{(A)}(z)$ is given by

$$\gamma_{m^*n^*}^{(A)}(z)=\gamma_{M^*N^*}^{(A)}(z), \tag{3.47}$$

since

$$m^*=m-\lambda_{mn}-1=M-\lambda_{MN}-1=M^*$$

and

$$n^*=n-\lambda_{mn}-1=N-\lambda_{MN}-1=N^*.$$

If $\mu_{MN}<s$, then step 7 gives

$$\bar{n}=s-\mu_{MN}-1\leqq N$$

and

$$\bar{m}=s+\lambda_{MN}\geqq\bar{n}.$$

The polynomial $R_1(z)$, computed in step 5, therefore satisfies $R_1(0)\neq 0$ and is of degree $\bar{m}+\bar{n}$. In addition, step 8 produces a polynomial $R_0(z)$ of degree $\bar{m}+\bar{n}$ such that $R_0(0)\neq 0$. Consequently, enough terms are available in $R_1(z)$ and $R_0(z)$ to compute, in step 9, the residual unit power series $\bar{A}(z) \bmod z^{\bar{m}+\bar{n}+1}$ defined by (3.12). By the inductive hypothesis, step 10 therefore correctly computes $\gamma_{\bar{m}\bar{n}}^{(\bar{A})}(z)\in\mathbf{\Gamma}_N(\bar{A})\in\mathbf{\Omega}_N$ and its predecessor. It now follows by Theorems 3.4 and 3.5 that step 11 correctly computes $\gamma_{mn}^{(A)}(z)$ and its predecessor. □

3.2.3. Cost analysis. Let $C(m, n)$ be the cost of computing the scaled Padé fraction of type (m, n) and its predecessor for an arbitrary power series $A(z)\in\mathbf{U}$ using Algorithm 1. For the sake of simplicity, assume $0\leqq n\leqq m\leqq 2n$. The case that $m>2n$ is considered later. In this section, asymptotic estimates of $C(m, n)$ are derived by counting the number of operations (additions, subtractions, multiplications and divisions in $\mathbf{F}$) performed by the algorithm. A detailed cost analysis and an implementation of the

algorithm are described by Verheijen [18], who compares Algorithm 1 with other algorithms for calculating Padé fractions.

When obtaining the asymptotic cost estimates, it is assumed that the algorithm makes use of fast methods for polynomial and power series arithmetic. Using fast Fourier transforms, two polynomials with coefficients in the field **F** and of degree M and N, respectively, can be multiplicated in $O((M+N)\log(M+N))$ operations in **F**. Using Newton's method and fast multiplication of polynomials, the first N terms of the quotient of two power series in **U** can be obtained in $O(N \log N)$ operations in **F**. These and other fast methods for polynomial and power series arithmetic are described, for example, in Aho et al. [1] and in Lipson [12].

Let $k = \lceil \log n \rceil$. Then it is easy to verify that the algorithm terminates after k iterations, and that after the execution of step 2 of iteration i

$$s = \begin{cases} 1, & i = 0, \\ 2^{i-1}, & 0 < i < k, \\ n - 2^{k-1}, & i = k. \end{cases} \tag{3.48}$$

Consequently, during iteration i,

$$N = \begin{cases} 0, & i = 0, \\ 2^{i-1}, & \text{otherwise}, \end{cases} \tag{3.49}$$

and

$$M = N + (m - n). \tag{3.50}$$

Assume that during iteration i, the scaled Padé fraction S_1/T_1 of type (M, N) and its predecessor S_0/T_0 are available.

The first nontrivial step requires the computation of the residual $R_1(z)$ in step 5. Let

$$A_1(z) = \sum_{j=0}^{M} a_j z^j \tag{3.51}$$

and

$$A_2(z) = \sum_{j=0}^{N+2s+\lambda_{MN}-1} a_{M+j+1} z^j. \tag{3.52}$$

With λ_{MN} given by step 4, it is known that

$$\begin{aligned} AT_1 - S_1 \bmod z^{M+N+2s+\lambda_{MN}+1} &= A_1 T_1 + z^{M+1} A_2 T_1 - S_1 \bmod z^{M+N+2s+\lambda_{MN}+1} \\ &= O(z^{M+N+1}). \end{aligned} \tag{3.53}$$

Since $A_1 T_1$ and S_1 are both of degree at most $M+N$, then μ_{MN} and $R_1(z)$ can be obtained directly from $z^{M+1} A_2 T_1$. The product $A_2(z^{-\lambda_{MN}} T_1)$ is a polynomial of at most degree $2N + 2s - 1 < 4N$. Thus, using fast polynomial multiplication, step 5 can be executed in $O(N \log N)$ operations.

If it is determined as a result of step 5 that $\mu_{MN} \geqq s$, then step 6 is performed trivially and the iteration is complete. Otherwise, the algorithm continues in step 8 with the calculation of the residual R_0 of the predecessor scaled Padé fraction S_0/T_0. Making the same observations as in step 4, it follows that R_0 can be obtained from the product of $A_3 T_0$, where

$$A_3 = \sum_{j=0}^{N+\bar{m}+\bar{n}-\lambda_{MN}-1} a_{M-\lambda_{MN}+j} z^j \tag{3.54}$$

and T_0 is a polynomial of degree at most $N-\lambda_{MN}-1$. Since the degree of the product is bounded by $2N+\bar{m}+\bar{n}-2\lambda_{MN}-2<4N$, step 8 can be executed in $O(N \log N)$ operations.

The computation of the residual power series $\bar{A}(z) \bmod z^{\bar{m}+\bar{n}+1}$ in step 9 requires the computation of the first $\bar{m}+\bar{n}=2s+\lambda_{MN}-\mu_{MN}-1 \leqq 3N$ terms of the quotient of R_0/R_1. Again, this may be performed in $O(N \log N)$ operations.

In step 10, the recursive call of Algorithm 1, in order to compute the scaled Padé fraction of type $(\bar{m}, \bar{n})$ for $\bar{A}(z)$, requires $C(\bar{m}, \bar{n})$ operations by assumption. For later purposes, it is important to observe that $0 \leqq \bar{n} \leqq N$ and that $\bar{n} \leqq \bar{m} \leqq 2N$.

The final nontrivial step requires eight polynomial multiplications to obtain the scaled Padé fraction of type $(M+s, N+s)$ and its predecessor for $A(z)$. Since $M \leqq 2N$, each of the polynomial products are of degree at most $M+s<3N$. Consequently, step 8 can be executed in $O(N \log N)$ operations.

It is an easy matter to show that

$$C(m_1, n_1) \leqq C(m_2, n_2), \tag{3.55}$$

whenever $m_1 \leqq m_2$ and $n_1 \leqq n_2$. The total cost of the ith iteration is then bounded by

$$\begin{aligned} C(\bar{m}, \bar{n})+c(2N) \log (2N) &\leqq C(2N, N)+c(2N) \log (2N) \\ &\leqq C(2^i, 2^{i-1})+ci2^i \end{aligned} \tag{3.56}$$

operations, for an appropriate constant c. Consequently, we have

THEOREM 3.6. *Given that $0 \leqq n \leqq m \leqq 2n$, Algorithm 1 can compute the scaled Padé fraction of type (m, n) for $A(z) \in \mathbf{U}$ in time $O(n \log^2 n)$.*

Proof. Consider the recurrence relation

$$C(2^{k+1}, 2^k)=\sum_{i=1}^{k}[C(2^i, 2^{i-1})+ci2^i], \qquad k \geqq 1,$$

where c is a positive constant. Then

$$\begin{aligned} C(2^{k+1}, 2^k) &= \sum_{i=1}^{k-1}[C(2^i, 2^{i-1})+ci2^i]+C(2^k, 2^{k-1})+ck2^k \\ &= 2C(2^k, 2^{k-1})+ck2^k. \end{aligned}$$

With $n=2^k$, results on recurrence relations (see Bentley et al. [3]) then yield

$$C(2n, n)=n\left[C(2,1)+c\sum_{i=1}^{k} i\right]=n[C(2,1)+ck(k+1)/2]=O(n \log^2 n).$$

The theorem now follows, since from (3.56) for m and n satisfying $0 \leqq n \leqq m \leqq 2n$

$$C(m, n) \leqq \sum_{i=1}^{k}[C(2^i, 2^{i-1})+ci2^i]. \qquad \square$$

LEMMA 3.7. *Let $m>n$, and let $A(z) \in \mathbf{U}$. Determine an integer $\delta \geqq 1$ such that*

$$A(z)=A_1(z)+z^{m-n+\delta}A_2(z), \tag{3.57}$$

where

$$A_1(z)=A(z) \bmod z^{m-n+1} \tag{3.58}$$

and $A_2(0) \neq 0$ if $\delta<\infty$. If $\delta \leqq n$, let $S_{n,n-\delta}(z)/T_{n,n-\delta}(z)$ be the scaled Padé fraction of type $(n, n-\delta)$ for $1/A_2(z)$. Then the scaled Padé fraction $S_{mn}(z)/T_{mn}(z)$ of type (m, n)

for $A(z)$ is given by

$$S_{mn}(z) = \begin{cases} A_1(z)S_{n,n-\delta}(z) + z^{m-n+\delta}T_{n,n-\delta}(z) & \text{if } \delta \leqq n, \\ A_1(z)z^n & \text{otherwise,} \end{cases} \tag{3.59}$$

$$T_{mn}(z) = \begin{cases} S_{n,n-\delta}(z) & \text{if } \delta \leqq n, \\ z^n & \text{otherwise.} \end{cases} \tag{3.60}$$

Proof. See Choi [7]. □

THEOREM 3.8. *For arbitrary $m \geqq n$, Algorithm 1 can compute the scaled Padé fraction of type (m, n) for $A(z) \in \mathbf{U}$ in time $O(m \log m) + O(n \log^2 n)$.*

Proof. If $\delta > n$ in (3.57), the result is trivial.

If $\delta \leqq n$, computation of the first $2n - \delta + 1$ terms of $1/A_2(z)$ requires $O(n \log n)$ operations. Using (3.55),

$$C(n, n-\delta) \leqq C(n, n),$$

and consequently by Theorem 3.6, it follows that the cost of computing the scaled Padé fraction $S_{n,n-\delta}(z)/T_{n,n-\delta}(z)$ for $1/A_2(z)$ is bounded by $O(n \log^2 n)$ operations. Finally, the cost of computing $S_{mn}(z)$ in equation (3.59) is $O(m \log m)$ operations. □

3.3. Fast off-diagonal algorithm for a quotient power series. Let $A(z)$, $B(z)$ be unit power series and let

$$C(z) = -A(z)/B(z) \tag{3.61}$$

be the quotient power series. Given the nonnegative integers $m \geqq n$, then Algorithm 1 OFFDIAG can be used to compute the scaled Padé fractions $S_{mn}(z)/T_{mn}(z)$ of type (m, n) for $C(z)$. As a result,

$$A(z)T_{mn}(z) + B(z)S_{mn}(z) = O(z^{m+n+1}). \tag{3.62}$$

Before applying the algorithm, the quotient

$$C(z) \bmod z^{m+n+1} = -A(z)/B(z) \bmod z^{m+n+1} \tag{3.63}$$

must be calculated. However, this division need be computed modulo z^{m-n+1}, only, by modifying Algorithm 1 as follows:

ALGORITHM 2: **OFFDIAG**

INPUT: A, B, m, n, where

(1) m and n are nonnegative integers with $m \geqq n$, and
(2) A, B are unit power series. (Note that only A mod z^{m+n+1} and B mod z^{m+n+1} are required.)

OUTPUT: $\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix}$, where

(1) S_1/T_1 is the scaled Padé fraction of type (m, n) for $-A/B$, and
(2) S_0/T_0 is the predecessor scaled Padé fraction of type $(m - \lambda_{mn} - 1, n - \lambda_{mn} - 1)$ for $-A/B$, given that

$$z^{\lambda_{mn}} = \text{GCD}(S_1, T_1).$$

Step 1: # Initialization #

$i \leftarrow -1$

$M \leftarrow (m-n)$

$N \leftarrow 0$

$$\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \leftarrow \begin{bmatrix} -A/B \bmod z^{M+1} & z^{M-1} \\ 1 & 0 \end{bmatrix}$$

Step 2: # Calculation of step-size #

$i \leftarrow i+1$

$s \leftarrow \min\{2^i - N, n-N\}$

Step 3: # Termination criterion #

If $s=0$ then exit

Step 4: # Calculation of scaling factor for S_1/T_1 #

Determine λ_{MN} such that

$z^{\lambda_{MN}} = \mathrm{GCD}(S_1, T_1)$

Step 5: # Computation of residual for $\gamma_{MN}(z) = S_1/T_1$ #

Compute μ_{MN} and R_1 such that

$(AT_1 + BS_1) \bmod z^{M+N+2s+\lambda_{MN}+1} = z^{M+N+\mu_{MN}+1} R_1$,

where $R_1(0) \neq 0$ if $\mu_{MN} < 2s + \lambda_{MN}$

Step 6: # Identification of Cases #

if $\mu_{MN} \geqq s$

then # Case of Theorem 3.1 #

$$\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \leftarrow \begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \begin{bmatrix} z^s & 0 \\ 0 & 1 \end{bmatrix}$$

go to step 11

else # Case of Theorem 3.4 and 3.5 #

go to step 7

Step 7: # Calculation of degrees for residual scaled Padé fractions #

$\bar{m} \leftarrow s + \lambda_{MN}$

$\bar{n} \leftarrow s - \mu_{MN} - 1$

Step 8: # Computation of residual for $\gamma_{M^*N^*}(z) = S_0/T_0$ #

Compute R_0 such that

$(A \cdot T_0 + B \cdot S_0) \bmod z^{M+N+\bar{m}+\bar{n}-2\lambda_{MN}} = R_0 z^{M+N-2\lambda_{MN}-1}$

where $R_0(0) \neq 0$

Step 9: # Computation of residual scaled Padé fractions #

$$\begin{bmatrix} \bar{S}_1 & \bar{S}_0 \\ \bar{T}_1 & \bar{T}_0 \end{bmatrix} \leftarrow \textbf{OFFDIAG}(R_0, R_1, \bar{m}, \bar{n})$$

Step 10: # Advancement of scaled Padé fraction computation #

$$\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \leftarrow \begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \begin{bmatrix} z^{-\lambda_{MN}} & 0 \\ 0 & z^{\lambda_{MN}+\mu_{MN}+2} \end{bmatrix} \begin{bmatrix} \bar{S}_1 & \bar{S}_0 \\ \bar{T}_1 & \bar{T}_0 \end{bmatrix}$$

Step 11: # Calculation of degrees of S_1/T_1 #
$M \leftarrow M+s$
$N \leftarrow N+s$
go to step 2

Algorithm 2 is a generalization of Algorithm 1, since it can be used to produce scaled Padé fractions for a single power series $A(z)$ simply by setting $B(z)=-1$. It differs from Algorithm 1 in that the division

$$\bar{A}(z) = -R_0(z)/R_1(z) \bmod z^{\bar{m}+\bar{n}+1} \tag{3.64}$$

in step 9 of Algorithm 1 is avoided. Instead, the division is delayed (i.e., immediately subsequent to, rather than prior to, the recursive call of OFFDIAG) until the initialization

$$S_1 = -A(z)/B(z) \bmod z^{m-n+1} \tag{3.65}$$

in step 1 of Algorithm 2.

There are also various sign changes introduced in steps 1, 5, 8 and 10. These account for the fact that the algorithm deals with the power series $-A(z)/B(z)$ rather than $A(z)/B(z)$. This notational change simplifies the development of subsequent results.

For implementation purposes, Algorithm 2 can result in considerable savings in cost. Practically, fast division is significantly slower than fast multiplication, by an asymptotic constant of approximately 7 (see Verheijen [18]). For example, if $-A(z)/B(z)$ is normal, then $\bar{m}-\bar{n}+1=2$ and the division in step 1 of Algorithm 2 becomes trivial. However, the asymptotic cost of Algorithm 2 remains the same as the asymptotic cost of Algorithm 1 given in § 3.2.3.

The proof of the correctness of Algorithm 2 is nearly identical to the proof of correctness of Algorithm 1, and therefore it is not given.

3.4. Classical off-diagonal algorithm for a quotient power series. Let S_1/T_1 be the scaled Padé fraction of type (M, N) for $-A(z)/B(z)$ such that

$$z^{\lambda_{MN}} = \mathrm{GCD}\,(S_1, T_1) = 1.$$

That is, $\lambda_{MN}=0$. Then

$$A \cdot T_1 + B \cdot S_1 = z^{M+N+\mu_{MN}+1} R_1,$$

where $R_1(0) \neq 0$ if $\mu_{MN} < \infty$. Thus, $(z^k S_1)/(z^k T_1)$ is the scaled Padé fraction of type $(M+k, N+k)$ for $-A(z)/B(z)$ whenever $0 \leqq k \leqq \mu_{MN}$. If $\mu_{MN} < \infty$, the next distinct Padé fraction for $-A(z)/B(z)$ along the $(M-N)$th off-diagonal path is of type $(M+s, N+s)$, where

$$s = \mu_{mn} + 1.$$

By so selecting the step-size s, Algorithm 2 may be used to compute all Padé fractions along the $(M-N)$th off-diagonal path. With this choice of s, step 7 in Algorithm 2 gives

$$\bar{m} = \mu_{MN} + 1,$$

$$\bar{n} = 0.$$

Consequently, the recursive call of OFFDIAG in step 9 of Algorithm 2 reduces to

$$\begin{bmatrix} \bar{S}_1 & \bar{S}_0 \\ \bar{T}_1 & \bar{T}_0 \end{bmatrix} \leftarrow \begin{bmatrix} -R_0/R_1 \bmod z^{\mu_{MN}+2} & z^{\mu_{MN}} \\ 1 & 0 \end{bmatrix},$$

and step 10 then yields

$$\begin{bmatrix} S_1 & S_0 \\ T_1 & T_0 \end{bmatrix} \leftarrow \begin{bmatrix} S_1 \cdot \bar{S}_1 + z^{\mu_{MN}+2} \cdot S_0 & z^{\mu_{MN}} \cdot S_1 \\ T_1 \cdot \bar{S}_1 + z^{\mu_{MN}+2} \cdot T_0 & z^{\mu_{MN}} \cdot T_1 \end{bmatrix}.$$

Since GCD $(\bar{S}_1, \bar{T}_1) = 1$, it follows from the proof of Theorem 3.4 that

$$\text{GCD}\,(S_1 \cdot \bar{S}_1 + z^{\mu_{MN}+2} \cdot S_0,\ T_1 \cdot \bar{S}_1 + z^{\mu_{MN}+2} \cdot T_0) = 1.$$

Thus, the above computations can be repeated for the scaled Padé fraction of type $(M+s, N+s)$.

The full details are provided in Algorithm 3, below. To simplify the presentation of subsequent results, at the ith iteration, the scaled Padé fractions S_1/T_1 of type (M, N) is denoted by S_i/T_i and its predecessor S_0/T_0 by S_{i-1}/T_{i-1}. In addition, the residual power series R_1 and R_0 are denoted by R_i and R_{i-1}, respectively.

ALGORITHM 3: **OFFDIAG**

INPUT: A, B, m, n, where

(1) m and n are nonnegative integers with $m \geqq n$, and
(2) A and B are unit power series. (Note that only $A \bmod z^{m+n+1}$ and $B \bmod z^{m+n+1}$ are required.)

OUTPUT: $\begin{bmatrix} S_{i+1} & S_i \\ T_{i+1} & T_i \end{bmatrix}$, where

(1) S_{i+1}/T_{i+1} is the scaled Padé fraction of type (m, n) for $-A/B$, and
(2) S_i/T_i is the scaled Padé fraction of type $(m-\lambda_{mn}-1, n-\lambda_{mn}-1)$ for $-A/B$, given that

$$z^{\lambda_{mn}} = \text{GCD}\,(S_{i+1}, T_{i+1}).$$

Step 1: # Initialization #

$i \leftarrow -1$
$M \leftarrow (m-n)$
$N \leftarrow 0$

$$\begin{bmatrix} S_{i+1} & S_i \\ T_{i+1} & T_i \end{bmatrix} \leftarrow \begin{bmatrix} -A/B \bmod z^{M+1} & z^{M-1} \\ 1 & 0 \end{bmatrix}$$

Step 2: # Termination criterion #

If $N = n$
then exit
else $i \leftarrow i+1$

Step 3: # Computation of residual for $\gamma_{MN} = S_i/T_i$ #

Compute μ_{MN} and R_i such that
$(A \cdot T_i + B \cdot S_i) \bmod z^{M+N+2\mu_{MN}+3} = z^{M+N+\mu_{MN}+1} R_i$,
where $R_i(0) \neq 0$ if $\mu_{MN} < 2(n-N)$.

Step 4: # Calculation of step-size #

$s \leftarrow \min\{\mu_{MN}+1, n-N\}$

Step 5: # Identification of Cases #
if $\mu_{MN} \geqq s$
then # Case of Theorem 3.1 #

$$\begin{bmatrix} S_{i+1} & S_i \\ T_{i+1} & T_i \end{bmatrix} \leftarrow \begin{bmatrix} S_i & S_{i-1} \\ T_i & T_{i-1} \end{bmatrix} \begin{bmatrix} z^s & 0 \\ 0 & 1 \end{bmatrix}$$

go to step 8
else # Case of Theorem 3.4 and 3.5 #
go to step 6

Step 6: # Computation of residual for $\gamma_{M^*N^*}(z) = S_{i-1}/T_{i-1}$ #
Compute R_{i-1} such that
$(A \cdot T_{i-1} + B \cdot S_{i-1}) \bmod z^{M+N+\mu_{MN}+1} = z^{M+N-1} R_{i-1}$,
where $R_{i-1}(0) \neq 0$

Step 7: # Advancement of scaled Padé fraction computation #

$$\begin{bmatrix} S_{i+1} & S_i \\ T_{i+1} & T_i \end{bmatrix} \leftarrow \begin{bmatrix} S_i & S_{i-1} \\ T_i & T_{i-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & z^{\mu_{MN}+2} \end{bmatrix} \begin{bmatrix} -R_{i-1}/R_i \bmod z^{\mu_{MN}+2} & z^{\mu_{MN}} \\ 1 & 0 \end{bmatrix}$$

Step 8: # Calculation of degrees of S_{i+1}/T_{i+1} #
$N \leftarrow N + s$
$M \leftarrow M + s$
go to step 2

In the case that $B(z) = -1$ and $m = n$, Algorithm 3 is precisely the algorithm given by Cabay and Kao [6] for computing diagonal Padé fractions for a single power series. Their $O(n^2)$ algorithm (since steps 3 and 6 require only $\mu_{MN}+2$ terms of product polynomials, and since one of the multipliers in step 7 is a polynomial of degree $\mu_{MN}+1$, for small μ_{MN}, no advantage can be gained from using fast methods for polynomial arithmetic) is shown to be faster than other $O(n^2)$ algorithms for computing diagonal Padé fractions, such as those of Rissanen [16] and Trench [17].

The more interesting observation about Algorithm 3 is made in the next section, however. It is shown that, when the given quotient power series is a rational function, Algorithm 3 along one specific off-diagonal path corresponds exactly to Euclid's extended algorithm for computing the greatest common divisor of the numerator and denominator of the given rational function. In this sense, Algorithm 3 is a generalization of Euclid's extended algorithm. It is for this reason that we choose to call Algorithm 3 classical.

4. Greatest common divisor computations of polynomials.

4.1. Introduction. In this section, Algorithm 2 and Algorithm 3 are examined when they are applied to $-A(z)/B(z)$, where $A(z)$ and $B(z)$ are polynomials of degrees m and n respectively. In addition, it is assumed that $A(0) \neq 0$ and $B(0) \neq 0$.

To permit the analysis, we need the following

DEFINITION. The reciprocal of a polynomial

$$P(z) = p_0 + p_1 z + \cdots + p_n z^n \tag{4.1}$$

of at most degree n is defined to be

$$P^R(z) = p_0 z^n + p_1 z^{n-1} + \cdots + p_n = z^n P(z^{-1}). \tag{4.2}$$

The name originates from the fact that the zeros of $P^R(z)$ are the reciprocals of those of $P(z)$ if $p_0 p_n \neq 0$.

Clearly

$$[P^R(z)]^R = P(z); \tag{4.3}$$

that is, the operation of forming the reciprocal polynomial is involutary. Also

$$[P(z) \cdot Q(z)]^R = P^R(z) \cdot Q^R(z), \tag{4.4}$$

and

$$[z^\lambda P(z)]^R = P^R(z). \tag{4.5}$$

Let $A(z)$ be a unit power series. The first $n+1$ terms of $A(z)$ are denoted by

$$A_n(z) = \sum_{i=0}^{n} a_i z^i = A(z) \bmod z^{n+1}. \tag{4.6}$$

The reciprocal $A_n^R(z)$ of $A_n(z)$ is then given by

$$A_n^R(z) = \sum_{i=0}^{n} a_{n-i} z^i. \tag{4.7}$$

4.2. Euclid's extended algorithm.

THEOREM 4.1. *Algorithm* 3 *applied to* $-A(z)/B(z)$, *where* $A(z)$ *and* $B(z)$ *are polynomials of degree* m *and* n, *respectively, is equivalent to Euclid's extended algorithm for computing the greatest common divisor of* $A^R(z)$ *and* $B^R(z)$.

Proof. For completeness in presentation, assume that modulo operations are not performed in steps 3 and 6 of Algorithm 3. Thus,

$$A \cdot T_i + B \cdot S_i = z^{M+N+\mu_{MN}+1} R_i, \tag{4.8}$$

$$A \cdot T_{i-1} + B \cdot S_{i-1} = z^{M+N-1} R_{i-1}, \tag{4.9}$$

where $R_i(z)$ and $R_{i-1}(z)$ are polynomials of degrees $n-N-\mu_{MN}-1$ and $n-N$, respectively.[4] By taking reciprocals of (4.8) and (4.9), it follows that

$$A^R \cdot T_i^R + B^R \cdot S_i^R = R_i^R \tag{4.10}$$

and

$$A^R \cdot T_{i-1}^R + B^R \cdot S_{i-1}^R = R_{i-1}^R. \tag{4.11}$$

Let

$$Q_{i+1}(z) = R_{i-1}(z)/R_i(z) \bmod z^{\mu_{MN}+2} \tag{4.12}$$

be a polynomial of degree $\mu_{MN}+1$. Then

$$R_{i-1}(z) - Q_{i+1}(z) \cdot R_i(z) = z^{\mu_{MN}+2} \bar{R}(z), \tag{4.13}$$

for some polynomial $\bar{R}(z)$ of degree $n-N-\mu_{MN}-2$. Taking reciprocals of (4.13), it follows that

$$R_{i-1}^R(z) - Q_{i+1}^R(z) \cdot R_i^R(z) = \bar{R}^R(z). \tag{4.14}$$

That is, $Q_{i+1}^R(z)$ and $\bar{R}^R(z)$ are the quotient and remainder, respectively, on division of $R_{i-1}^R(z)$ by $R_i^R(z)$.

With $Q_{i+1}(z)$ defined by (4.12), step 7 of Algorithm 3 yields

$$\begin{bmatrix} S_{i+1} & S_i \\ T_{i+1} & T_i \end{bmatrix} \leftarrow \begin{bmatrix} z^{\mu_{MN}+2} \cdot S_{i-1} - Q_{i+1} \cdot S_i & z^{\mu_{MN}} \cdot S_i \\ z^{\mu_{MN}+2} \cdot T_{i-1} - Q_{i+1} \cdot T_i & z^{\mu_{MN}} \cdot T_i \end{bmatrix}, \tag{4.15}$$

[4] By convention, a polynomial of negative degree is the zero polynomial.

which on taking reciprocals becomes

$$\begin{bmatrix} S_{i+1}^R & S_i^R \\ T_{i+1}^R & T_i^R \end{bmatrix} \leftarrow \begin{bmatrix} S_{i-1}^R - Q_{i+1}^R \cdot S_i^R & S_i^R \\ T_{i-1}^R - Q_{i+1}^R \cdot T_i^R & T_i^R \end{bmatrix}. \tag{4.16}$$

Looking ahead one iteration, we obtain

$$\begin{aligned} R_{i+1}^R &= A^R \cdot T_{i+1}^R + B^R \cdot S_{i+1}^R \\ &= A^R \cdot [T_{i-1}^R - Q_{i+1}^R \cdot T_i^R] + B^R \cdot [S_{i-1}^R - Q_{i+1}^R \cdot S_i^R] \\ &= R_{i-1}^R - Q_{i+1}^R \cdot R_i^R. \end{aligned} \tag{4.17}$$

Thus, $\bar{R}^R(z) = R_{i+1}^R(z)$ in (4.14).

Summarizing, let Q_{i+1}^R be the quotient on division of R_{i-1}^R by R_i^R. Then

$$\begin{aligned} R_{i+1}^R &= R_{i-1}^R - Q_{i+1}^R \cdot R_i^R, \\ S_{i+1}^R &= S_{i-1}^R - Q_{i+1}^R \cdot S_i^R, \\ T_{i+1}^R &= T_{i-1}^R - Q_{i+1}^R \cdot T_i^R, \end{aligned} \tag{4.18}$$

which are the fundamental relations describing Euclid's extended algorithm.

To complete the proof, it remains to show that the initial conditions for Euclid's extended algorithm are satisfied. Initialization in step 1 of Algorithm 3 yields

$$\begin{bmatrix} S_0 & S_{-1} \\ T_0 & T_{-1} \end{bmatrix} = \begin{bmatrix} -Q_0 & z^{m-n-1} \\ 1 & 0 \end{bmatrix}, \tag{4.19}$$

where, as in (4.12), Q_0^R is the quotient on division of A^R by B^R. Steps 3 and 6 with $i = 0$ then become

$$A \cdot 1 + B \cdot (-Q_0) = z^{m-n+\mu_{m-n,0}+1} R_0 \tag{4.20}$$

and

$$A \cdot 0 + B \cdot z^{m-n-1} = z^{m-n-1} R_{-1}. \tag{4.21}$$

Taking reciprocals of (4.19), (4.20) and (4.21), it follows that

$$\begin{aligned} R_{-1}^R &= B^R, \quad S_{-1}^R = 1, \quad T_{-1}^R = 0, \\ R_0^R &= A^R - Q_0^R \cdot B^R, \quad S_0^R = -Q_0^R, \quad T_0^R = 1. \end{aligned} \tag{4.22}$$

□

COROLLARY 4.2. *At the* ith *iteration of Algorithm* 3, *let* S_i/T_i *be the scaled Padé fraction of type* (M, N) *for* $-A/B$, *such that*

$$A \cdot T_i + B \cdot S_i = z^{M+N+\mu_{MN}+1} R_i.$$

Then,

$$\begin{aligned} R_{i-1} \cdot S_i - z^{\mu_{MN}+2} R_i \cdot S_{i-1} &= (-1)^i \cdot A, \\ R_{i-1} \cdot T_i - z^{\mu_{MN}+2} R_i \cdot T_{i-1} &= (-1)^{i+1} \cdot B, \\ S_{i-1} \cdot T_i - S_i \cdot T_{i-1} &= (-1)^{i+1} z^{M+N-1}. \end{aligned} \tag{4.23}$$

Furthermore, the algorithm terminates for some $i = k$, *where* $k < n$. *On termination,* S_{k+1}/T_{k+1} *is the scaled Padé fraction of type* (m, n) *for* $-A/B$ *such that*

$$A \cdot T_{k+1} + B \cdot S_{k+1} = 0. \tag{4.24}$$

In addition,

$$A \cdot T_k + B \cdot S_k = z^{m+n-2\lambda_{mn}-1} R_k, \tag{4.25}$$

where

$$z^{\lambda_{mn}} = \mathrm{GCD}\,(S_{k+1}, T_{k+1})$$

and

$$R_k = \mathrm{GCD}\,(A, B).$$

Proof. From Euclid's extended algorithm (4.18) with initial conditions (4.22), it follows that (see, for example, McEliece and Shearer [13])

$$\begin{aligned} R^R_{i-1} \cdot S^R_i - R^R_i \cdot S^R_{i-1} &= (-1)^i \cdot A^R, \\ R^R_{i-1} \cdot T^R_i - R^R_i \cdot T^R_{i-1} &= (-1)^{i+1} \cdot B^R, \\ S^R_{i-1} \cdot T^R_i - S^R_i \cdot T^R_{i-1} &= (-1)^{i+1}. \end{aligned} \tag{4.26}$$

By using the correspondence established in Theorem 4.1 and taking reciprocals, equations (4.26) result in equations (4.23).

Furthermore, Euclid's extended algorithm terminates for some $k < n$ when

$$A^R \cdot T^R_{k+1} + B^R \cdot S^R_{k+1} = R^R_{k+1} = 0 \tag{4.27}$$

and

$$A^R \cdot T^R_k + B^R \cdot S^R_k = R^R_k = \mathrm{GCD}\,(A^R, B^R), \tag{4.28}$$

where $\lambda = \partial(R^R_k)$, $\partial(S^R_{k+1}) \leqq m - \lambda$, $\partial(T^R_{k+1}) \leqq n - \lambda$, $\partial(S^R_k) < m - \lambda$ and $\partial(T^R_k) < n - \lambda$.

Taking reciprocals of (4.28) with respect to $z^{m+n-\lambda-1}$ results in

$$A \cdot T_k + B \cdot S_k = z^{m+n-2\lambda-1} \cdot R_k,$$

where $R_k(0) \neq 0$. Thus, S_k / T_k is the scaled Padé fraction of type $(m - \lambda - 1, n - \lambda - 1)$ for $-A/B$. Taking reciprocals of (4.27) with respect to z^{m+n} gives

$$A \cdot T_{k+1} + B \cdot S_{k+1} = 0, \tag{4.29}$$

where $\partial(S_{k+1}) \leqq m$ and $\partial(T_{k+1}) \leqq n$. Thus, S_{k+1} / T_{k+1} is the scaled Padé fraction of type (m, n) such that

$$\mathrm{GCD}\,(S_{k+1}, T_{k+1}) = z^{\lambda}. \tag{4.30}$$

Thus $\lambda = \lambda_{mn}$. □

As a consequence of Corollary 4.2. Algorithm 3 computes co-multipliers S_k and T_k, only, such that

$$A^R \cdot T^R_k + B^R \cdot S^R_k = R^R_k = \mathrm{GCD}\,(A^R, B^R). \tag{4.31}$$

The remainder R^R_k is available only if the multiplications in steps 3 and 6 are performed without the modulo operation.

4.3. Fast GCD computations.

THEOREM 4.3. *Algorithm 2 applied to $-A(z)/B(z)$, where $A(z)$ and $B(z)$ are polynomials of degree m and n, respectively, returns the scaled Padé fraction S_1 / T_1 of type (m, n) such that*

$$A^R \cdot T^R_1 + B^R \cdot S^R_1 = 0 \tag{4.32}$$

and its predecessor S_0/T_0 *of type* $(m-\lambda_{mn}-1, n-\lambda_{mn}-1)$ *such that*

$$A^R \cdot T_0^R + B^R \cdot S_0^R = R_0^R, \tag{4.33}$$

where

$$z^{\lambda_{mn}} = \mathrm{GCD}(S_1, T_1)$$

and

$$R_0 = \mathrm{GCD}(A, B).$$

Proof. Since scaled Padé fractions are unique, the result is an immediate consequence of Corollary 4.2. □

The greatest common divisor R_0 is not explicitly computed by Algorithm 2. However, $\partial(R_0) = \lambda_{mn} \leqq n$. Using fast multiplication, it can therefore be determined in $O(n \log n)$. As a consequence of this and Theorem 3.8, we have

THEOREM 4.4. *Algorithm* 2 *can compute the greatest common divisor, the cofactors and the comultipliers of two polynomials of degrees* m *and* n, *where* $m \geqq n$, *in time* $O(m \log m) + O(n \log^2 n)$.

Thus, Algorithm 2 for GCD computations is basically of the same asymptotic complexity as the fast algorithms of Moenck [14], Aho et al. [1] and Brent et al. [4], which are of complexity $O((m+n)\log^2(m+n))$. However, Algorithm 2 has the advantage of being partly iterative (approximately half as many recursive calls are used in comparison with the other fast methods). This can result in significant cost savings in an implementation environment (Verheijen [18]).

4.4. Antidiagonal computations. Let

$$A(z) = \sum_{i=0}^{\infty} a_i z^i \tag{4.34}$$

be a unit power series, and let

$$A_d^R(z) = \sum_{i=0}^{d} a_{d-i} z^i, \qquad d \geqq 0, \tag{4.35}$$

be the reciprocal of the first $d+1$ terms of $A(z)$. In this section, we examine the intermediate results obtained by Algorithm 2[5] while computing the scaled Padé fraction of type (n, n), $2n \leqq d$, for the two polynomials $1 + zA_d^R(z)$ and $B^R(z) = -1$.

At the ith iteration of Algorithm 2, the scaled Padé fraction $S_{NN}(z)/T_{NN}(z)$ of type (N, N) for $-\{1+zA_d^R(z)\}/B^R(z)$ and its predecessor $S_{N^*N^*}(z)/T_{N^*N^*}(z)$ are determined such that

$$\{1+zA_d^R(z)\} \cdot T_{NN}(z) - S_{NN}(z) = z^{2N+\mu_{NN}+1} R_1(z), \tag{4.36}$$

$$\{1+zA_d^R(z)\} \cdot T_{N^*N^*}(z) - S_{N^*N^*}(z) = z^{2N^*+1} R_0(z), \tag{4.37}$$

where

$$N^* = N - \lambda_{NN} - 1$$

and

$$z^{\lambda_{NN}} = \mathrm{GCD}(S_{NN}, T_{NN}).$$

[5] It is assumed that no truncation of polynomial operations takes place.

By taking reciprocals of (4.36) and (4.37), it follows that

$$A_d(z) \cdot T^R_{NN}(z) - R^R_1(z) = z^{d+1}\{S^R_{NN}(z) - T^R_{NN}(z)\}, \tag{4.38}$$

$$A_d(z) \cdot T^R_{N^*N^*}(z) - R^R_0(z) = z^{d+1}\{S^R_{N^*N^*}(z) - T^R_{N^*N^*}(z)\} \tag{4.39}$$

where

$$\partial(R^R_1) = d - N - \mu_{NN} \tag{4.40}$$

and

$$\partial(R^R_0) = d - N^*. \tag{4.41}$$

THEOREM 4.5. *Let $M = d - N$ and $M^* = d - N^*$. Then the scaled Padé fractions of type (M, N) and (M^*, N^*) for $A(z)$ are $R^R_1(z)/T^R_{NN}(z)$ and $R^R_0(z)/T^R_{N^*N^*}(z)$, respectively.*

Proof. From (4.38), (4.39), (4.40) and (4.41), clearly the degree and order conditions for scaled Padé fractions are satisfied. To complete the proof, let

$$G(z) = \text{GCD}\,(T^R_{NN}, R^R_1).$$

Then, from (4.38), $G(z)$ must divide not only T^R_{NN} but $z^{d+1}S^R_{NN}(z)$, as well. Since $\text{GCD}\,(S^R_{NN}, T^R_{NN}) = 1$, it follows that

$$G(z) = z^\lambda$$

for some $\lambda \geqq 0$. Thus, $R^R_1(z)/T^R_{NN}(z)$ is a scaled Padé fraction of type (M, N). Similarly, $R^R_0(z)/T^R_{N^*N^*}(z)$ is a scaled Padé fraction of type (M^*, N^*). □

Observe that

$$M + N = d \tag{4.42}$$

and

$$M^* + N^* = d. \tag{4.43}$$

The scaled Padé fractions $R^R_1(z)/T^R_{NN}(z)$ and $R^R_0(z)/T^R_{N^*N^*}(z)$ both lie along the dth anti-diagonal path of the scaled Padé table $\Gamma(A)$, where

DEFINITION (see, for example, Brent et al. [4]). The dth anti-diagonal path of the scaled Padé table $\Gamma(A)$ for a unit power series $A(z)$ is defined to the set of all scaled Padé fractions of type (i, j), where

$$i + j = d. \tag{4.44}$$

But, by Theorem 4.5, the scaled Padé fraction $R^R_1(z)/T^R_{NN}(z)$ and its predecessor $R^R_0(z)/T^R_{N^*N^*}(z)$, along the dth anti-diagonal path, are obtained by applying Euclid's extended algorithm to the reciprocals of $1 + zA^R_d(z)$ and $B^R(z) = -1$, that is, to the polynomials $A_d(z) + z^{d+1}$ and $-z^{d+1}$. However,

$$\text{GCD}\,(A_d(z) + z^{d+1}, -z^{d+1}) = \text{GCD}\,(A_d(z), -z^{d+1}). \tag{4.45}$$

This is precisely the result given by McEliece et al. [13]; namely, Euclid's extended algorithm applied to $A_d(z)$ and $-z^{d+1}$ yields all the Padé fractions along the dth anti-diagonal path of the Padé table for $A(z)$. Equivalently, by Theorem 4.5, the same Padé fractions can be obtained by applying Algorithm 3 to the two polynomials $1 + zA^R_d(z)$ and $B^R(z) = -1$, and then taking reciprocals of the results.

In addition, by Theorem 4.5, Algorithm 2 can be used to compute any specific scaled Padé fraction $R^R_1(z)/T^R_{NN}(z)$ of type (M, N), where $M + N = d$, along the dth

anti-diagonal path for $A(z)$. This requires $O(N \log^2 N)$ arithmetic operations to compute, by Algorithm 2, the scaled Padé fraction $S_{NN}(z)/T_{NN}(z)$ of type (N, N) for $1+zA_d^R(z)$, plus an additional $O((d-N)\log(d-N))$ operations to determine $R_1(z)$ which satisfies (4.36). This asymptotic cost is the same as the cost of the fast algorithm of Brent et al. [4] for computing the greatest common divisor of $A_d(z)$ and $-z^{d+1}$. However, their PRSDC algorithm can compute a specific Padé fraction along the dth anti-diagonal only with significantly more complications.

5. Conclusions. Central to the classical theory of Padé approximants of power series are the concepts of the Padé form, which always exists but may not be unique, and the Padé fraction, which is unique but in a certain sense may not exist. The fundamental definition introduced in this paper is the scaled Padé fraction which exists uniquely. It is shown that scaled Padé fractions satisfy a three-term relationship between elements along an off-diagonal path of the scaled Padé table. This relationship circumvents the problems of the degenerate case—i.e., the problems which plague other relationships such as those upon which the ε-algorithm and the qd-algorithm are based.

The three-term relationship is used to develop an $O(n^2)$ algorithm (Algorithm 3) which computes along an off-diagonal path, a finite sequence of successive scaled Padé fractions for the quotient of two power series. In the case that the power series is normal, Algorithm 3 is identical to the one described by Brezinski [5]. In the case that computations progress along the diagonal, Algorithm 3 becomes that of Cabay and Kao [6]. Furthermore, it is shown that if the two power series are finite (i.e., polynomials), then Algorithm 3 with computations along one specific off-diagonal path is exactly equivalent to Euclid's extended algorithm for computing the greatest common divisor of two polynomials. However, other off-diagonal paths can be used to compute greatest common divisors, and it remains a subject for future research to determine the optimal one.

By doubling the step size at each iteration, the three-term relationship gives rise to an $O(n \log^2 n)$ algorithm (Algorithm 1 or Algorithm 2). The cost complexity assumes the existence of fast methods for polynomial arithmetic. The decision to double the step-size (rather than to triple it, for example) is not accidental. We believe that this choice is optimal (within the constraints placed by fast polynomial arithmetic methods), but a formal proof remains to be obtained.

When applied to polynomials, Algorithm 2 at each iteration routinely produces intermediate polynomial remainder sequences. For this reason, and because Algorithm 2 is simply a fast version of Algorithm 3, Algorithm 2 is truly a fast Euclid's extended algorithm. The PGCD, HGCD and EMGCD algorithms of Moenck [14], of Aho et al. [1], and of Brent et al. [4], respectively, also compute greatest common divisors with the same cost complexity, but these methods are simply GCD methods. They do not produce intermediate polynomial remainder sequences. In addition, the artificial splitting of polynomials used by these methods to achieve their speed makes them difficult to understand. Finally, these methods are totally recursive, which makes them more costly to implement than Algorithm 2 which is semi-iterative.

For computing scaled Padé fractions for a power series, Brent et al. [4] have shown that the EMGCD algorithm can be modified, with substantial extra detail, to compute entries along an anti-diagonal path of the Padé table. We have shown that Algorithm 2 and Algorithm 3 can also be used to compute such entries, routinely. From a practical point of view, however, it seems to us that computations along an anti-diagonal path are not as natural as they are along an off-diagonal path. If the choice of the anti-diagonal path is incorrect, computations must be restarted. For

off-diagonal computations, n need not be known a priori, and from an application point of view, this may be one of the most important contributions of this paper.

It is not clear at this time whether or not the fast method, Algorithm 2, is useful in a practical environment. Initial experiments performed by Verheijen [18] indicate that the fast method, Algorithm 2, outperforms the classical one, Algorithm 3, only when n is greater than approximately 1600. However, as for the fast methods for polynomial multiplication, a hybrid of Algorithm 2 and Algorithm 3 should significantly lower this cross-over point. This remains a subject for future research.

An effective implementation of an off-diagonal algorithm should prove to be a substantial tool in the design of a symbolic and algebraic system. A single routine serves

(1) to obtain rational approximants of power series,
(2) to convert rational functions from their power series representation,
(3) to compute greatest common divisors of polynomials, and
(4) to solve Hankel and Toeplitz systems of equations.

The scope of this paper includes those power series with coefficients over a field, only. This restriction is relevant whenever division is required by the off-diagonal algorithms. By choosing to perform pseudo-division, rather than division, the algorithms can be modified so that they are applicable to power series over a Euclidean domain, rather than a field. As for Euclid's extended algorithm, the modified algorithms shall experience exponential growth of coefficients. It is a subject for future research to determine if methods similar to those of Collins [9] can be used to keep the growth linear; and if so, would the resulting algorithm compare favorably with the method proposed by Geddes [10]. Other plausible methods for extending the results to Euclidean domains, and which need to be investigated, include the Chinese Remainder and Hensel algorithms (see Yun [20]).

The extension of results to Euclidean domains includes as a special case the multivariate power series. It is of interest to determine how this extension would compare with current methods for solving block Hankel systems, and for constructing Padé approximants for multivariate power series. This remains a subject for future research.

As a final suggestion for future research, the stability of the algorithms for the numerical computation of scaled Padé fractions requires investigation. Some very preliminary results are encouraging.

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] G. A. BAKER, JR, *Essentials of Padé Approximants*, Academic Press, New York, 1975.

[3] J. BENTLEY, D. HANKEN AND J. SAXE, *A general method for solving divide-and-conquer recurrences*, ACM SIGACT NEWS, 3 (1980), pp. 33-44.

[4] R. BRENT, F. G. GUSTAVSON AND D. Y. Y. YUN, *Fast solution of Toeplitz systems of equations and computation of Padé approximants*, J. Algorithms, 1 (1980), pp. 259-295.

[5] C. BREZINSKI, *Computation of Padé approximants and continued fractions*, J. Comp. Appl. Math., 2 (1976), pp. 113-123.

[6] S. CABAY AND T. T. C. KAO, *The diagonal Padé table and the triangular decomposition of Hankel matrices*, Tech. Rept. TR83-10, Dept. Computing Science, Univ. Alberta, Edmonton, 1983.

[7] D. K. CHOI, *Algebraic computations of scaled Padé fractions*, Ph.D. dissertation, Univ. Alberta, Edmonton, 1984.

[8] G. CLAESSENS, *A new look at the Padé table and the different methods for computing its elements*, J. Comp. Appl. Math., 1 (1975), pp. 141-152.

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

[10] K. O. GEDDES, *Symbolic computation of Padé approximants*, ACM Trans. Math. Software, 5 (1979), pp. 218-233.
[11] W. B. GRAGG, *The Padé table and its relation to certain algorithms of numerical analysis*, SIAM Rev., 14 (1972), pp. 1-61.
[12] J. D. LIPSON, *Elements of Algebra and Algebraic Computing*, Addison-Wesley, Reading, MA, 1981.
[13] R. J. MCELIECE AND J. B. SHEARER, *A property of Euclid's algorithm and an application to Padé approximation*, SIAM J. Appl. Math., 34 (1978), pp. 611-617.
[14] R. MOENCK, *Studies in fast algebraic algorithms*, Tech. Rep. TR-57, Computer Science Dept., Univ. Toronto, Toronto, Ontario, 1973.
[15] H. PADÉ, *Sur la représentation approchée d'une fonction par des fractions rationelles*, Thesis, Ann Ecole Nor., 9 (1982), pp. 1-93, supplement.
[16] J. RISSANEN, *Solution of linear equations with Hankel and Toeplitz matrices*, Numer. Math., 22 (1974), pp. 361-366.
[17] W. F. TRENCH, *An algorithm for the inversion of finite Toeplitz matrices*, J. Soc. Ind. Appl. Math., 12 (1964), pp. 515-522.
[18] A. VERHEIJEN, *Evaluation of diagonal Padé algorithms*, M.Sc. thesis, Univ. Alberta, Edmonton, 1983.
[19] P. WYNN, *The rational approximation of function which are formally defined by a power series expansion*, Math. Comp., 14 (1960), pp. 147-186.
[20] D. Y. Y. YUN, *The Hensel lemma in algebraic manipulation*, Univ. Massachusetts, Amherst, Mac TR-138, 1974.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986

© 1986 Society for Industrial and Applied Mathematics

CONSTRUCTING BELTS IN TWO-DIMENSIONAL ARRANGEMENTS WITH APPLICATIONS*

H. EDELSBRUNNER† AND E. WELZL†

Abstract. For H a set of lines in the Euclidean plane, $A(H)$ denotes the induced dissection, called the arrangement of H. We define the notion of a belt in $A(H)$, which is bounded by a subset of the edges in $A(H)$, and describe two algorithms for constructing belts. All this is motivated by applications to a host of seemingly unrelated problems including a type of range search and finding the minimum area triangle with the vertices taken from some finite set of points.

Key words. computational geometry, arrangements of lines, plane-sweep, maintenance of convex hulls, efficient algorithms, halfplanar range search

1. Introduction. The systematic study of algorithms for low-dimensional geometric problems started with the doctoral dissertation of Shamos [S] around 1975. Since then, this area of research experienced a steady increase in activity which can be explained by the host of theoretically interesting problems in the field and also by the relevance of the developed results to more practically oriented branches in computer science.

This paper is concerned with several seemingly unrelated problems dealing with finite sets of points in the Euclidean plane. The problems are discussed in §§ 4.1 through 4.5. All solutions rely heavily on the more basic developments of §§ 2 and 3. Section 2 introduces the concept of a dual plane which hosts a line for each point in the original space. Such a set of lines cuts the (dual) plane into convex regions, edges, and vertices which express certain convexity properties of the (original) point-set fairly explicitly. The set of lines also turns out to favour the development of algorithmic solutions for the (original) problems, once the correspondences between both settings are reasonably understood. Section 3 presents two algorithms for constructing belts, a fundamental concept defined for sets of lines in § 2.

We explicate one problem to illustrate the general character of our results: Let S be a set of n points in the plane. The *halfplanar range search problem* requires a data structure for S such that the number of points which lie in a later specified query halfplane can be determined efficiently. All known solutions either take a lot of space ($O(n^2)$ space suffices to achieve $O(\log n)$ time for answering a query [EKM]) or suboptimal time (with $O(n)$ space, the currently best solutions answers a query in $O(n^{0.695})$ time [EW2]). Recent arguments of Fredman [F] also support the thesis that halfplanar range search is inherently more complex than classical orthogonal range search (see Bentley and Friedman [BF] for an early survey of solutions for the latter). Motivated by these results, we relax the problem and ask for rough ideas of the number of points in a query halfplane rather than for the exact answer. Sections 4.1 and 4.2 offer families of solutions which realize varying degrees of accuracy. Most striking, Theorem 4.6 shows the existence of a constant space structure which discriminates between halfplanes that contain less than one third and more than two thirds of all points.

2. Geometric preliminaries. This section introduces the geometric concepts and facts which are needed in the forthcoming discussions. The primary concern are so-called k-sets of planar point-sets and their appearance under a dualizing geometric transform.

* Received by the editors October 4, 1982, and in revised form October 29, 1984.
† Institutes for Information Processing, Technical University of Graz, A-8010 Graz, Austria.

Let S denote a set of n points in the plane, for some positive integer n. We call a subset S' of S a *k-set of S*, for $0 \leqq k \leqq n$, if it contains k points and there exists a halfplane which intersects S in S'. Let $f_k(S)$ denote the number of k-sets realized by S and let $f_k(n)$ denote the maximum of $f_k(S)$, for all sets S of n points in the plane. Then $f_0(n) = f_n(n) = 1$ and obviously $f_k(n) = f_{n-k}(n)$, for $0 \leqq k \leqq n$. Bounds on the asymptotic behaviour of $f_k(n)$, for $1 \leqq k \leqq n-1$, are developed in Erdös, Lovasz, Simmons, and Straus [ELSS] and in Edelsbrunner and Welzl [EW1]:

PROPOSITION 2.1. *Let k and n denote two positive integers with $k \leqq \lfloor n/2 \rfloor$. Then $f_k(n)$ and $f_{n-k}(n)$ are in $\Omega(n \log(k+1))$ and in $O(nk^{1/2})$.*

It is often the case that geometric problems formulated for point-sets are more conveniently solved in dual space, which can, e.g., be obtained by application of the following geometric transform T which was also used in Brown [Br] to solve geometric problems different from ours.

(1) A point $p = (p_x, p_y)$ is mapped into the line $T(p)$ whose points (x, y) satisfy $y = p_x x + p_y$, and

(2) a nonvertical line L whose points (x, y) satisfy $y = L_x x + L_y$ is mapped into the point $T(L) = (-L_x, L_y)$ (see Fig. 2-1).

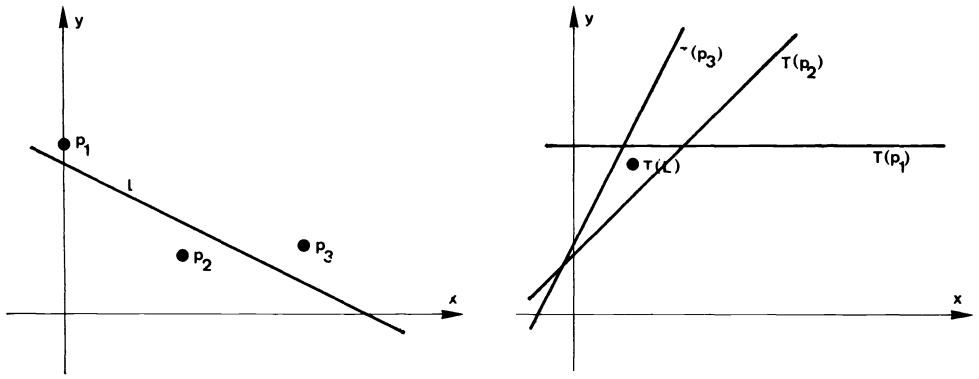


FIG. 2.1. *T applied to three points and a line.*

A set S of points is transformed into a set $T(S)$ of nonvertical lines. The nice property of T is the maintenance of the relative position between a point and a line. Let $p = (p_x, p_y)$ be a point and L: $y = L_x x + L_y$ a nonvertical line. Let $y_0 = L_x p_x + L_y$. Then we say that p *lies below*, *on*, or *above* L depending on whether $p_y < y_0$, $p_y = y_0$, or $p_y > y_0$. We also say that L *lies above* p, *contains* p, or *lies below* p in these cases.

Observation 2.2. Let p denote a point and L a nonvertical line in the plane. Then p lies below (or above) L if and only if $T(p)$ lies below (or above) $T(L)$.

This effect can be observed in Fig. 2.1 where p_2 is the only point below L and $T(p_2)$ is the only line below $T(L)$.

The set $H = T(S)$ of lines induces a dissection of the plane called the *arrangement $A(H)$ of H*. $A(H)$ consists of *vertices* (intersections of lines), *edges* (maximal connected subsets of the lines which contain no vertex), and *regions* (maximal connected subsets of the plane which contain no edge or vertex). For convenience, we say that two parallel lines intersect in a vertex at infinity. Also for convenience, we define the notions of complete edges and regions: A *complete edge of $A(H)$* is a bounded edge, an unbounded edge on a line that has a parallel line in H, or the union of two unbounded edges both supported by the same line which has no parallel line in H. A *complete*

region of $A(H)$ is a bounded region, an unbounded region enclosed between two parallel lines, or the union of two unbounded regions R_1 and R_2 such that any line in H separates R_1 from R_2. Notice the similarity of these notions to the concepts of edges and regions in the projective plane. There exists an important correspondence between the k-sets of S and the complete regions of $A(H)$:

Observation 2.3. Let S' be some k-set of S. Then there exists a complete region R in $A(H)$ such that a line L separates S' from $S-S'$ if and only if $T(L)$ lies in R.

For each point p in the plane let $b(p)$, $o(p)$, and $a(p)$ denote the number of lines in H which lie below p, contain p, and lie above p, respectively. Evidently, $b(p)+o(p)+a(p)=n$ (n the cardinality of H), for each point p. We define the *k-belt of H*, for $0 \leqq k \leqq \lceil n/2 \rceil$, as the set of points p in the plane such that $b(p)+o(p) \geqq k$ and $a(p)+o(p) \geqq k$ (see Fig. 2.2). The 0-belt is the whole plane, and the k-belt, for $k \geqq 1$, is bounded below and above by an unbounded polygonal chain, respectively. These polygonal chains are monotone in x, that is, any one of them intersects each vertical line in exactly one point. For $k=(n+1)/2$, the two boundaries of the k-belt coincide. Obviously, the k'-belt is contained in the k''-belt if $0 \leqq k'' < k' \leqq \lceil n/2 \rceil$.

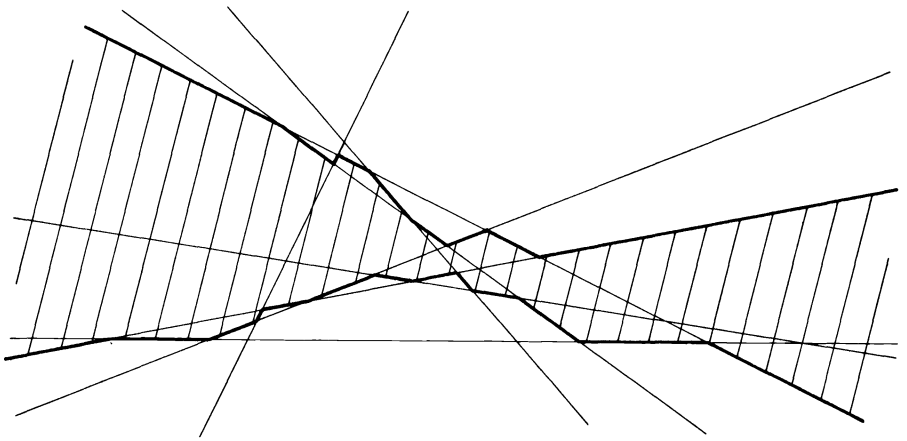


FIG. 2.2. *The 3-belt for 8 lines.*

Let $p=(p_x, p_y)$ be an arbitrary point in the plane and let B be the k-belt of H, for some integer k. Let y_b and y_a denote the y-coordinates of the intersections of the vertical line through p and the lower and upper boundary of B, respectively. We say that p lies *below*, *in*, or *above* B if $p_y < y_b$, $y_b \leqq p_y \leqq y_a$, or $y_a < p_y$ is true, respectively.

In the following sections, a k-belt will be represented by the sequences of edges of its boundaries. Let $b_k(H)$ denote the number of complete edges bounding the k-belt of H, and let $b_k(n)$ denote the maximum of $b_k(H)$, for all sets H of n lines in the plane. The strong relationship between the k-sets of S and the complete regions of $A(H)$ imply that $b_k(n)$ and $b_{n-k}(n)$ are in $\Theta(f_{k-1}(n)+f_k(n))$ and therefore in $\Omega(n \log (k+1))$ and in $O(nk^{1/2})$. The interested reader is invited to verify the following:

LEMMA 2.4. *Let S denote a set of n points in the plane with no three collinear, and define $H=T(S)$. Then*

$$b_0(H)=0,$$

$$b_1(H)=f_1(S),$$

$$b_k(H)=f_{k-1}(S)+f_k(S), \quad \text{for } 2 \leqq k \leqq \lceil (n-1)/2 \rceil.$$

In § 4 of this paper, so-called simplified belts of sets of lines are used to decrease the space requirements needed for solving halfplanar range estimation problems. These simplified belts are obtained from ordinary belts by replacing sequences of complete edges by single complete edges. To this end, let B denote the k-belt of a set H of $n \geqq 2$ lines and $1 \leqq k \leqq \lceil n/2 \rceil$. For convenience, we assume that no three lines of H are concurrent, that is, no three intersect in a common vertex. The unrestricted case will be discussed later. We call a vertex of the lower and upper boundary of B a *lower* and an *upper vertex*, respectively. Note that two parallel lines define a vertex at infinity which is defined a lower vertex. If a vertex belongs to both boundaries then it is considered as two vertices, one of each kind. We assign the numbers $0, 1, \cdots, b_k(H)-1$ to the $b_k(H)$ vertices of B such that $i<j$ if i is a lower and j an upper vertex, and vertex i is to left of the vertical line through vertex j, otherwise (see Fig. 2.3).

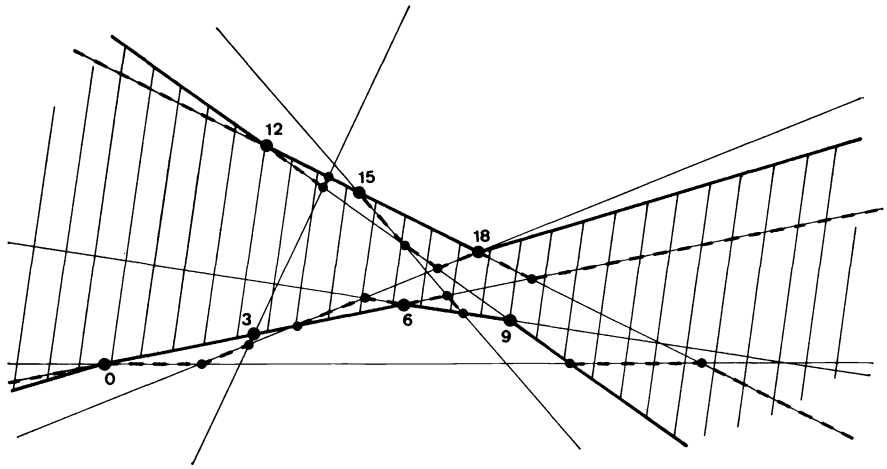


FIG. 2.3. *3-simplified 3-belt for 8 lines.*

Let m be a positive integer with $m \leqq 2\lceil n/2 \rceil - 2k + 1$ and $m \leqq b_k(H) - 1$. The former restriction on m guarantees that the m-simplified belt is going to be bounded by two noncrossing polygonal chains (see Lemma 2.6). The latter restriction implies that the m-simplified belt will avoid over-simplification. The *complete edges* of the boundaries of the m-simplified k-belt connect the vertices 0 and m, m and $2m, \cdots, rm$ and 0, with $r = \lfloor (b_k(H)-1)/m \rfloor$. A complete edge which connects two lower or two upper vertices is the line segment which has the two vertices as endpoints. A complete edge connecting a lower and an upper vertex consists of two rays on the line through the two vertices which emanate from the two vertices such that they do not overlap. The *m-simplified k-belt* is the set of points in between the lower and upper boundary including the boundary points. The reason for the introduction of simplified belts is the smaller number of vertices and edges they consist of as compared to ordinary belts.

Observation 2.5. Let S denote a set of n points in the plane such that no three are collinear. Then the m-simplified k-belt of $H = T(S)$, for $1 \leqq k \leqq \lceil n/2 \rceil$, $1 \leqq m \leqq 2\lceil n/2 \rceil - 2k + 1$, and $m \leqq b_k(H) - 1$, has at most $\lceil b_k(H)/m \rceil$ vertices.

The nice property of simplified k-belts is the fact that they are reasonable approximations of ordinary k-belts.

LEMMA 2.6. *Let S be a set of n points in the plane such that no three are collinear, and define $H = T(S)$. Then the m-simplified k-belt, for $1 \leqq m \leqq 2\lceil n/2 \rceil - 2k + 1$ and $m \leqq b_k(H) - 1$, contains the $(k + \lfloor m/2 \rfloor)$-belt and is contained in the $\max\{0, k - \lfloor m/2 \rfloor\}$-belt of H.*

Proof. Consider an arbitrary complete edge e of the m-simplified k-belt which we call M; see Fig. 2.4. Let L denote a line in H which properly intersects e, that is, it does not support e. It is readily seen that at least one complete edge of the k-belt of H which is supported by L is in the sequence of complete edges replaced by e (see Fig. 2-4). (Note that this is true only because no three lines intersect in a common point.) Consequently, at most $m-2$ lines of H properly intersect e.

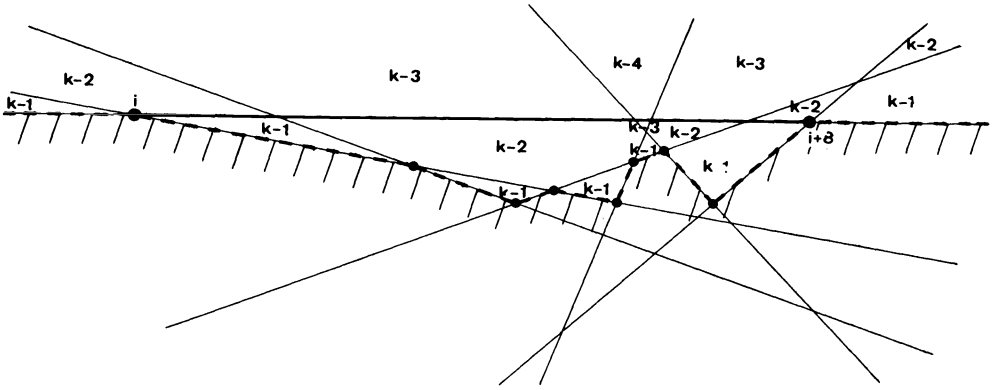


FIG. 2.4. *Simplification with* $m = 8$.

From the correspondence between the k-sets of S and the complete regions of $A(H)$ (see Observations 2.2 and 2.3) and the definition of k-belts we know that the regions which share an edge with the k-belt correspond to $(k-1)$-sets or k-sets. W.l.o.g. let the regions above the considered sequence of edges correspond to $(k-1)$-sets. Those regions above the latter regions which share edges with them correspond to $(k-2)$-sets, etc., etc. Let k' be minimal such that there is a complete region R which corresponds to a k'-set and R intersects e or lies between e and the sequence of edges replaced by e. Since e properly intersects only $m-2$ lines, we have $k' \geqq k - \lfloor m/2 \rfloor$. (The details of the verification of this fact are left to the interested reader.) This implies the second part of Lemma 2.6 while the analogous reasoning implies the first part. This completes the argument.

As an immediate consequence of Lemma 2.6, the m-simplified k-belt intersects each vertical line in a single and nonempty interval provided $k + \lfloor m/2 \rfloor \leqq \lceil n/2 \rceil$ which is equivalent to $m \leqq 2\lceil n/2 \rceil - 2k + 1$. Thus, we can extend the below-in-above relation defined for points and ordinary belts to points and simplified belts without ambiguity.

Let us now return to the general case of a point-set S which may contain an arbitrary number of points on a single line. Difficulties arise when a belt of $H = T(S)$ is simplified which contains a vertex common to more than two lines. If the simplification is carried out as described then Lemma 2.6 need not be true. This can be remedied by taking care that a new complete edge properly intersects at most $m-2$ lines of H. A simple way to achieve this is to consider a vertex v which is common to $i \geqq 2$ lines as a sequence of $\min\{m-1, i-2\}$ degenerate edges and one more vertex. This invalidates the definition given for simplified belts, and Lemma 2.6 is true again. Observation 2.5 is no longer true in the strength it is stated. Nevertheless we have:

Observation 2.7. Let S denote an arbitrary set of n points in the plane. Then the m-simplified k-belt of $H = T(S)$, with $1 \leqq k \leqq \lceil n/2 \rceil$ and $1 \leqq m \leqq 2\lceil n/2 \rceil - 2k + 1$, contains at most $\lceil b_k(n)/m \rceil$ complete edges.

Although there may be new complete edges which replace less than m old ones, the number is compensated by the loss of edges caused by the concurrency of lines.

3. Constructing belts. Let S be a set of n points in the plane. Throughout this section, we will assume that n and k are positive integers such that $k \leqq \lceil n/2 \rceil$. We will introduce two algorithms for constructing a belt of $H = T(S)$ which is represented by two linear lists storing the lower and upper vertices ordered from left to right, respectively. While the first algorithm is reasonably simple, the second is reasonably efficient. In fact, no better algorithm is currently known except for the special case $k = 1$. For the time being, we assume that no two lines of H are parallel and that no three lines are concurrent. The unrestricted case will be discussed later. W.l.o.g. only the construction of the lower boundary of a belt is described.

ALGORITHM TRIVIAL PLANE SWEEP:

Let V denote a vertical line sweeping from left to right. (Note that no line in H is vertical, thus, each line intersects V in exactly one point.) V is associated with a linear array D which maintains the sorted order of the lines w.r.t. their intersections with V as V sweeps from left to right. Hence, at any moment, the kth line in D is also the kth bottommost line at the current x-coordinate. With V we associate also a priority queue Q which contains the x-coordinates of the anticipated intersections of any two adjacent lines in D.

If V encounters the intersection of two lines L_b and L_a, with L_b below L_a to the left of the intersection, then the following actions are taken: (1) Delete the topmost element from Q, that is, the x-coordinate of the intersection. (2) Let L_{bb} and L_{aa} denote the lines immediately below L_b and above L_a, respectively. If L_b and L_{bb}, or L_a and L_{aa} intersect to the right of V then delete the x-coordinate of the respective intersection from Q. (3) Reverse the order of L_b and L_a in D. (4) If L_b and L_{aa}, or L_a and L_{bb} intersect to the right of V, then insert the x-coordinates of those intersections into Q. (5) If L_b or L_a is the kth line before the intersection is encountered, then the intersection point completes an edge of the k-belt which is reflected in the structure of the k-belt.

Trivial modifications enable the algorithm to handle degeneracies like parallel and concurrent lines. In the former case, intersection points in infinity are created. In the latter case, the order of all lines which meet in a vertex encountered by V is reversed in D.

LEMMA 3.1. *Let H be a set of n nonvertical lines in the plane. Algorithm* TRIVIAL PLANE SWEEP *constructs the k-belt of H in $O(n^2 \log n)$ time and $O(n + b_k(H))$ space.*

Proof. There are at most $\binom{n}{2}$ intersections to be handled with the data structures D and Q. Each intersection of two lines requires $O(\log n)$ time, see Aho, Hopcroft, and Ullman [AHU]. Common intersections of $i > 2$ lines require $O(i \log n)$ time which is less than the amount that would be needed if all pairs of the i lines intersect in unique points. (Deletions from Q can be handled by maintaining a pointer from each pair of adjacent lines in D for which an intersection is anticipated to the x-coordinate of this intersection in Q.) The space required by D and Q is $O(n)$ and the one for the k-belt is $O(b_k(H))$. This completes the argument.

It is worth noting that Algorithm TRIVIAL PLANE SWEEP can construct all belts of H within the same asymptotic time bounds. The algorithm is also an interesting general method for certain types of point problems if it neglects the construction of belts and performs other actions instead. In fact, § 4.5 demonstrates one such example.

For computing a single belt, a considerably more efficient method is obtained by refining Algorithm TRIVIAL PLANE SWEEP: Instead of considering all $\binom{n}{2}$ intersections, only those coinciding with vertices of the belt are processed. This is made possible by maintaining the lines below and above the kth line in separate data structures of

a particular kind. For this algorithm we need the following result due to Overmars and van Leeuwen [OvL1]:

PROPOSITION 3.2. *The intersection of a set S of n halfplanes can be computed in $O(n \log n)$ time and $O(n)$ space. This intersection can then be maintained with $O(\log^2 n)$ time per insertion and deletion such that the adjacent edges of a given edge are available in constant time. All bounds state the requirements in the worst case and n denotes the current number of halfplanes stored.*

Again only the construction of the lower boundary of the belt is described, and for the time being, we assume that no two lines are parallel and no three are concurrent.

ALGORITHM SOPHISTICATED PLANE SWEEP:

Let V denote a vertical line sweeping from left to right. At some moment in time let L in H cause the kth bottommost intersection V_L with V. The lines below V_L are interpreted as lower boundaries of halfplanes which are stored in a data structure H_B. The lines above V_L are interpreted as upper boundaries of halfplanes which are stored in a similar data structure H_A. H_B and H_A are instances of the data structure referred to in Proposition 3.2.

The next intersection of L to the right of V is determined as follows: (1) The halfplane bounded below by L is inserted into H_B. The adjacent edge in counterclockwise order of the edge caused by this halfplane (if it exists) gives the leftmost intersection to the right of V of L with a line L_B below V_L (see Fig. 3.1, where no such edge exists). (2) The halfplane bounded above by L is inserted into H_A. The adjacent edge in clockwise order of the edge caused by this halfplane (if it exists) gives the leftmost intersection to the right of V of L with a line L_A above V_L (see Fig. 3.1). (3) W.l.o.g. let L_A intersect L above L_B (as illustrated in Fig. 3.1). Then L is deleted from H_B and L_A is deleted from H_A. (If neither L_A nor L_B exists, then L supports the rightmost and unbounded edge of the lower boundary.) In addition, the new edge of the k-belt created is reflected in the data structure of the k-belt, and L_A is the new kth line.

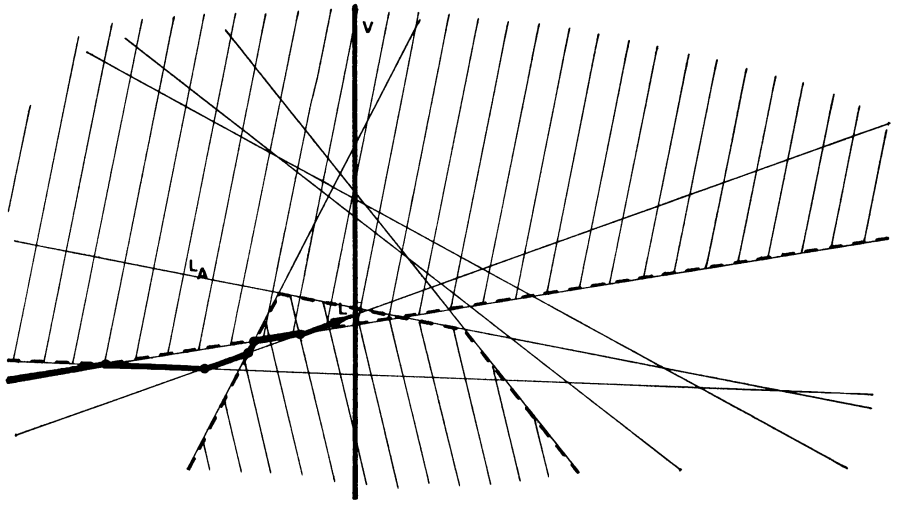


FIG. 3.1. *Constructing the 3-belt of 8 lines.*

A short moment of reflection shows that for the correctness of the algorithm it must be guaranteed that L contributes an edge to the intersection of halfplanes in H_B and H_A, respectively. In addition, this edge must intersect V at its current position. We will argue that this is true for H_B as the case of H_A is completely analogous.

Let I_B denote the intersection of the halfplanes in H_B. Since the bounding lines of those halfplanes are all below L at the current position of V, L has at least the intersection with V in common with I_B. This point lies also on the boundary of the halfplane which is bounded below by L. This implies what has to be guaranteed.

Similarly to Algorithm TRIVIAL PLANE SWEEP, degeneracies like parallel and concurrent lines can be incorporated easily. While the former case is completely trivial, the latter requires some care. It occurs when L intersects I_B or I_A in a vertex or both in the same point. If L intersects i lines in a single point, then up to $i+3$ insertions and deletions have to be performed.

LEMMA 3.3. *Algorithm* SOPHISTICATED PLANE SWEEP *constructs the k-belt of H in* $O(b_k(n)\log^2 n)$ *time and* $O(n+b_k(H))$ *space.*

Proof. At the far left, the line with the kth smallest slope is also the kth bottommost line. This line L can be determined in $O(n)$ time and the data structures H_B and H_A for the lines below and above L, respectively, can be constructed in $O(n \log n)$ time. These activities require $O(n)$ space.

Each intersection point of two lines on the boundaries of the k-belt can be determined in $O(\log^2 n)$ time and the structures H_B and H_A can then be adjusted in $O(\log^2 n)$ time, see Proposition 3.2. A careful implementation processes a vertex common to $i>1$ lines in $O(i \log^2 n)$ time. Note that this is less than the amount which is required in the worst case if those i lines define $\binom{i}{2}$ intersection points. This implies that $O(b_k(n)\log^2 n)$ time is spent for the computation and $O(b_k(H))$ space is occupied by the description of the k-belt. This completes the argument.

Both algorithms can easily be adapted to construct simplified belts instead of ordinary ones. For computing the m-simplified k-belt, only each mth vertex of the k-belt gives rise to a new edge to be created. Recall that a vertex common to $i \geqq 2$ lines is interpreted as $\min\{m, i-1\}$ vertices. This implies:

THEOREM 3.4. *Let S denote a set of n points in the plane. Then there exists an algorithm which constructs the m-simplified k-belt (the ordinary k-belt is the 1-simplified k-belt) of* $H = T(S)$ *in* $O(b_k(n)\log^2 n)$ *time and* $O(b_k(n)/m)$ *space.*

The space bound is actually $O(n+b_k(H))$ for ordinary k-belts. The proof is an immediate consequence of Lemma 3.3 and Observations 2.5 and 2.7.

This result is not optimal at least for $k=1$ which is shown by the following argument: Let L^+ denote the open halfplane which is bounded below by the nonvertical line L, and let L^- denote the open halfplane which is bounded above by L. Then the 1-belt B of H can be written as

$$B = E^2 - \bigcap_{L\in H} L^+ - \bigcap_{L\in H} L^-.$$

Since the intersection of n halfplanes can be determined in $O(n \log n)$ time, see Proposition 3.2, this gives also a method which computes B in $O(n \log n)$ time.

4. Applications. The methods presented in § 3 have interesting applications to seemingly unrelated problems. Algorithm TRIVIAL PLANE SWEEP yields a new method for finding a minimum area triangle with the three vertices taken from some given finite set. This is discussed in § 4.5. Applications of k-belts to halfplanar range estimation, finding so-called centerpoints, k-nearest neighbour search, and a problem with points moving on a line are discussed in §§ 4.1 through 4.4.

4.1. Halfplanar range estimation. This section describes the use of ordinary and simplified belts for halfplanar range estimation. Representative for a variety of conceivable variants, we define three of the most basic halfplanar range estimation problems and show how to solve them.

Let S denote a set of n points in the plane and let k be some positive integer, with $k \leqq \lceil n/2 \rceil$. The simplest halfplanar range estimation problem, termed *simple* $EP(k)$, reads as follows: Let h denote a query halfplane which contains $A(h)$ points of S. Decide whether $0 \leqq A(h) < k$, $k \leqq A(h) \leqq n-k$, or $n-k < A(h) \leqq n$.

THEOREM 4.1. *Let S be a set of n points in the plane. There exists a data structure which requires $O(b_k(n))$ space and $O(b_k(n) \log^2 n)$ time for construction such that $O(\log n)$ time suffices to answer a query of the simple $EP(k)$, with $k \leqq \lceil n/2 \rceil$.*

Proof. The asserted bounds can be achieved with the k-belt of $H = T(S)$ as will be shown. The required amount of space and time for the construction follows immediately from Theorem 3.4. Thus, let us consider a query of the simple $EP(k)$. W.l.o.g. let the query halfplane h be bounded above by the line L. Then

(1) h contains less than k points if and only if $T(L)$ lies below the k-belt of H,

(2) h contains no less than k and no more than $n-k$ points if and only if $T(L)$ is contained in the k-belt, and

(3) h contains more than $n-k$ points if and only if $T(L)$ lies above the k-belt of H.

Which one of the three cases applies can be decided in $O(\log n)$ time by binary search since the two boundaries of the k-belt are monotone in x. This completes the argument.

Considerably more accurate answers can be obtained using several belts instead of a single one. We define the *uniform* $EP(\varepsilon)$, with $0 < \varepsilon < 1$, as follows: Let $j = \lfloor \lceil n/2 \rceil / \lfloor n^\varepsilon \rfloor \rfloor$ and let $A(h)$ denote the number of points of S in the query halfplane h. Determine an integer i, with $-j \leqq i \leqq j$, such that $(j-i)\lfloor n^\varepsilon \rfloor \leqq A(h) < (j-i+1)\lfloor n^\varepsilon \rfloor$ if $i < 0$, $j\lfloor n^\varepsilon \rfloor \leqq A(h) \leqq n - j\lfloor n^\varepsilon \rfloor$ if $i = 0$, and $n-(j-i+1)\lfloor n^\varepsilon \rfloor < A(h) \leqq n-(j-i)\lfloor n^\varepsilon \rfloor$ if $i > 0$. This somewhat ugly definition of the problem has been chosen since the most simple combination of belts is able to solve it. Similar but differently defined halfplanar range estimation problems can be dealt with in more complicated but essentially equivalent combinations of belts. For convenience, we define $B_K(H) = \sum_{k \in K} b_k(H)$ for $H = T(S)$ and K subset of $\{1, 2, \cdots, \lceil n/2 \rceil\}$. $B_K(n)$ is then defined as the maximum of $B_K(H)$ for all sets S of n points.

THEOREM 4.2. *Let S be a set of n points in the plane and write H for $T(S)$. For the uniform $EP(\varepsilon)$, with $0 < \varepsilon < 1$, there exists a data structure which requires $O(B_K(n))$ space and $O(B_K(n) \log^2 n)$ time for construction, where $K = \{\lfloor n^\varepsilon \rfloor, 2\lfloor n^\varepsilon \rfloor, \cdots, j\lfloor n^\varepsilon \rfloor\}$, such that a query can be answered in $O(\log n)$ time.*

Before proceeding to the proof of this assertion let us comment on $B_K(n)$. No reasonable upper bounds are known for these sums. The best bound follows from Proposition 2.1 and claims that $B_K(n)$ is in $O(n^2)$ and in $O(|K| n^{3/2})$, with $|K|$ the number of indices in K.

Proof of Theorem 4.2. The data structure used for the uniform $EP(\varepsilon)$ consists of the k-belts of H, for k in K. Let h denote a query halfplane which is, say, bounded above by the line L. In order to answer the query, we determine the largest i' in $\{0, 1, \cdots, j\}$ such that $T(L)$ is contained in the $i\lfloor n^\varepsilon \rfloor$-belt of H. Then the answer is $-j+i'$ if $i' < j$ and $T(L)$ lies below the $(i'+1)\lfloor n^\varepsilon \rfloor$-belt, and it is $j-i'$, otherwise.

Notice that binary search testing $T(L)$ against the boundaries of the various belts is no longer appropriate in order to achieve $O(\log n)$ query time. In the place of that strategy we exploit a method due to Kirkpatrick [K]. Let D be a partition of the plane with a total of $|D|$ regions, straight edges, and vertices. Kirkpatrick's method permits us to determine in $O(\log |D|)$ time the region of D that contains a query point. $O(|D|)$ space and $O(|D| \log |D|)$ time for construction is required. The assertion follows since the collection of k-belts used induces a partition D of the plane with $|D| = O(B_K(n))$. This completes the argument.

Before considering simplified belts for halfplanar range estimation we note that k-belts of a line arrangement can be combined almost arbitrarily leading to all sorts of answers reflecting the approximate number of points in the respective query halfplane.

A serious drawback of ordinary belts is the superlinear space required in the worst case. In particular, the combination of a large number of belts needs a rather large amount of space. This drawback is bypassed by exploiting simplified belts as introduced in § 2.

Let S denote a set of n points in the plane, define $H = T(S)$, let k denote an integer with $1 \leqq k \leqq \lceil n/2 \rceil$, and let m denote an integer with $2 \leqq m \leqq 2\lceil n/2 \rceil - 2k + 1$ and $m \leqq b_k(H) - 1$. Note that the simple $EP(k)$ cannot be solved with the m-simplified k-belt of H, if $m > 1$. We define the *simple* $EP(k, m)$ as follows: Let h be a query halfplane and let $A(h)$ designate the number of points of S which are contained in h. Decide whether $0 \leqq A(h) < k + \lfloor m/2 \rfloor$, $k - \lfloor m/2 \rfloor \leqq A(h) \leqq n - k + \lfloor m/2 \rfloor$, or $n - k - \lfloor m/2 \rfloor < A(h) \leqq n$. Note that the three cases are not exclusive which implies that $A(h)$ does not uniquely determine the answer.

THEOREM 4.3. *Let S be a set of n points in the plane and $H = T(S)$. For the simple $EP(k, m)$ with $1 \leqq k \leqq \lceil n/2 \rceil$, $2 \leqq m \leqq 2\lceil n/2 \rceil - 2k + 1$, and $m \leqq b_k(H) - 1$, there exists a data structure which requires $O(b_k(n)/m)$ space and $O(b_k(n) \log^2 n)$ time for construction such that $O(\log n)$ time suffices to answer a query.*

Proof. The bounds for the required space and time for construction are trivially achieved by the m-simplified k-belt of H which we call M, see Observation 2.7 and Theorem 3.4. By Lemma 2.6, M is contained in the $\max\{0, k - \lfloor m/2 \rfloor\}$-belt of H and contains the $(k + \lfloor m/2 \rfloor)$-belt of H. Since $k - \lfloor m/2 \rfloor \geqq 0$ and $k + \lfloor m/2 \rfloor \leqq \lceil n/2 \rceil$ by definition of m, a query with halfplane h bounded above by L can be answered by deciding whether $T(L)$ lies below, in, or above M. This can be done in $O(\log n)$ time using binary search which completes the argument.

We say that a halfplanar range estimation problem has *accuracy* m if the overlap between two different answers is at most m. E.g. the problem solved by the $\lceil n^{1/2} \rceil$-simplified $\lfloor n/3 \rfloor$-belt M of H has accuracy $2\lfloor n^{1/2}/2 \rfloor$. That is, M decides with accuracy $2\lfloor n^{1/2}/2 \rfloor$ whether there are less than $\lfloor n/3 \rfloor$, from $\lfloor n/3 \rfloor$ to $n - \lfloor n/3 \rfloor$, or more than $n - \lfloor n/3 \rfloor$ of the points in the query halfplane. Due to Proposition 2.1 and Observation 2.7, M is known to require $O(n)$ space.

As for ordinary belts, almost arbitrary combinations of simplified belts can be used to obtain different solutions for different halfplanar range estimation problems. For example, a relaxed version of the uniform $EP(\varepsilon)$ can be solved with $\lceil n^{\varepsilon} \rceil$-simplified belts yielding:

THEOREM 4.4. *Let S be a set of n points in the plane and let ε be a real number, with $0 < \varepsilon < 1$. Then there exists a data structure which requires $O(n^{1-2\varepsilon} b_{n/2}(n))$ space and $O(n^{1-\varepsilon} b_{n/2}(n) \log^2 n)$ time for construction such that $O(\log n)$ time suffices to answer the uniform $EP(\varepsilon)$ with accuracy $\lfloor n^{\varepsilon} \rfloor$.*

4.2. Centerpoints. Let S be a set of n points in the plane. A point c not necessarily in S is called a *centerpoint of* S if the two closed halfplanes of any line through c contain both at least $\lceil n/3 \rceil$ points. The existence of a centerpoint for every set S is a consequence of Helly's theorem [H]. In dual space, $T(c)$ is a line such that for any point p of $T(c)$ there are respective at least $\lceil n/3 \rceil$ lines of $H = T(S)$ which are not below and not above p. So, $T(c)$ is contained in the $\lceil n/3 \rceil$-belt of H. The construction of the region of all centerpoints of S thus might proceed as follows:

1. Construct the $\lceil n/3 \rceil$-belt B of H.

2. Compute the intersection C of all closed halfplanes h defined as follows: Let line L bound h, then $T(L)$ is a vertex of B, and there are at most $\lceil n/3 \rceil - 2$ points outside of h.

Theorem 3.4 and a method for constructing the intersection of m halfplanes in $O(m \log m)$ time (Brown [Br]) imply

THEOREM 4.5. *$O(b_{\lceil n/3 \rceil}(n) \log^2 n)$ time and $O(b_{\lceil n/3 \rceil}(n))$ space suffices to construct the region of all centerpoints of a set of n points in the plane.*

A line $T(c)$ dual to a centerpoint c of S can be used for a surprisingly space efficient solution of a weak halfplanar range estimation problem.

THEOREM 4.6. *Let S be a set of n points in the plane. There is a data structure which takes constant space and $O(b_{\lceil n/3 \rceil}(n) \log^2 n)$ time for its construction such that constant time suffices to decide whether a given halfplane contains at most $n - \lceil n/3 \rceil$ or at least $\lceil n/3 \rceil$ points of S.*

We mention that developments following the ones reported in this paper lead to new algorithms which compute a single centerpoint in $O(n \log^4 n)$ time [CSY], [C].

4.3. *k*-nearest neighbours search. Let S denote a set of n points in the plane and let k be some integer with $1 \leqq k \leqq n-1$. The *k-nearest neighbours search problem* [SH], [L] requires the accommodation of S such that the k nearest to a later specified query point can be determined efficiently. We consider the problem restricted to query points in infinity which are better interpreted as directions. We assume that k is small compared to n which are the only interesting cases.

Let B denote the k-belt of $H = T(S)$. To each edge e of the lower boundary of B we assign the list of lines below including the line which supports e. Similarly, to each edge e of the upper boundary we assign the list of lines above including the line which supports e. Note that two unbounded edges whose union is a complete edge have assigned the same list which is now assigned to the complete edge. By construction, two adjacent complete edges can have the same list of lines. In such a case both complete edges are replaced by a single complete edge as described for simplified belts in § 2. The new complete edge has assigned the same list as the two complete edges it replaces. Let B' denote the modified belt which is obtained by repeatedly replacing adjacent complete edges with identical lists of lines. (Note that Algorithm SOPHISTICATED PLANE SWEEP can be used to construct B' without creating B.)

A direction in the original space transforms under T into a vertical line. The query is answered by binary search to identify the two complete edges of B' which intersect this vertical line. The k-nearest neighbours are those points whose corresponding lines are assigned to one of the two complete edges determined.

Some amount of space can be saved, in particular for large k, via the following method: Instead of assigning a list to each complete edge of B', we assign a list only to every kth of its complete edges. Let e denote a complete edge which has assigned a list of lines. The complete edge f immediately to the right of e stores the lines that must be deleted from, resp. inserted into, the list of e in order to obtain the one of f. In nondegenerate cases, f stores two lines. To cope with cases where f stores $2i > 2$ lines, f is effectively treated like a sequence of i complete edges. The analogous information is stored for each of the next $k-2$ complete edges to the right of f. The space required in the worst case is reduced by a factor k while the query time remains the same by the following strategy: For a vertical query line q, the lower (or upper) complete edge of B' which intersects q is determined. Then the first complete edge to the left of the latter is identified which has assigned a full list of lines. This list is updated during the walk back to the originally determined complete edge. The lists

are stored as doubly linked lists and the updates to be performed are indicated by pointers to what has to be changed. A careful implementation of this method finally implies:

THEOREM 4.7. *Let S denote a set of n points in the plane. There exists a data structure which requires $O(f_k(S))$ space and $O(b_k(n)\log^2 n)$ time for construction such that $O(\log n + k)$ time suffices to report the k first points of S in a query direction.*

4.4. Moving points on a line. Let S be a set of n moving points on a vertical line. Each point p of S is specified by its location p_0 at time 0 and by its constant speed p_s which is positive if p moves upwards and negative if p moves downwards. We say that a point p is *at position k at time t* if it is the kth point from above at this moment t. Let $P_k(S)$ denote the sequence of points at position k during the time interval from minus infinity to plus infinity. This model of moving points is also considered by Ottmann and Wood [OW] who examine the construction of $P_1(S)$, for example.

THEOREM 4.8. *Let S be a set of n moving points on a line. Then there exists an algorithm which computes $P_k(S)$ in $O(b_k(n)\log^2 n)$ time and $O(n+|P_k(S)|)$ space, where $|P_k(S)|$ denotes the length of $P_k(S)$.*

Proof. Each point p of S is transformed into a line by introducing time as second coordinate with horizontal axis, say. Then the line corresponding to p intersects the vertical coordinate axis at location p_0 and has slope p_s. Thus, our problem transforms to computing the upper boundary of the k-belt of the line arrangement obtained. The assertion follows from Theorem 3.4 which completes the argument.

4.5. Minimum area triangle. Let S denote a set of n points in the plane, with $n \geqq 3$. A *minimum area triangle of S* is a triangle with minimum area whose vertices are chosen from S. Dobkin and Munro [DM] were the first to come up with a nontrivial solution that takes $O(n^2\log^2 n)$ time and $O(n^2\log n)$ space. Their method is based on

Observation 4.9. Let TR denote a minimum area triangle of S with vertices p, q, and r. Then r is a point in S different from p and q which is nearest to the line through p and q.

Observation 4.9 can be exploited for the determination of TR: for each line through two points of S compute a nearest of the remaining points. Let us consider the scenario in the dual space obtained by application of the transform T. The line through p and q corresponds to the intersection v of $T(p)$ and $T(q)$ and r corresponds to a line immediately below or above this intersection point, that is, r is hit first when v moves upwards or downwards in the vertical direction. Note that the line immediately below (or above) this intersection point is not unique if several lines intersect exactly immediately below (or above) this point. A trivial modification of Algorithm TRIVIAL PLANE SWEEP described in § 3 yields:

THEOREM 4.10. *Let S denote a set of $n \geqq 3$ points in the plane. Then there exists an algorithm which computes the minimum area triangle of S in $O(n^2\log n)$ time and $O(n)$ space.*

We note that following our developments [CGL], [EOS] developed an $O(n^2)$ time and space algorithm for minimum area triangles which is based on the same geometric observations as the algorithm above. The improvement in time is achieved by a new method for constructing a complete arrangement.

5. Discussions and dynamization. The authors consider the introduction of k-belts in arrangements of lines as the main contribution of this paper. This concept has applications to several problems defined for lines or, in dual space, for points. Among these applications are

(i) space efficient data structures for halfplanar range estimation in the plane,

(ii) finding centerpoints in the plane,

(iii) determining k first points for query directions, and

(iv) computing the sequence of kth points of a dynamically changing set on a line.

All these applications rely on an efficient algorithm which computes a belt with $O(\log^2 n)$ time per edge. A less efficient but more powerful method which constructs belts in $O(n^2 \log n)$ time can also be used to find a triangle with minimum area whose vertices are taken from a finite set of points in the plane.

All data structures presented for halfplanar range estimation are inherently *static*, that is, there is no way to perform insertions of new points or deletions of points efficiently. Data structures which accommodate those two operations are called *dynamic*. There exists an extensive literature about general methods which convert static data structures for so-called decomposable searching problems into dynamic ones, see e.g. Bentley [B], Maurer and Ottmann [MO], and Overmars and van Leeuwen [OvL2]. A searching problem is said to be *decomposable* if the answer for any set S of objects and any query objects q can be derived in constant time from the answers for S_1 and q, and for S_2 and q, respectively, for every partition of S into S_1 and S_2. It is easily verified that the halfplanar range search problem is decomposable while the halfplanar range estimation problems considered in § 4.1 are not. Nevertheless, the general dynamization methods can be applied to the data structures of § 4.1. The strange effect of those conversions is that the data structures as well as the problems are changed in a meaningful way.

Representative for other and similar cases, let us consider the simple $EP(\lfloor n/3 \rfloor)$ for a set S of n points in the plane. Let $S_1, S_2, \cdots, S_m$ be an arbitrary partition of S into subsets of about $n^{1/2}$ points each. Hence, m is also about $n^{1/2}$. Let $|S_i|$ denote the number of points in S_i. Instead of the $\lfloor n/3 \rfloor$-belt for $T(S)$ we maintain an $\lfloor |S_i|/3 \rfloor$-belt for each subset S_i. For details of the maintenance strategies of this collection of belts we refer to Maurer and Ottmann [MO]. A point is inserted into (or deleted from) the system by reconstructing only one belt which takes $O(b_k(n/m) \log^2 n)$ time, with k about $n^{1/2}/3$. A query is performed on each data structure which takes $O(n^{1/2} \log n)$ time. Let $A(h)$ denote the exact number of points of S contained in h. The m answers can be used to derive an interval of length about $n/3$ which contains $A(h)$. This interval can be derived from the m answers since the answer for S_i, with $1 \leqq i \leqq m$, decides for about two thirds of the points in S_i whether or not they are contained in the query halfplane.

Let us finally give a number of open problems which come naturally from the investigations presented. (1) Give tighter bounds on $f_k(n)$, that is, improve Proposition 2.1. It is worthwhile to note that Erdös, Lovasz, Simmons and Straus [ELSS] and Edelsbrunner and Welzl [EW1] conjecture that the derived lower bounds are closer to the truth than the upper bounds. (2) Section 4.1 has exploited collections of k-belts as data structures. Let K be some fixed subset of $\{1, 2, \cdots, \lceil n/2 \rceil\}$ and let $F_K(n)$ denote the maximum of $\sum_{k \in K} f_k(S)$ for all sets S of n points in the plane. Calculate a lower and an upper bound for $F_K(n)$. (3) Let $a(S)$ denote the maximal number of edges of an arbitrary x-monotone polygonal chain which is a subset of $\bigcup_{p \in S} T(p)$. Let $a(n)$ denote the maximum of $a(S)$ for all sets S of n points in the plane. Note that Proposition 2.1 implies that $a(n)$ is in $\Omega(n \log n)$. Derive an upper bound for $a(n)$. (4) Can the construction of a k-belt of a set of n lines be accomplished in $O(b_k(n) \log n)$ time? (Compare Theorem 3.4.) (5) The data structure described in § 4.3 has similarities with the order-k Voronoi diagram, see Shamos and Hoey [SH] or Lee [L]. This diagram for a set S of n planar points is a partition of the plane into convex regions. Each region R is associated with a subset S' of S which contains k points such that an arbitrary point falls into R if and only if the points in S' are nearer to p

than the points in $S - S'$. Does there exist an algorithm which constructs the order-k Voronoi diagram of S based on the method described in § 4.3? It is tempting to conjecture that such an algorithm can improve the algorithm due to Lee [L] which constructs, successively, all order-i Voronoi diagrams, for $1 \leqq i \leqq k$. (6) At last, it is also interesting to pose the same questions in a higher dimensional environment. Most important: How many k-sets can be realized by a set of n points in three dimensions?

Note added in proof. It has been proved that $F_K(n) \in O(n(\sum_{k \in K} k)^{1/2})$ (E. Welzl, *More on k-sets of finite sets in the plane*, Rep. 204, Inst. for Information Processing, Technical Univ. Graz, Graz, Austria, 1985). Moreover, we observed that $a(n) \in \Omega(n^2)$.

REFERENCES

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

[B] J. L. BENTLEY, *Decomposable searching problems*, Inform. Process. Lett., 8 (1979), pp. 244-251.

[BF] J. L. BENTLEY AND J. H. FRIEDMAN, *Data structures for range searching*, ACM Comput. Surveys, 11 (1979), pp. 397-409.

[Br] K. Q. BROWN, *Geometric transforms for fast geometric algorithms*, Ph.D. thesis, Rep. CMU-CS-80-101, Dept. Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, 1980.

[CGL] B. M. CHAZELLE, L. J. GUIBAS AND D. T. LEE, *The power of geometric duality*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, 1983, pp. 217-225.

[C] R. COLE, *Slowing down sorting networks to obtain faster sorting algorithms*, Rep. 117, Dept. Comput. Sci., New York Univ., NY, 1984.

[CSY] R. COLE, M. SHARIR AND C. K. YAP, *On k-hulls and related problems*, Proc. 16th Annual ACMS Symposium on Theory of Computing 1984, pp. 154-166.

[DM] D. P. DOBKIN AND J. I. MUNRO, private communication.

[EKM] H. EDELSBRUNNER, D. G. KIRKPATRICK AND H. A. MAURER, *Polygonal intersection searching*, Inform. Process. Lett., 14 (1982), pp. 74-79.

[EOS] H. EDELSBRUNNER, J. O'ROURKE AND R. SEIDEL, *Constructing arrangements of lines and hyperplanes with applications*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Sciences, 1983, pp. 83-91; this Journal, 15 (1986), to appear.

[EW1] H. EDELSBRUNNER AND E. WELZL, *On the number of line-separations of a finite set in the plane*, J. Combin. Theory Ser. A, 38 (1985), pp. 15-29.

[EW2] ——— *Halfplanar range search in linear space and $O(n^{0.695})$ query time*, Rep. F111, Inst. for Information Processing, Technical Univ. Graz, Graz, Austria, 1983.

[ELSS] P. ERDÖS, L. LOVASZ, A. SIMMONS AND E. G. STRAUS, *Dissection graphs of planar point sets*, in A Survey of Combinatorial Theory, J. N. Srivastava et al., eds., North-Holland, Amsterdam, 1973, pp. 139-149.

[F] M. L. FREDMAN, *The inherent complexity of dynamic data structures which accommodate range queries*, Proc. 21st Annual IEEE Symposium on Foundations of Computer Science, 1980, pp. 191-199.

[H] E. HELLY, *Ueber Mengen konvexer Koerper mit gemeinschaftlichen Punkten*, Jber. deutsch. Math. Verein., 32 (1923), pp. 175-176.

[K] D. G. KIRKPATRICK, *Optimal search in planar subdivisions*, this Journal, 12 (1983), pp. 28-35.

[L] D. T. LEE, *On k-nearest neighbor Voronoi diagrams in the plane*, IEEE Trans. Comput. C-31 (1982), pp. 478-487.

[MO] H. A. MAURER AND TH. OTTMANN, *Dynamic solutions of decomposable searching problems*, in Discrete Structures and Algorithms, U. Pape, ed., Carl Hanser, 1979, pp. 17-24.

[OW] TH. OTTMANN AND D. WOOD, *Dynamical sets of points*, Rep. CS-82-56, Dept. Computer Science, Univ. Waterloo, Waterloo, Ontario, Canada, 1982.

[OvL1] M. H. OVERMARS AND L. VAN LEEUWEN, *Maintenance of configurations in the plane*, J. Comput. System Sci., 23 (1981), pp. 166-204.

[OvL2] M. H. OVERMARS AND J. VAN LEEUWEN, *Worst-case optimal insertion and deletion for decomposable searching problems*, Inform. Process. Lett., 12 (1981), pp. 168-173.

[S] M. I. SHAMOS, *Computational geometry*, Ph.D. Thesis, Dept. Computer Science, Yale Univ., New Haven, CT, 1978.

[SH] M. I. SHAMOS AND D. HOEY, *Closest-point problems*, Proc. 16th Annual IEEE Symposium on Foundations of Computer Science, 1976, pp. 208-215.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
020

AVERAGE CASE COMPLETE PROBLEMS*

LEONID A. LEVIN†

Abstract. Many interesting combinatorial problems were found to be NP-complete. Since there is little hope to solve them fast in the worst case, researchers look for algorithms which are fast just "on average". This matter is sensitive to the choice of a particular NP-complete problem and a probability distribution of its instances. Some of these tasks were easy and some not. But one needs a way to distinguish the "difficult on average" problems. Such negative results could not only save "positive" efforts but may also be used in areas (like cryptography) where hardness of some problems is a frequent assumption. It is shown below that the Tiling problem with uniform distribution of instances has no polynomial "on average" algorithm, unless every NP-problem with every simple probability distribution has it. It is interesting to try to prove similar statements for other NP-problems which resisted so far "average case" attacks.

Key words. complexity, algorithm, probability, completeness

Conventions. A *random problem* is a pair (μ, R), where $R \subset \{1, 2, \cdots\}^2$ is an "instance-witness" (or input-output) relation, and $\mu: \{1, 2, \cdots\} \mapsto [0, 1]$ is a probability *distribution function* on inputs (i.e. $\mu(x)$ is the probability of all instances not exceeding x). Its *density* $\mu'(x) = \mu(x) - \mu(x-1)$ is the probability of a particular input. A problem is in NP, if both R and μ are computable in time polynomial in length $|x| = \lceil \log x \rceil$ of input. A machine independent notion of a *polynomial on average* problem (μ, R) assumes $\bar{R}(x) \Leftrightarrow \exists y R(x, y)$ to be computable in time polynomial in t where $t(x)/|x|$ is bounded by a constant on average i.e. $\sum \mu'(x) t(x)/|x| < \infty$. *Domination* $\mu \lesssim \mu_1$ means $\exists k \forall x\ \mu'(x)/\mu_1'(x) < |x|^k$. A polynomial time algorithm f *reduces* a problem (μ_1, R_1) to $(f(\mu_2), R_2)$, if $\mu_1 \lesssim \mu_2$ (so, likely inputs of one problem are mapped into likely inputs of the other) and $\bar{R}_1(x) \Leftrightarrow \bar{R}_2(f(x))$. Here $f(\mu)$ is the distribution of outputs of f and maps x to $\sum_{f(y) \leq x} \mu'(y)$.

Reductions are closed under composition, and if $A(x)$ is a fast on average algorithm for $(f(\mu_2), R_2)$ then $A(f(x))$ works at most polynomially slower for (μ_1, R_1). The polynomials $|x|^k$ in domination and in reduction time may be replaced by a polynomial on average $t^k(x)$ to get *weak* reducibility. The definitions can also be modified for a more elegant "inverting" formulation of NP problems: to actually find y for which $x = r(y)$.

DEFINITION. A random NP problem is *complete*, if every random NP problem is reducible to it.

Example: Tiling. A tile is a square with a latin letter in every node. Tiles with matching letters can be joined. An instance (u, v, s) of the Tiling problem, has a subset u of tile types considered "legal", a string $v = 0^n$ of 0's, and a string s of matching legal tiles. The problem is to extend s to a square of n^2 matching legal tiles. The joint probability, of u, n and $k = |s| < n$ is, say, $O(n^{-3})$ and every tile in s is chosen sequentially with equal probability for all "legal" tiles matching the previous one.

PROPOSITION. *Tiling is an* NP*-complete random problem.*

Proof. Padding makes (μ_1, R_1) computable in time, say, $|x|^2$. We first reduce (μ_1, R_1) (no matter how special μ_1 may be) to a problem with "almost uniform" distribution. If $\mu \gtrsim \mu_1$ then $f(x) = x$ reduces (μ_1, R) to (μ, R). Thus, using $\mu_1(x) := \mu_1(x)/2 + 1/2 - 1/2x$, we get $\mu_1'(x) > 1/2x^2$. Now, by a linear number of successive

* Received by the editors September 12, 1983, and in final revised form March 18, 1985. This work was supported by the National Science Foundation under grants MCS-8104211 and MCS-8304498.

† Boston University, Boston, Massachusetts 02215, and Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

roundings, μ_1 is replaced by a *perfectly rounded* $\mu > \mu_1/4$, which means that $\mu(x)$ is the shortest binary rational within $(\mu(x-1),\ \mu(x+1))$. As all perfectly rounded measures, μ has *integer* $m(x) = \mu(x)/\mu'(x)$ and $\log_2 \mu'(x)$. Monotone μ is computable and invertible (by binary search) in polynomial time. And so is m since $\mu(x)/m(x) = \mu'(x)$ is a power of 2 and $1/2 < \mu(x) < 1$. The resulting probability of $z = m(x)$ is $\mu'(m^{-1}(z)) = \mu'(x) = \mu(x)/m(x) < 1/m(x) = 1/z$. So, $m(\mu)$ is "almost uniform".

Let p be the program for a universal Turing machine U with time bound $O(|x|^3)$ for which $R(x, y) = U(0^{|x|}1pm(x), y)$. Let $\lambda'(0^n1s) = O(n^{-3})/s$ for $|s| < 3n$. Then $f(x) = 0^{|x|}1pm(x)$ reduces (μ, R) to (λ, U), since $\lambda'(f(x)) = \lambda'(0^{|x|}1pm(x)) = O(|x|^{-3})/pm(x) \geq 1/m(x) \geqq \mu'(x)$. Finally, (λ, U) is reducible to the Tiling problem in a standard way: the tiled square corresponds to the space-time history of the Turing computation accepting $U(w, y)$, where w is chosen randomly and y is guessed non-deterministically. A Tile letter represents either the tape symbol and the direction (left or right) to the head or the head state and the direction to the neighboring cell it looks at. □

COROLLARY. *For any* $\varepsilon < 0$, *Tiling is polynomial on average iff it is polynomial with probability* $1 - n^{\varepsilon}$ (*by the "padding" argument*) *and only if such are all* NP *random problems.*

Random NP problems look like "fair games" between suppliers of questions and answers (if both are restricted to a polynomial-time probabilistic machine). So, their average hardness seems to be a more balanced question than "P = NP?".

Essential intuitive comments and an excellent survey of the area may be found in [1]. The author is grateful to R. Rivest for encouragement and discussion, to D. Johnson for valuable corrections and to U. Vazirani and Z. Galil for improvements of the account.

REFERENCE

[1] DAVID S. JOHNSON, *The* NP-*completeness column: An ongoing guide.* J. Algorithms, 5 (1984), pp. 284–299.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
021

THE ULTIMATE PLANAR CONVEX HULL ALGORITHM?*

DAVID G. KIRKPATRICK† AND RAIMUND SEIDEL‡

Abstract. We present a new planar convex hull algorithm with worst case time complexity $O(n \log H)$ where n is the size of the input set and H is the size of the output set, i.e. the number of vertices found to be on the hull. We also show that this algorithm is asymptotically worst case optimal on a rather realistic model of computation even if the complexity of the problem is measured in terms of input as well as output size. The algorithm relies on a variation of the divide-and-conquer paradigm which we call the "marriage-before-conquest" principle and which appears to be interesting in its own right.

Key words. computational geometry, convex hull, divide-and-conquer, lower bounds

AMS(MOS) subject classifications. 68P10, 52-04, 52A10

1. Introduction.

The *convex hull* of a finite point set S in the plane is the smallest convex polygon containing the set. The vertices (corners) of this polygon must be points of S. Thus in order to compute the convex hull of a set S it is necessary to find those points of S which are vertices of the hull. For the purposes of constructing upper bounds we define the *convex hull problem*, as the problem of constructing the ordered sequence of points of S which constitute the sequences of vertices around the hull.

The convex hull problem was one of the first problems in the field of computational geometry to have been studied from the point of view of computational complexity. In fact, efficient algorithmic solutions were proposed even before the term "computational geometry" was coined. This, along with its very extensive analysis in recent years, reflects both the theoretical and practical importance of the problem.

Of the convex hull algorithms proposed so far several have $O(n \log n)$ worst case time bounds [4], [8], [14], [15], [17], where n is the size of the input point set. Shamos [17] even argued that the $O(n \log n)$ time bound is worst case optimal. He observed that a set S of n real numbers could be sorted by finding the convex hull of the planar set $S' = \{(x, x^2) | x \in S\}$. But sorting, of course, has an $\Omega(n \log n)$ lower bound on a wide range of computational models. Yao [19] and on weaker computational models Avis [2], van Emde Boas [7], and Preparata and Hong [15] proved the $\Omega(n \log n)$ bound for a less demanding version of the convex hull problem: just the vertices of the convex hull are to be identified, irrespective of their sequence.

In contrast to the results above, it is interesting to observe that algorithms exist which solve the planar convex hull problem in $O(nH)$ time, where H is the number of vertices found to be on the hull [6], [9]. For small H, these algorithms seem to be superior to the $O(n \log n)$ methods. (This, of course, does not contradict the previously cited lower bound results, as H could be as large as n). It is notable, however, that all of the lower bound arguments mentioned above are insensitive to H in that they assume that some fixed fraction of the data points are vertices of the convex hull.

In this paper we present a convex hull algorithm with worst case time complexity $O(n \log H)$. Thus its running time is not only sensitive to both n and H, but it is also worst case optimal in the traditional sense when the running time is measured as a function of n only. However, we also show that our algorithm is asymptotically worst case optimal even if the complexity of the problem is measured as a function of both n and H.

* Received by the editors November 15, 1983, and in revised form August 15, 1984. This research was supported by the Natural Sciences and Engineering Research Council of Canada, grant A3583.

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

‡ Department of Computer Science, Cornell University, Ithaca, New York 14853.

Our algorithm is based on a variation of the divide-and-conquer paradigm that appears to be interesting in its own right. Traditional divide-and-conquer algorithms adhere to the following strategy: First break the problem into subproblems (*divide*), then recursively solve the subproblems (*conquer*), and finally combine the subsolutions to form the global solution (*marry*). Our algorithm reverses the last two steps. After dividing the problem it first determines how the solutions of the subproblems will combine (without actually computing them!) and then proceeds to solve the subproblems recursively. We thus call this approach the "marriage-before-conquest" principle. Its advantage lies in the fact that it allows to remove parts of the subproblems that upon merging (or marrying) turn out to be redundant. Thus it reduces the sizes of the subproblems that are to be solved recursively. We have recently been able to apply the marriage-before-conquest principle also successfully to the maximal vector problem [10]. It remains to be seen whether this principle has other applications.

Sections 2 and 3 of this paper describe our new algorithm. In § 4 we show how our algorithm can be randomized, and § 5 deals with the lower bound aspects of the convex hull problem. Throughout the paper, unless stated otherwise, we deal with sets of points in the plane. For a point p, $x(p)$ and $y(p)$ denote its standard cartesian coordinates. We will feel free to use loose but descriptive geometric terminology such as "vertical line", "a point lies above a line", etc.

2. The main algorithm. In this section we show how the "marriage-before-conquest" principle can be used for an improved convex hull algorithm. We construct the convex hull in two pieces, the upper hull and the lower hull (see Fig. 2.1). It should be clear that if the two chains forming the upper and lower hull are given, they can be concatenated in constant time (at most two vertical edges may need to be inserted) to yield the sequence of vertices around the hull. Also observe that an algorithm for constructing the upper hull could easily be modified to construct the lower hull also. Therefore we concentrate at first on constructing an algorithm for finding the sequence of vertices on the upper hull.

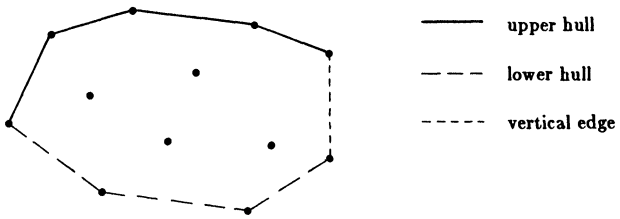


FIG. 2.1

Exploiting the "marriage-before-conquest" principle, our convex hull algorithm should do something like the following: First find a vertical line that divides the given point set in two approximately equal sized parts. Next determine the "bridge" crossing this line, i.e. the edge of the upper hull that intersects this line. Eliminate the points that lie underneath the bridge, and finally apply the algorithm recursively to the two sets of the remaining points on the left and right side of the vertical line.

The only difficult part in such an algorithm appears to be the construction of the bridge. We show a linear time solution to this problem in § 3.

The following PIDGIN-ALGOL routine presents our convex hull algorithm in some detail. It takes as input a set $S = \{p_1, \cdots, p_n\}$ of n points in the plane and prints the sequence of indices of the vertices on the upper hull of S. It uses the function BRIDGE specified in § 3, which given a set $S \subset \mathbf{R}^2$ and a real a returns the indices to

the left and right endpoint of the edge of the upper hull that intersects the vertical line $L=\{(x, y)|x=a\}$.[1]

ALGORITHM 2.1.
Procedure UPPER-HULL(S)

1. Initialization
   Let *min* and *max* be the indices of two points in S that form the left and right endpoint of the upper hull of S respectively, i.e.

$$x(p_{min}) \leqq x(p_i) \leqq x(p_{max}) \text{ and}$$
$$y(p_{min}) \geqq y(p_i) \quad \text{if} \quad x(p_{min}) = x(p_i),$$
$$y(p_{max}) \geqq y(p_i) \quad \text{if} \quad x(p_{max}) = x(p_i) \quad \text{for } i = 1, \cdots, n.$$

   If $min = max$ then print *min* and stop.
   Let $T := \{p_{min}, p_{max}\} \cup \{p \in S | x(p_{min}) < x(p) < x(p_{max})\}$.
2. CONNECT (*min*, *max*, T)
   where CONNECT (k, m, S) is
   begin
2.1 Find a real number a such that
   $x(p_i) \leqq a$ for $\lceil |S|/2 \rceil$ points in S and
   $x(p_i) \geqq a$ for $\lfloor |S|/2 \rfloor$ points in S.
2.2 Find the "bridge" over the vertical line $L=\{(x, y)|x=a\}$, i.e.
   $(i, j) :=$ BRIDGE (S, a).
2.3[2] Let $S_{left} := \{p_i\} \cup \{p \in S | x(p) < x(p_i)\}$.
   Let $S_{right} := \{p_j\} \cup \{p \in S | x(p) > x(p_j)\}$.
2.4 If $i = k$ then print (i)
   else CONNECT (k, i, S_{left}).
   If $j = m$ then print (j)
   else CONNECT (j, m, S_{right}).
   end.

THEOREM 2.1. *Algorithm* UPPER-HULL *correctly determines the sequence of vertices on the upper hull of S in $O(n)$ space and $O(n \log H_u)$ time, where H_u is the number of edges on the upper hull of S.*

Proof. If the upper hull of S consists of only one vertex (i.e. all of S lies on one vertical line) then the algorithm is trivially correct and reports that vertex in linear time in step 1.

Otherwise the correctness of the algorithm follows from an inductive argument. A call CONNECT (k, m, S) discovers a previously unknown edge (p_i, p_j) on the upper hull. If p_i turns out to be the leftmost vertex of the upper hull its index will be printed, otherwise the recursive call CONNECT (k, i, S_{left}) will cause the sequence of vertices of the upper hull from p_k up to p_i to be printed. Similarly, if p_j is the rightmost vertex of the upper hull its index will be printed, otherwise the call CONNECT (j, m, S_{right}) will cause the portion of the upper hull from p_j up to p_m to be printed.

For the complexity bounds first observe that step 1 of the algorithm can easily be implemented to run in linear time. Thus it remains to show that the procedure

[1] In the case that two edges of the upper hull, (p_i, p_j) and (p_j, p_k), intersect L, i.e. vertex p_j lies on L, BRIDGE will return (j, k).

[2] S_{left} contains p_i and the points of S to the left of the vertical line through p_i. M. McQueen from McGill University has pointed out that S_{left} could be restricted to contain p_k, p_i and all the points of S above the straight line through p_k and p_i. S_{right} can be restricted analogously.

CONNECT takes no more than $O(n \log H_u)$ time. Note that using the median finding algorithm of Blum et al. [1, p. 99] and using our bridge finding algorithm of § 3, steps 2.1 to 2.3 can be implemented to run in linear time. Thus the running time of CONNECT is determined by $f(|S|, H_u)$ where the function f must satisfy the recurrence relation

$$f(n, h) \leqq \begin{cases} cn & \text{if } h = 2, \\ cn + \max\limits_{h_l + h_r = h} \left\{ f\left(\frac{n}{2}, h_l\right) + f\left(\frac{n}{2}, h_r\right) \right\} & \text{if } h > 2, \end{cases}$$

where c is some positive constant and $n \geqq h > 1$.

We claim that $f(n, h) = O(n \log h)$. To prove this we show that $f(n, h) = cn \log h$ satisfies the above recurrence relation. This is trivially true for the base case $h = 2$. For $h > 2$ note that

$$f(n, h) \leqq cn + \max_{h_l + h_r = h} \left\{ c\frac{n}{2} \log h_l + c\frac{n}{2} \log h_r \right\}$$

$$= cn + \frac{1}{2} cn \max_{h_l + h_r = h} \{\log (h_l h_r)\}.$$

Using elementary calculus it is easy to verify that the maximum is realized when $h_l = h_r = h/2$. Thus

$$f(n, h) \leqq cn + \frac{1}{2} cn \log \left(\frac{h}{2}\right)^2 = cn + cn \log \left(\frac{h}{2}\right)$$

$$= cn + cn \log h - cn = cn \log h.$$

The linear space bound is trivial. Q.E.D.

COROLLARY. *The convex hull of a set of n points in the plane can be found in time $O(n \log H)$ using $O(n)$ space, where H is the number of vertices found to be on the hull.*

3. Finding the bridge. We are given a set S of n points in the plane and a vertical line L which has points of S to its left and right. We are to find the edge of the upper hull of S that intersects L. If two edges intersect L, i.e. L contains a vertex v of the upper hull, we want to identify the edge for which v is the left endpoint. Call this edge the *bridge* and its endpoints *bridge points* (see Fig. 3.1). Let us define a *supporting*

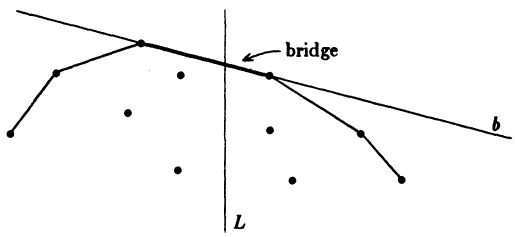


FIG. 3.1

line of S to be a nonvertical straight line which contains at least one point of S but has no points of S above it. Obviously the bridge must be contained in some supporting line. Call this line b and let s_b be the slope of b.

For our purposes, finding the bridge means identifying the two bridge points. One possible way of achieving this is to successively eliminate points from S as candidates for bridge points. For this purpose we pair up the points of S into $\lfloor n/2 \rfloor$ couples. The

following two lemmas show how forming pairs of points facilitates the elimination of candidates for bridge points.

LEMMA 3.1. *Let p, q be a pair of points of S. If $x(p)=x(q)$ and $y(p)>y(q)$ then q cannot be a bridge point.*

Proof. Trivial.

LEMMA 3.2. *Let p, q be a pair of points of S with $x(p)<x(q)$, and let s_{pq} be the slope of the straight line h through p and q.*

(1) *If $s_{pq}>s_b$ then p cannot be a bridge point.*

(2) *If $s_{pq}<s_b$ then q cannot be a bridge point.*

Proof (for case (1); the proof for case (2) is symmetrical). Assume p was a bridge point. By virtue of $s_{pq}>s_b$ and $x(p)<x(q)$, q would lie above the bridge line b which would contradict the fact that b is a supporting line of S (see Fig. 3.2). Q.E.D.

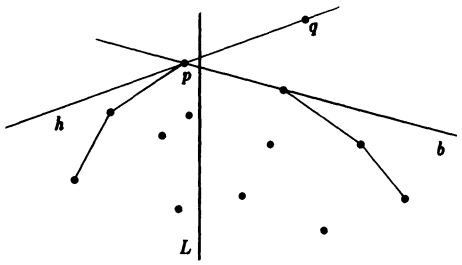


FIG. 3.2

These two lemmas can be used to eliminate a bridgepoint candidate from every one of the $\lfloor n/2 \rfloor$ pairs. However, it is not clear at first how a condition like $s_{pq}>s_b$ can be tested without explicitly knowing s_b, the slope of b, and hence knowing the bridge, which after all is the entity that we want to compute. The solution to this problem is suggested by the following lemma.

LEMMA 3.3. *Let h be the supporting line of S with slope s_h.*

(1) *$s_h<s_b$ iff h contains only points of S that are strictly to the right of L.*

(2) *$s_h=s_b$ iff h contains a point of S that is strictly to the right of L and a point of S that is to the left of or on L.*

(3) *$s_h>s_b$ iff h contains only points of S that are to the left of or on L.*

Proof. Trivial.

Thus to test whether $s_{pq}>s_b$ it suffices to find the supporting line h of S with slope s_{pq} and to determine whether h contains points of S to the right or to the left of L. Of course, finding this supporting line h requires linear time which is clearly too expensive to be done for every one of the $\lfloor n/2 \rfloor$ pairs individually. However, this problem can be overcome by judiciously choosing a slope s_h with the property that if $s_h>s_b$ then $s_{pq}>s_h$ (and hence $s_{pq}>s_b$) for a large number of pairs p, q and, if $s_h<s_b$ then $s_{pq}<s_h$ (and hence $s_{pq}<s_b$) for a large number of pairs p, q. A natural choice for an s_h with this property is the median of the slopes of the lines defined by the $\lfloor n/2 \rfloor$ pairs of points.

Now we are ready to give a more detailed PIDGIN-ALGOL description of our bridge finding algorithm. The function BRIDGE (S, a) takes as parameters a set $S=\{p_1, \cdots, p_n\}$ of $n>1$ points and a real number a representing the vertical line $L=\{(x, y) | x=a\}$. It is assumed that the point p_{min} in S with minimum x-coordinate is unique and that $x(p_{min}) \leqq a$. Similarly, the point p_{max} in S with maximum x-coordinate is assumed to be unique and with $x(p_{max})>a$. BRIDGE (S, a) returns as its value a pair (i, j), where p_i and p_j are the left and right bridge point respectively.

ALGORITHM 3.1.
Function BRIDGE (S, a)

0. $CANDIDATES := \varnothing$
1. If $|S| = 2$ then return $((i, j))$, where $S = \{p_i, p_j\}$ and $x(p_i) < x(p_j)$.
2. Choose $\lfloor |S|/2 \rfloor$ disjoint sets of size 2 from S.
   If a point of S remains, then insert it into *CANDIDATES*.
   Arrange each subset to be an ordered pair (p_i, p_j), such that $x(p_i) \leqq x(p_j)$.
   Let *PAIRS* be the set of these ordered pairs.
3. Determine the slopes of the straight lines defined by the pairs.
   In case the slope does not exist for some pair, apply Lemma 3.1, i.e.:
   For all (p_i, p_j) in *PAIRS* do
   if $x(p_i) = x(p_j)$ then delete (p_i, p_j) from *PAIRS*
   if $y(p_i) > y(p_j)$ then insert p_i into *CANDIDATES*
   else insert p_j into *CANDIDATES*

   else let $k(p_i, p_j) := \dfrac{y(p_i) - y(p_j)}{x(p_i) - x(p_j)}$.

4. Determine K, the median of $\{k(p_i, p_j) | (p_i, p_j) \in PAIRS\}$.
5. Let $SMALL := \{(p_i, p_j) \in PAIRS | k(p_i, p_j) < K\}$.
   Let $EQUAL := \{(p_i, p_j) \in PAIRS | k(p_i, p_j) = K\}$.
   Let $LARGE := \{(p_i, p_j) \in PAIRS | k(p_i, p_j) > K\}$.
6. Find the set of points of S which lie on the supporting line h with slope K, i.e.:
   Let MAX be the set of points $p_i \in S$, s.t. $y(p_i) - K * x(p_i)$ is maximum.
   Let p_k be the point in MAX with minimum x-coordinate.
   Let p_m be the point in MAX with maximum x-coordinate.
7. Determine if h contains the bridge, i.e.:
   if $x(p_k) \leqq a$ and $x(p_m) > a$ then return$((k, m))$.
8. h contains only points to the left of or on L:
   if $x(p_m) \leqq a$ then
   for all $(p_i, p_j) \in LARGE \cup EQUAL$ insert p_j into *CANDIDATES*.
   for all $(p_i, p_j) \in SMALL$ insert p_i and p_j into *CANDIDATES*.
9. h contains only points to the right of L:
   if $x(p_k) > a$ then
   for all $(p_i, p_j) \in SMALL \cup EQUAL$ insert p_i into *CANDIDATES*.
   for all $(p_i, p_j) \in LARGE$ insert p_i and p_j into *CANDIDATES*.
10. return(BRIDGE (*CANDIDATES*, a)).

THEOREM 3.1. *The function* BRIDGE *correctly determines the left and right bridge point in* $O(n)$ *worst case time and space.*

Proof. The algorithm is trivially correct if S contains only two points. As long as S contains more than two points, BRIDGE either finds the bridge in step 7 or discards redundant points of S applying the rules of Lemmas 3.1 and 3.2 (steps 3, 8, 9) and calls itself recursively with a smaller pointset.

Using the linear time median algorithm of Blum et al. [1, p. 99], the body of BRIDGE without the recursive call can be executed in linear time and space. Furthermore, at least one quarter of the points of S are eliminated and not contained in *CANDIDATES*. Thus the worst case time and space requirements for the algorithm

are bounded by

$$f(n)=\begin{cases}O(1), & n=2,\\ f\left(\frac{3n}{4}\right)+O(n), & n>2.\end{cases}$$

But it is well known that such a recursive function is $O(n)$ [1, p. 64]. Q.E.D.

At this point we want to mention that our bridge finding algorithm was inspired by the linear time two variable linear programming algorithms of M. Dyer [5] and N. Megiddo [13]. A closer look even shows that the bridge problem can be formulated as a linear programming problem. However, for the sake of simplicity and completeness it seems worthwhile to spell out the bridge finding algorithm explicitly.

4. The expected time case. The divide-and-conquer algorithms in the two preceding sections are not terribly complicated. At first sight it even seems possible to actually implement these algorithms in some high level programming language in an hour's time, or so. However, one quickly discovers that the major obstacle to doing so is the median find algorithm. Thus quite naturally the question arises whether it is possible to do without it.

The median find algorithm is used in our algorithms to find a vertical line that divides a given point set evenly. What happens if we follow the example of Quicksort and choose a separating line at random? Ample experimental results have shown that Quicksort is one of the fastest sorting algorithms and these results have been supported by a careful theoretical analysis of the algorithm [11], [16]. As it turns out the method of choosing a separator at random can also be successfully applied to our algorithms, thus changing the worst case time complexity to $O(n^2)$ but retaining the $O(n \log H)$ expected case time complexity.

THEOREM 4.1. *If step* 2.1 *in Algorithm* 2.1 *is replaced by*

2.1. Let $a = x(p_i)$, where p_i is randomly chosen from $S-\{p_m\}$ such that the choice of every point in $S-\{p_m\}$ is equally likely.

then the modified algorithm has $O(n \log H_u)$ expected case time complexity.

Proof. The expected case running time of the modified algorithm can be bounded by the function g that must satisfy the following relation:

$$g(n,h)\leqq\begin{cases}bn & \text{if } n\geqq h=2,\\ bn+\dfrac{1}{n-1}\displaystyle\sum_{1\leqq i<n}\max_{h_l+h_r=h}\{g(i,h_l)+g(n-i,h_r)\} & \text{if } n\geqq h>2,\end{cases}$$

where b is some positive constant.

We claim that $g(n,h)=O(n \log h)$, i.e. there is positive real constant c, such that for all $n\geqq h\geqq 2$, $g(n,h)\leqq cn \log h$.[3] We prove our claim by induction.

The claim is trivially true for all n if $h=2$ and for all $n\leqq 5$ otherwise. Now we want to show the claim for some $n>5$ and $h<n$ on the assumption that $g(n',h')\leqq cn' \log h'$ for all $n'<n$ and $h'<h$. By definition of g and our inductive assumption we thus have

$$g(n,h)\leqq bn+\frac{1}{n-1}\sum_{1\leqq i<n}\max_{h_l+h_r=h}\{ci \log h_l+c(n-i)\log h_r\}.$$

[3] In this proof we use w.l.o.g. the natural logarithm.

Using elementary calculus it is easy to show that for every i the maximum is realized when $h_l = ih/n$ and $h_r = (n-i)h/n$. Therefore

$$g(n,h) \leqq bn + \frac{c}{n-1} \sum_{1 \leqq i < n} \left(i \log i \frac{h}{n} + (n-i) \log (n-i) \frac{h}{n} \right)$$

$$= bn + \frac{2c}{(n-1)} \sum_{1 \leqq i < n} i \log i \frac{h}{n}$$

$$= bn + \frac{2c}{(n-1)} \log \frac{h}{n} \sum_{1 \leqq i < n} i + \frac{2c}{(n-1)} \sum_{1 \leqq i < n} i \log i.$$

As $\sum_{1 \leqq i < n} i \log i \leqq \frac{1}{2} n^2 \log n - \frac{1}{4} n^2$ (see [1, p. 94]) and $\sum_{1 \leqq i < n} i = \frac{1}{2} n(n-1)$ we have

$$g(n,h) \leqq bn + cn \log h - cn \log n + c \frac{n}{n-1} n \log n - \frac{c}{2} \frac{n^2}{n-1}$$

$$\leqq bn - \frac{c}{2} n + cn \frac{\log n}{n-1} + cn \log h.$$

As $\log n/(n-1) < \frac{1}{2}$ for all integers $n > 5$, there exists a real constant $c > 0$ such that $bn - \frac{1}{2}cn + cn(\log n/(n-1)) < 0$ for all $n > 5$ and hence

$$g(n,h) \leqq cn \log h. \qquad \text{Q.E.D.}$$

The median find algorithm is used on one more occasion in our algorithms: in the bridge finding procedure. Again we can dispense with the median find algorithm and use random choice instead. The worst case complexity of such a modified bridge finding procedure is $O(n^2)$; however the expected case running time is still $O(n)$.

THEOREM 4.2. *If step* 4 *of Algorithm* 3.1 *is replaced by*

4. Randomly choose an element (p_i, p_j) from *PAIRS* such that the choice of every element is equally likely, and let $K := k(p_i, p_j)$,

then the modified algorithm has expected case time complexity $O(n)$.

Proof. In the worst case no points are eliminated in step 3 of the modified function BRIDGE, and all the slopes $k(p_i, p_j)$ generated in that step are distinct. By the random choice of K, the cardinalities of *SMALL* and *LARGE* are uniformly distributed between 0 and $N-1$, where $N = \lfloor |S|/2 \rfloor$, the cardinality of *PAIRS*.

Assume pessimistically that whenever $|SMALL| \leqq N/2$, the supporting line h contains only points to the right of L, and by step 9 only $|SMALL|+1$ points are eliminated. Symmetrically, assume that if $|LARGE| < N/2$, h contains only points to the left or on L, and step 8 is applied.

With these pessimistic assumptions the expected case running time of the modified algorithm is bounded from above by the function f, where for some positive constant b

$$f(n) = \begin{cases} bn & \text{if } n \leqq 2, \\ bn + \dfrac{4}{n} \displaystyle\sum_{1 \leqq i \leqq n/4} f(n-i) & \text{if } n > 2. \end{cases}$$

It is an easy exercise in induction to show that $f(n) = O(n)$. Q.E.D.

5. Lower bounds. The results of this section demonstrate that our $O(n \log H)$ upper bound for the convex hull problem is the best possible on a quite general model of computation. Specifically, we prove an $\Omega(n \log H)$ lower bound for this problem on dth order algebraic decision trees, for any fixed d.

There exist at least four variants of the convex hull problem characterized by increasingly stringent conditions on the form of the output. Let $S = \{p_1, \cdots, p_n\}$ be a set of points in $\mathbf{R}^2$, and let $ext(S)$ denote the set of vertices of the convex hull of S. The *convex hull sequence problem* asks for the elements of $ext(S)$ in consecutive cyclic order. The *convex hull set problem* asks for the elements of $ext(S)$ in arbitrary order. The *convex hull multiset problem* asks for a listing, in arbitrary order, of elements of S that coincide with elements of $ext(S)$. (This differs from the set problem only if S is a multiset). Finally, the *convex hull size problem* asks for the cardinality of $ext(S)$ (i.e. H).

It should be clear that the algorithm outlined in § 3 can be adapted to solve all of these problem variants in worst case time $O(n \log H)$. Furthermore, since the sequence variant is at least as hard as the set variant, which in turn is at least as hard as the size variant, it will suffice to demonstrate a lower bound on the convex hull size problem, preferably using input point sets with no multiplicities. In fact we establish a lower bound on the even weaker *convex hull size verification problem*: given S and H, confirm that $|ext(S)| = H$. We show that any dth order algebraic decision tree algorithm for this verification problem must take $\Omega(n \log H)$ steps in the worst case, even if it can be assumed that all input points are distinct.

We follow Steele and Yao [18] and Ben-Or [3] in adopting algebraic decision trees as our model of computation. A *dth order algebraic decision-tree algorithm* (hereafter a *tree algorithm*) T for testing membership in a set $W \subset \mathbf{R}^n$ is a rooted tree whose internal nodes are labelled by multivariate polynomials of degree at most d and whose leaves are labelled either YES or NO. Each internal node has out-degree three; the edges are labelled $<$, $=$, and $>$ reflecting possible outcomes on comparison with 0. Every input $\vec{z} \in \mathbf{R}^n$ determines a unique root to leaf path in T in the obvious way. We say that T decides membership in W if, for every $\vec{z} \in \mathbf{R}^n$, $\vec{z}$ leads to a YES leaf of T if and only if $\vec{z} \in W$.

Yao [19] establishes an $\Omega(n \log n)$ worst case lower bound for the convex hull set problem on algebraic decision trees of order two. This result is generalized by Ben-Or [3], who demonstrates the same $\Omega(n \log n)$ lower bound for the convex hull size problem on algebraic decision trees of any fixed order d. Ben-Or's result is just one of a number of applications of the following general theorem concerning tree algorithms.

THEOREM 5.1 [3, Thm. 8]. *Let $W \subset \mathbf{R}^n$ be any set and let T be any dth order algebraic decision tree that solves the membership problem for W. If W has N disjoint connected components, then T must have height (and hence worst case complexity) $\Omega(\log N - n)$.*

We use the following generalization of the element distinctness problem [3] to establish our lower bound. The *multiset size verification problem* asks to confirm, given a multiset $Z = \{z_1, \cdots, z_n\} \subset \mathbf{R}$ and an integer k, that Z has k distinct elements.

COROLLARY 5.1. *The multiset size verification problem requires $\Omega(n \log k)$ steps in the worst case, with any dth order decision algorithm.*

Proof. It suffices to prove that the set

$$M_k = \{(z_1, \cdots, z_n) \in \mathbf{R}^n \mid |\{z_1, \cdots, z_n\}| = k\}$$

has at least $k!k^{n-k}$ disjoint connected components.

Consider all tuples $(z_1, \cdots, z_n)$ with $z_1, \cdots, z_k$ set to distinct integers between 1 and k, and $z_{k+1}, \cdots, z_n$ set to arbitrary integers in that range. There are $k!k^{n-k}$ such tuples and each of them must lie in a different connected component of M_k. Q.E.D.

We are now prepared to demonstrate our lower bound.

THEOREM 5.2. *The convex hull size verification problem requires $\Omega(n \log H)$ steps, in the worst case, with any dth order decision tree algorithm.*

Proof. We reduce the multiset size verification problem to the convex hull size verification problem in the following obvious way: Let $Z = \{z_1, \cdots, z_n\}$ and k be an instance of the multiset size verification problem. Define $S = \{p_1, \cdots, p_n\} \subset \mathbf{R}^2$ by $p_i = (z_i, z_i^2)$. Then the set $ext(S)$ has exactly k elements if and only if Z has exactly k distinct elements. Q.E.D.

The proof of the above theorem is somewhat disappointing in that the convex hull problem formed in the reduction has multiplicities on the convex hull. This straightforward reduction leaves open the possibility that there exists an algorithm solving the convex hull size verification problem (or any of the other variants) in $o(n \log H)$ steps for point sets that are known a priori to contain no duplicates. Fortunately, we can strengthen our lower bound to include tree algorithms based on this rather dubious assumption as well. We will show that a convex hull algorithm that is only guaranteed to be correct when the input points are distinct could be used to solve a certain perturbed convex hull problem without input restrictions. An algorithm for this perturbed problem in turn yields a solution for the multiset size problem.

For the sake of notation let $(\vec{x}, \vec{y})$ be shorthand for $(x_1, \cdots, x_n, y_1, \cdots, y_n)$ and let $\bar{x}$ and $\bar{y}$ denote $(1/n)\sum_{i=1}^{n} x_i$ and $(1/n)\sum_{i=1}^{n} y_i$ respectively. We call a tuple $(\vec{x}, \vec{y})$ *center-free* iff $(\bar{x}, \bar{y}) \neq (x_i, y_i)$ for $1 \leqq i \leqq n$. Define

$$C_H = \{(\vec{x}, \vec{y}) \in \mathbf{R}^{2n} \mid |\text{ext}(\{(x_i, y_i) | 1 \leqq i \leqq n\})| = H\} \quad \text{and}$$

$$P_H = \{(\vec{x}, \vec{y}) \in \mathbf{R}^{2n} \mid |\text{ext}(\{(x_i + i(x_i - \bar{x})\varepsilon, y_i + i(y_i - \bar{y})\varepsilon) | 1 \leqq i \leqq n\})| = H\}$$
for all $\varepsilon > 0$ sufficiently small.

Note that testing membership in C_H is the convex size verification problem. The intuitive meaning for P_H is the following: P_H encodes the point sets $\{(x_i, y_i) \in \mathbf{R}^2 | 1 \leqq i \leqq n\}$ with the property that if each point $p_i = (x_i, y_i)$ moved radially away from $\bar{p} = (\bar{x}, \bar{y})$ for sufficiently small but positive time ε at speed proportional to the index i and proportional to the distance from p_i to $\bar{p}$, then the convex hull of the new point set would have H extreme points. Observe therefore, that if $(\vec{x}, \vec{y}) \in C_H$ and the encoded 2-dimensional point set has no point on a convex hull edge, then $(\vec{x}, \vec{y}) \in P_H$.

The following lemma shows that the convex hull size verification problem with this dubious distinctness restriction is no easier to solve than the general membership problem in P_H for center-free tuples.

LEMMA 5.1. *Let T be any dth order decision tree algorithm for deciding membership in C_H, assuming that all of the points (x_i, y_i), $1 \leqq i \leqq n$, are distinct. Then there exists a dth order decision tree T', with height $(T') \leqq (d+1)$ height (T), that decides membership of center-free tuples in P_H without the distinctness assumption.*

Proof. We define a transformation on every subtree of T. The leaves of T are not changed (i.e. they retain their YES-NO labels). Consider an arbitrary subtree rooted at a vertex v_j with label $f_j(\vec{x}, \vec{y})$ (see Fig. 5.1). Define the multivariate polynomials $f_{j,0}, f_{j,1}, \cdots, f_{j,d}$ by the equality

$$f_j(\vec{x}', \vec{y}') = f_{j,0}(\vec{x}, \vec{y}) + f_{j,1}(\vec{x}, \vec{y})\varepsilon + \cdots + f_{j,d}(\vec{x}, \vec{y})\varepsilon^d,$$

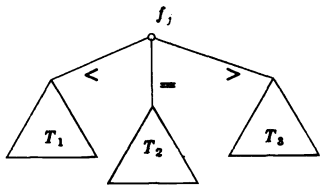


FIG. 5.1

where

$$\vec{x}' = (x_1 + (x_1 - \bar{x})\varepsilon, x_2 + 2(x_2 - \bar{x})\varepsilon, \cdots, x_n + n(x_n - \bar{x})\varepsilon) \quad \text{and}$$

$$\vec{y}' = (y_1 + (y_1 - \bar{y})\varepsilon, y_2 + 2(y_2 - \bar{y})\varepsilon, \cdots, y_n + n(y_n - \bar{y})\varepsilon).$$

Clearly, the degree of each $f_{j,k}$ is at most d.

Let T_i' be the transformed versions of T_i, $i = 1, 2, 3$. The transformed version of the full subtree is given by Fig. 5.2.

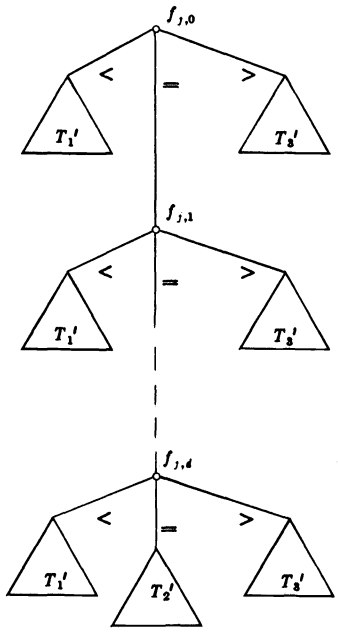


FIG. 5.2

Note that T' does not depend on ε. Furthermore, a straightforward inductive argument shows that height $(T') \leqq (d+1)$ height (T). The correctness of T' follows from the following observations.

(i) If $\varepsilon > 0$ is chosen to be sufficiently small, then for center-free $(\vec{x}, \vec{y})$ the set $\{(x_i + i(x_i - \bar{x})\varepsilon, y_i + i(y_i - \bar{y})\varepsilon), 1 \leqq i \leqq n\}$ has distinct elements.

(ii) The decision tree T' with input $(\vec{x}, \vec{y})$ agrees with the decision tree T with input $(\vec{x}', \vec{y}')$, for all sufficiently small $\varepsilon > 0$.

Observation (ii) holds since for any $(\vec{x}, \vec{y})$ the polynomial $f_j(\vec{x}', \vec{y}') = 0$ for all sufficiently small $\varepsilon > 0$ iff $f_{j,k}(\vec{x}, \vec{y}) = 0$ for all k, and otherwise the sign of $f_j(\vec{x}', \vec{y}')$ for all sufficiently small $\varepsilon > 0$ agrees with the sign of $f_{j,k}(\vec{x}, \vec{y})$ for the least k with $f_{j,k}(\vec{x}, \vec{y}) \neq 0$.

Thus T' decides membership of center-free $(\vec{x}, \vec{y})$ in P_H without assuming that all of the pairs (x_i, y_i) are distinct. Q.E.D.

The next lemma shows that deciding membership for P_H is no easier than the multiset size verification problem.

LEMMA 5.2. *The multiset size verification problem reduces to the membership problem for center-free tuples in P_H.*

Proof. It suffices to note that as no three distinct points on a parabola can be collinear

$$(x_1, \cdots, x_n) \in M_H \quad \text{iff} \quad (x_1, \cdots, x_n, x_1^2, \cdots, x_n^2) \in P_H,$$

and that $(x_1, \cdots, x_n, x_1^2, \cdots, x_n^2)$ is center-free (except for the uninteresting case when all x_i are identical). Q.E.D.

The preceding corollary and lemmas immediately yield the final theorem.

THEOREM 5.3. *The convex hull size verification problem requires $\Omega(n \log H)$ steps, in the worst case, with any dth order decision tree algorithm, even if the input points may be assumed to be distinct.*

6. Conclusions. We have introduced a variation of the familiar divide-and-conquer paradigm and have illustrated this approach in the development of a new algorithm for the planar convex hull problem. Our algorithm unifies and improves the best worst case complexity bounds known for this problem in terms of the size of input and output (i.e. number of data points and number of hull vertices). In fact, we demonstrate that the algorithm is worst case optimal in terms of these two parameters in a very general model of computation.

In a companion paper [10] we apply the same strategy to the maximal vector problem. We are able to demonstrate an $O(n \log V)$ upper bound for the 2-dimensional maximal vector problem, where V is the number of maximal vectors found. The same upper bound applies to the 3-dimensional maximal vector problem, and also to the d-dimensional maximal vector problem, $d > 3$, when V is sufficiently small compared to n. These bounds tighten the best bounds known for the maximal vector problem. It remains to be seen whether our "marriage-before-conquest" approach can be applied successfully to other problems.

The results of this paper suggest other more specific open problems as well. In particular, it is natural to ask whether our results on planar convex hulls (like those for the maximal vector problem) extend to higher dimensions. For example, does there exist an $O(n \log H)$ algorithm for the 3-dimensional convex hull problem?

Another practical open question is whether, like the algorithm of Bentley and Shamos [4], our convex hull algorithm modified as suggested in footnote 2 has linear expected time complexity for reasonable input point distributions. We suspect that this is the case.

Acknowledgments. We are grateful to John Gilbert for his very careful reading of the manuscript.

REFERENCES

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

[2] D. AVIS, *On the complexity of finding the convex hull of a set of points*, School of Computer Science, Report SOCS 79.2, McGill Univ., Montreal, 1979.

[3] M. BEN-OR, *Lower bounds for algebraic computation trees*, Proc. 15th ACM STOC, 1983, pp. 80–86.

[4] J. L. BENTLEY AND M. I. SHAMOS, *Divide and conquer for linear expected time*, Inform. Proc. Lett., 7 (1978), pp. 87-91.
[5] M. E. DYER, *Two variable linear programs are solvable in linear time*, Manuscript, Dept. Mathematics and Statistics, Teesside Polytechnic, Middlesborough, Cleveland, UK, 1982.
[6] U. F. EDDY, *A new convex hull algorithm for planar sets*, ACM Trans. Math. Software, 3 (1977), pp. 398-403 and pp. 411-412.
[7] P. VAN EMDE BOAS, *On the $O(n \log n)$ lower-bound for convex hull and maximal vector determination*, Inform. Proc. Let., 10 (1980), pp. 132-136.
[8] R. L. GRAHAM, *An efficient algorithm for determining the convex hull of a finite planar set*, Inform. Proc. Lett., 1 (1972), pp. 132-133.
[9] R. A. JARVIS, *On the identification of the convex hull of a finite set of points in the plane*, Inform. Proc. Lett., 2 (1973), pp. 18-21.
[10] D. G. KIRKPATRICK AND R. SEIDEL, *Output size sensitive algorithms for finding maximal vectors*, in Proc. ACM Symposium on Computational Geometry, 1985, pp. 89-96.
[11] D. E. KNUTH, *The Art of Computer Programming, Volume* 3, Addison-Wesley, Reading, MA, 1973.
[12] H. T. KUNG, F. LUCCIO AND F. P. PREPARATA, *On finding the maxima of a set of vectors*, J. ACM, 22 (1975), pp. 469-476.
[13] N. MEGIDDO, *Linear-time algorithms for linear programming in R^3 and related problems*, Proc. 23rd FOCS (1982), pp. 329-338.
[14] F. P. PREPARATA, *An optimal real time algorithm for planar convex hulls*, Comm. ACM, 22 (1979), pp. 402-405.
[15] 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.
[16] R. SEDGEWICK, *Quicksort*, Ph.D. thesis, Stanford Univ., Stanford, CA, 1975.
[17] M. I. SHAMOS, *Computational geometry*, Ph.D. thesis, Yale Univ., New Haven, CN, 1978.
[18] J. M. STEELE AND A. C. YAO, *Lower bounds for algebraic decision trees*, J. Algorithms, 3 (1982), pp. 1-8.
[19] A. C. YAO, *A lower bound to finding convex hulls*, J. ACM, 28 (1981), pp. 780-789.

SIAM J. COMPUT.
Vol. 15, No. 1, February 1986
© 1986 Society for Industrial and Applied Mathematics
022

COMPUTING THE LARGEST EMPTY RECTANGLE*

B. CHAZELLE†, R. L. DRYSDALE‡ AND D. T. LEE§

Abstract. We consider the following problem: Given a rectangle containing N points, find the largest area subrectangle with sides parallel to those of the original rectangle which contains none of the given points. If the rectangle is a piece of fabric or sheet metal and the points are flaws, this problem is finding the largest-area rectangular piece which can be salvaged. A previously known result [13] takes $O(N^2)$ worst-case and $O(N \log^2 N)$ expected time. This paper presents an $O(N \log^3 N)$ time, $O(N \log N)$ space algorithm to solve this problem. It uses a divide-and-conquer approach similar to the ones used by Bentley [1] and introduces a new notion of Voronoi diagram along with a method for efficient computation of certain functions over paths of a tree.

Key words. computational geometry, divide-and-conquer, free tree, location theory, optimization

1. Introduction. We consider the following problem: Given a rectangle containing n points, find the largest area subrectangle with sides parallel to those of the original rectangle which contains none of the given points. If the rectangle is a piece of fabric or sheet metal and the points are flaws, this problem is finding the largest-area rectangular piece which can be salvaged. The special case in which a largest empty square is desired has been solved in $O(n \log n)$ time using Voronoi diagrams in L_1-(L_∞-)metric [7], [12], which is just a variation of the largest empty circle problem studied by Shamos [14], [15]. In [13] an $O(n^2)$ worst-case and $O(n \log^2 n)$ expected-time algorithm is presented for the largest empty rectangle problem. Other related problems can be found in [3], [5].

This paper presents an $O(n \log^3 n)$ time, $O(n \log n)$ space algorithm to solve this problem. It uses a divide-and-conquer approach similar to the ones used by Bentley [1].

2. General approach. We first note that the largest area rectangle with sides parallel to the bounding rectangle will have each edge supported by either an edge of the bounding rectangle or by at least one of the given points. (If the set of points contains two or more points lying on a vertical or horizontal line, an edge of the largest rectangle may be supported by more than one point.) Any rectangle is uniquely determined by its four supports (points or edges of the bounding rectangle). Therefore a naive algorithm could choose quadruples of support and then test to see if any points lie inside the rectangle formed. This method requires $O(n^5)$ time. However, it is shown in [13] that the number of such empty rectangles is at most $O(n^2)$ and that by carefully maintaining those rectangles the one with the largest area can be found in $O(n^2)$ time.

We shall in this paper present a divide-and-conquer algorithm. Let $p_1, p_2, \cdots, p_n$ be the n points sorted by x-coordinate and $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$ be the boundaries of the bounding rectangle. Let the coordinates of point p_i be (x_i, y_i). Our algorithm splits the points into two halves by x-coordinate. We recursively solve the problem for the sets $S_1 = \{p_1, \cdots, p_{\lfloor n/2 \rfloor}\}$ and $S_2 = \{p_{\lfloor n/2 \rfloor + 1}, \cdots, p_n\}$. (The bounding rectangles of these recursive calls must be adjusted. The right boundary for the left call is $x_{\lfloor n/2 \rfloor}$ and the left boundary for the right call is $x_{\lfloor n/2 \rfloor + 1}$.) These calls determine the largest

* Received by the editors July 7, 1983, and in revised form September 29, 1984. This research was supported in part by the National Science Foundation under grants MCS 8202359, and ECS 8121741.

† Department of Computer Science, Brown University, Providence, Rhode Island 02912.

‡ Department of Mathematics and Computer Science, Dartmouth College, Hanover, New Hampshire 03755.

§ Department of Electrical Engineering/Computer Science, Northwestern University, Evanston, Illinois 60201.

rectangles with all four supporting points or edges in one half or the other. What remains are rectangles with supports in both halves. These rectangles contain either three supports in one half and one support in the other or two supports in each half (see Fig. 1). Our algorithm finds the largest rectangle of each type, and then returns the largest rectangle found with either all four supports in one half, three supports in one half and one in the other, or two on each side as the largest rectangle.

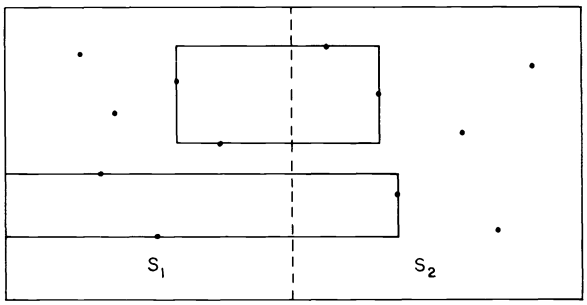


FIG. 1. *Possible empty rectangles.*

Therefore the run time of our algorithm is governed by the time required to find the largest rectangle with three supports in one half and one in the other and the time to find the largest rectangle with two supports in each half. That is, we have

$$T(N) \leqq 2T(N/2) + C(N) + D(N), \tag{1}$$

where $T(N)$ denotes the run time of the algorithm for the largest empty rectangle problem for N points, $C(N)$ the time for finding the largest empty rectangle with three supports in one half and one support in the other half and $D(N)$ the time for finding the rectangle with two supports in each half. As will be shown later, $C(N) = O(N)$ and $D(N) = O(N \log^2 N)$, which gives $T(N) = O(N \log^3 N)$.

3. Three supports in one half, one in the other. This is the easier of the two subproblems. We will look at the case of three supports in the left half and one in the right and present a linear time algorithm for this case. The other case is symmetrical, and is solved the same way.

Our first observation is that we need only consider approximately n rectangles. If the left support is a given point p_j, the rectangle is completely determined. The upper edge is supported by the first point above p_j which also lies to its right. If no such point exists, the rectangle is supported by the top edge of the bounding rectangle. Graphically, this support is found by drawing a horizontal ray to the right from p_j and sweeping it upward until it encounters either a point in the left half or the top edge of the bounding rectangle. Similarly the bottom edge is supported by the first point below p_j which also lies to its right if such a point exists, and the bottom edge of the bounding rectangle otherwise. These are the only top and bottom supports possible if all three supports are to lie in the left half. Let upper_j and lower_j denote, respectively, the upper and lower supports of the rectangle whose left support is p_j. The right support right_j is found by extending the rectangle supported above by upper_j and below by lower_j to the right until it encounters either a point in the right half or the right edge of the bounding rectangle. (Note that if several points in the left half life on a horizontal line, only the rightmost will support the left side of a rectangle.)

There are $\lfloor n/2 \rfloor + 1$ rectangles supported by the left edge of the bounding rectangle. If the p_j are sorted from top to bottom there is one rectangle above the top point, one

between each pair of points, and one below the bottom point. (If two or more points lie on a horizontal line, then some of these rectangles will be empty.)

A naive algorithm would take each point in the left and find the upper and lower supports of its rectangle, using $O(n)$ time for each. Finding the right support would take $O(n)$ additional time. Similarly, the right support of each rectangle supported by the left side of the bounding rectangle could be found in $O(n)$ time, so the whole process would take $O(n^2)$ time. However, we can find the largest rectangle in $O(n)$ time given the points sorted by y-coordinate.

Finding the upper and lower supports of rectangles with left support at the left side of the bounding rectangle is trivial if the points are sorted by y-coordinate. We present below a linear time algorithm to find the upper, lower and right supports of each rectangle supported on the left by a point p_j in the left half. Suppose that the points in the left half are sorted from top to bottom as $p_1, p_2, \cdots, p_m$. gap_j is defined to be the right support of the rectangle supported above by p_{j-1} and below by p_j, with $p_0 = (x_{\max}, y_{\max})$ and $p_{m+1} = (x_{\max}, y_{\min})$. $\text{gap}_1, \text{gap}_2, \cdots, \text{gap}_m$ are obtained in linear time with the points presorted by y-coordinate. Right_j is initialized to be the leftmost point in the right half with y-coordinate y_j, or $q_j = (x'_{\max}, y_j)$ if no such point exists, where $x'_{\max}$ is the right boundary of the current bounding rectangle for the right half. The arrays *above* and *below* are used to hold the points with running minimum x-coordinates (see Fig. 2).

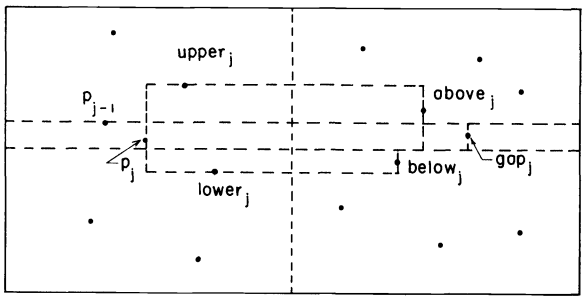


FIG. 2. *Illustration of supports of a point* p_j.

Our algorithm uses a stack. It takes advantage of the fact that when processing the points from top to bottom, the upper support of a point is the first point lying above it which also lies to its right. Therefore, points lying above the current point but to its left will never be upper supports for subsequent points and can be eliminated. A symmetric argument can be made about lower supports for which we process the points from bottom to top. This gives rise to the following algorithm.

1. Initialize the stack with upper support p_0 and $q_0 = (x'_{\max}, y_{\max})$ so that top is p_0, $x(\text{top}) = x_{\max}$, $y(\text{top}) = y_{\max}$, above (top) $= q_0$ and $x(q_0) = x'_{\max}$.
2. Scan the points $p_1, p_2, \cdots, p_m$ from top to bottom. For each point p_j encountered we do the following:
   2.1. If $x(\text{gap}(j)) < x(\text{right}(j))$ then above (j) is set to gap (j) and to right (j) otherwise.
   2.2. While $x(\text{top}) \leqq x(j)$ do;
   if $x(\text{above}(j)) \geqq x(\text{above (top)})$ then above (j) is set to above (top);
   pop the stack.
   2.3. upper (j) is set to top.
   2.4. Push p_j onto the stack.

3. Reinitialize the stack with lower support p_{m+1} and $q_{m+1}=(x'_{\max}, y_{\min})$ so that top is p_{m+1}, $x(\text{top})=x_{\max}$, $y(\text{top})=y_{\min}$, below (top) $=q_{m+1}$ and $x(q_{m+1})=x'_{\max}$.
4. Scan the points $p_1, p_2, \cdots, p_m$ from bottom to top. For each point p_j encountered we do the following:
   4.1. If $x(\text{gap}(j+1))<x(\text{right}(j))$ then below (j) is set to gap $(j+1)$ and to right (j) otherwise.
   4.2. While $x(\text{top})\leqq x(j)$ do;
   if $x(\text{below}(j))\geqq x(\text{below (top)})$ then below (j) is set to below (top);
   pop the stack.
   4.3. lower (j) is set to top.
   4.4. Push p_j onto the stack.
5. For each point p_j, $j=1, 2, \cdots, m$ do;
   if $x(\text{above}(j))<x(\text{below}(j))$
   then right (j) is set to above (j)
   else right (j) is set to below (j).

As can be easily shown, the algorithm examines each candidate left support (steps 2, 4 and 5) once, taking a total of $O(m)$ time. So we conclude this section with the following.

LEMMA 1. *The time $C(N)$ for finding the largest empty rectangle for N points with three supports in one half and one support in the other half is $O(N)$.*

4. Two supports in each half. Notice that the two supports must be on adjacent sides of the rectangle. Namely, the two supports in the left half must determine either the upper left corner or the lower left corner of the rectangle and the other two supports in the right half determine the lower right corner or the upper right corner of the rectangle respectively. Since these two cases are similar, we shall consider only the case where the two supports in the left half determine the lower left corner and the two supports in the right half determine the upper right corner of the rectangle. If we can identify all the possible lower left corner points in the left half and all the possible upper right corner points in the right half, then what remains to be solved is to find the so-called *largest empty corner rectangle* (LECR) which is determined by a corner point in each half. Therefore, we shall first compute all the possible corner points in each half and then devote ourselves to the problem of finding the largest empty corner rectangle.

4.1. Computation of corner points. We observe that two points p_i and p_j determine the lower left corner point of an empty rectangle iff p_j is lower$_i$. Thus, the point LC_i, $i=1, 2, \cdots, m$, determined by p_i and lower$_i$ is a lower left corner point and has as its x- and y-coordinates equal to x_i and $y(\text{lower}_i)$ respectively. In addition to these lower left corner points we include the points $L_i=(x_{\min}, y_i)$, $i=1, 2, \cdots, m$, i.e., the points on the left boundary, and the original set of points to form the set $CL=\{LC_1, LC_2, \cdots, LC_s\}$ where $s\leqq 3m$. All the possible upper right corner points in the right half can be computed in an analogous manner. We now have two sets of corner points $CL=\{LC_1, LC_2, \cdots, LC_s\}$ and $CR=\{RC_1, RC_2, \cdots, RC_t\}$ and want to find the largest empty (corner) rectangle whose lower left corner and upper right corner are from CL and CR respectively. Figure 3 shows the corner points in each half, with ● and × representing given and newly created points, respectively. Before we give the algorithm for finding the LECR, some observations are in order. Notice that not every point in CL can be paired with a point in CR. The empty rectangle that we seek must be a rectangle with *exactly two supports in each half.* For example, in Fig. 3 the point

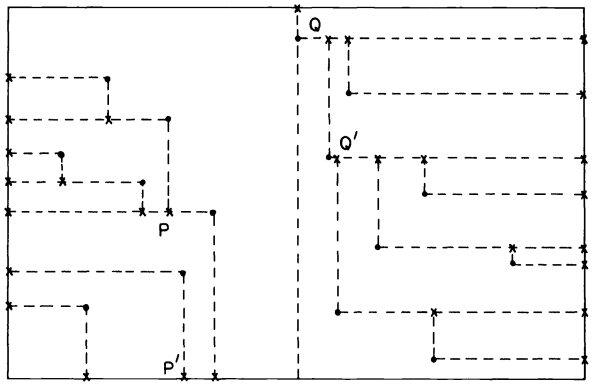


FIG. 3. *Newly created corner points.*

P in CL can only be paired with point Q in CR (but not Q'). Specifically, the following *pairing condition* must be satisfied: the corner point LC_i with left support p_l and bottom support p_b can only be paired with a point RC_j whose corresponding top support is *higher* than p_l and right support *higher* than p_b. Secondly, since original points are included in CL and CR, we should not use any of these points as corner points of the corner rectangle. However, as we will see in the following two lemmas, these problems will not arise as far as the computation of the LECR is concerned. Inclusion of the given points in CL and CR is to ensure that the corner rectangle thus determined contains no given points in its interior.

LEMMA 2. *The largest empty corner rectangle cannot use any of the given points as corner points. Furthermore, the corner points from CL and CR, respectively, satisfy the pairing condition prescribed above.*

Proof. Let the LECR be determined by LC_i and RC_j for some i and j. Suppose that LC_i is one of the given points in the left half. Since there exists in CL a point LC_k to the left of LC_i which is the corner point determined by some point p and LC_i, where $x(LC_k) = x(p)$ and $y(LC_k) = y(LC_i)$, and LC_k can be paired with RC_j to form a larger empty corner rectangle, we have a contradiction. On the other hand, if LC_i violates the pairing condition that the associated left support is higher than RC_j, then we can always find another corner point $LC_{k'}$ in CL with $y(LC_{k'}) = y(LC_i)$ that satisfies the pairing condition and can be paired with RC_j to form a larger empty corner rectangle, a contradiction. The case where RC_j is one of the given points in the right half or it violates the pairing condition that the associated right support is lower than LC_i can be handled in a similar way. This completes the proof.

Thus, the largest empty corner rectangle must use the newly created points as its corner points. We note that we only require that the corner rectangle contain none of the given points, so it may contain some of the newly created corner points in its interior. However, the following lemma rules out this possibility.

LEMMA 3. *If a corner rectangle does not contain any given point in its interior, then it must also not contain any newly created corner points.*

Proof. Suppose it contains a newly created left corner point LC in its interior. Let p_i and p_j be the two points in the left half that determine LC so that LC and p_j have the same y-coordinate. Since the corner rectangle is determined by a lower left corner and an upper right corner points that are in the left and right halves respectively, it must contain point p_j in its interior as well, a contradiction. The case where it contains a newly created right corner point in its interior can be handled similarly.

With the above two lemmas we can proceed to find a largest corner rectangle which is determined by a point in CL and a point in CR and which is "empty" in the sense that it does not contain any point (including those newly created corner points) in its interior.

4.2. Computing the largest empty corner rectangle. We first assume that the points in CL and in CR have been sorted in both x- and y-coordinates. Divide the sets CL and CR each into two subsets CL_1, CL_2 and CR_1, CR_2, respectively, with CL_1 above CL_2 and CR_1 above CR_2, using a horizontal line such that $CL_1 \cup CR_1$ is approximately of the same size as $CL_2 \cup CR_2$ (Fig. 4). Assume recursively that we have computed the LECR in $CL_1 \cup CR_1$ and in $CL_2 \cup CR_2$. So we may concentrate on the case where the lower left corner is in CL_2 and upper right corner is in CR_1. If $E(N)$ denotes the time complexity of the latter problem and $D(N)$ denotes that of the former problem, we have

$$D(N) \leqq 2D(N/2) + E(N). \tag{2}$$

Our first observation is that we may discard all the points of CR_1 that "dominate" any other. (A point p is said to dominate point q if both p's x- and y-coordinates are greater than those of q's; and a point is maximal if it is not dominated by any other point.) This is identical to keeping the maxima of the mirror image of CR_1 with respect to the origin, so it can be accomplished in linear time, since the points of CR_1 are sorted in x-order. For points with the same y-coordinate we further trim them by keeping only the rightmost one that does not dominate any other point in CR_1. Similarly, for points with the same x-coordinate we keep only the topmost point that does not dominate any other point in CR_1. See Fig. 4, in which points eliminated are marked

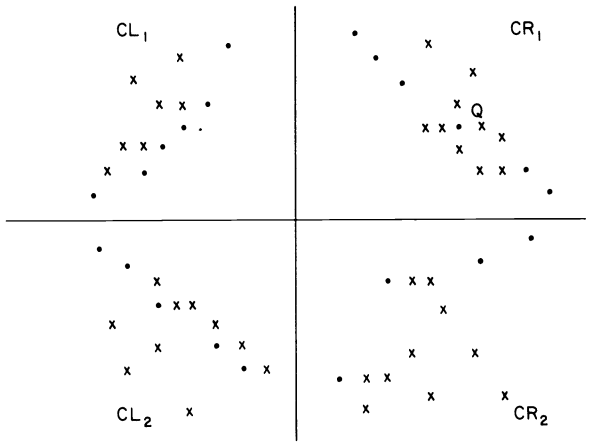


FIG. 4. *Subdivision of the points into four subsets and the trimming operation.*

as $\times$. The reason why they can be trimmed is that they cannot form the LECR with a point in CL_2. Similar remarks can be made about CL_2, and we can trim this set in a similar way. Finally, we can also apply this clean-up to CL_1 and CR_2. Note that this final clean-up removes from CL_1 the points that have the same y-coordinate except the rightmost one and removes from CR_2 the points that have the same y-coordinate except the leftmost one. This procedure can be accomplished in $O(N)$ time. In what follows we assume that these sets have been trimmed.

The next step is to determine, for each point in CL_2, the set of points in CR_1 with which the point can be paired. This set is clearly a contiguous subsequence of the

points $M_1, M_2, \cdots, M_u$ of CR_1 given in ascending x-order. We will therefore precompute the functions $l(P)$, and $r(P)$ such that the set $\{M_{l(P)}, M_{l(P)+1}, \cdots, M_{r(P)}\}$ contains exactly the points of CR_1 which can be paired with P to form an empty corner rectangle (Fig. 5). It is easy to precompute the function l (and by the same token, r) in linear time, proceeding as follows: assume that each set CL_1, CL_2, CR_1 and CR_2 has been sorted by x-coordinates. In a merge-like scan through CL_1 and CL_2, we compute, for each point P of CL_2, its "counterpart" in CL_1, i.e., the leftmost point of CL_1 to the right of the vertical line passing through P. We perform the same operation with respect to CL_1 and CR_1 (rotated by 90 degrees), and the conjunction of the two lists thus obtained precisely provides the desired correspondence between P and $M_{l(P)}$. We compute the function $r(P)$ similarly.

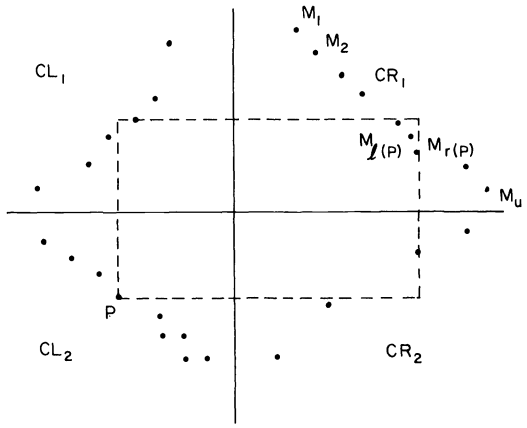


FIG. 5. *The points in CR_1 that can be paired with P in CL_2.*

Once the set $\{M_{l(P)}, M_{l(P)+1}, \cdots, M_{r(P)}\}$ associated with P is computed, in order to facilitate searching of a point M_i in the set with which to pair P to form the LECR we make use of the notion of so-called LL-diagram (Lower-Left-diagram), which is similar to the notion of Voronoi diagram (see, for example, [10], [11], [15]).

Computing the LL-diagram. The LL-diagram of a set $S = \{M_1, M_2, \cdots, M_N\}$, denoted $LL(S)$, is defined as follows: $LL(S)$ is a set of regions $\{V(M_1), V(M_2), \cdots, V(M_N)\}$, where

$$V(M_i) = \{M \in NE^* \mid d(M, M_i) \geqq d(M, M_j) \text{ for all } j = 1, 2, \cdots, N\}$$

and $d(A, B)$, the d-distance between A and B, measures the area of the corner rectangle between A and B if B dominates A and is zero otherwise; NE^* denotes the region $(-\infty, x_{\max}] \times (-\infty, y_{\max}]$ excluding the smallest enclosing rectangle of S, where $x_{\max}$ and $y_{\max}$ are maximum x- and y-coordinates of S respectively, and is the crossed-line area shown in Fig. 6. Note that if a point M of S is dominated by another, its associated region $V(M)$ is empty. Thus, we only consider the case where S contains only maxima. The LL-diagram of S has the following properties (Lemmas 4, 5 and 6).

LEMMA 4. *Let S be a set of N maxima, $M_1, M_2, \cdots, M_N$. Then $LL(S)$ consists of a set of possibly unbounded polygons which partitions NE^*. All the polygons (except one) are convex, and $LL(S)$ involves only $O(N)$ edges.*

Proof. First consider the case where S consists of only two points $A(0, v)$ and $B(u, 0)$ with u and v positive. The points $M(x, y)$ in NE^* farther from A than from B (with respect to d) satisfy the relation $x(v-y) \leqq y(u-x)$. This reduces to $y \geqq 0$ or $vx - uy \leqq 0$, which is the area of NE^* above the line passing by the diagonal (other

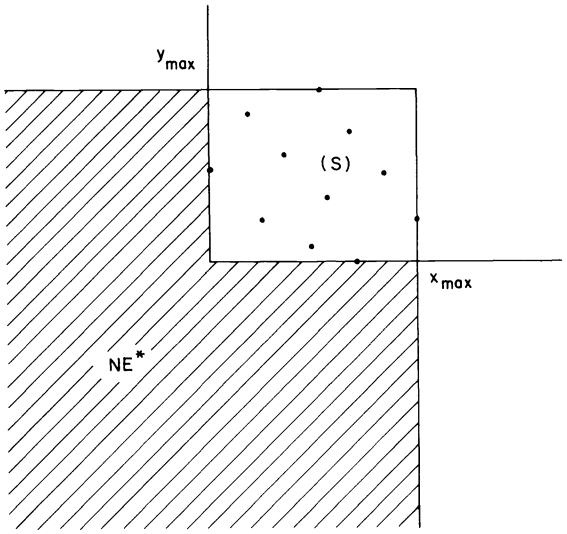


FIG. 6. *Domain of definition of LL-diagrams.*

than AB) of the corner rectangle determined by A and B (Fig. 7). In general, consider the intersection W of the regions associated with M_i in $LL(M_k, M_i)$, $k = 1, 2, \cdots, N$. Since W is the intersection of unbounded convex polygons, it is itself a possibly unbounded convex polygon. Also, since the domain of definition, NE^*, as defined for each $LL(M_k, M_i)$, contains the domain of definition for $LL(S)$, the region $V(M_i)$ is simply the intersection of W with NE^*; it is therefore a possibly unbounded polygon, which is always convex, except for the polygon which contains the "corner" of NE^*. We can also see that $V(M_i)$ has an edge on the boundary of NE^*. (Note that none of the regions associated with M_i in $LL(M_k, M_i)$ lies strictly inside NE^*.) The last point to make is that since $LL(S)$ is a planar graph with $O(N)$ faces, all of whose vertices have degree $\geqq 3$, it involves $O(N)$ vertices.

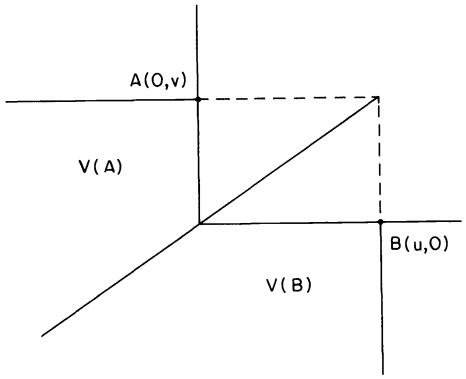


FIG. 7. *LL-diagram for points A and B.*

LEMMA 5. *Let $M_1, M_2, \cdots, M_N$ occur in this order with ascending x-coordinate and let L be any line parallel to the x-axis. It is impossible to find two points A and B on L in NE^* with increasing coordinates such that A and B lie in $V(M_i)$ and $V(M_j)$, respectively, with $i > j$* (Fig. 8).

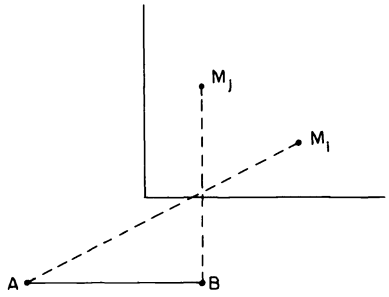


FIG. 8. *Point A cannot be in $V(M_i)$ and point B in $V(M_j)$.*

Proof. Immediate from the definition of $LL(M_i, M_j)$.

We now proceed with the computation of $LL(S)$ using a standard divide-and-conquer technique. We sort the points in S in ascending x-coordinates as $M_1, M_2, \cdots, M_N$. Recursively compute $L_1 = LL(S_1)$ and $L_2 = LL(S_2)$, where $S_1 = \{M_1, M_2, \cdots, M_{\lfloor N/2 \rfloor}\}$ and $S_2 = \{M_{\lfloor N/2 \rfloor + 1}, \cdots, M_N\}$, and then merge them. We start at the lower left corner of the corner rectangle determined by $M_{\lfloor N/2 \rfloor}$ and $M_{\lfloor N/2 \rfloor + 1}$ drawing the diagonal AB (Fig. 9a) downwards until we hit an edge of L_1 or L_2, at which point we *stitch* the two segments. We illustrate the stitching operation in Fig. 9b. Let us call a *diagonal* of a segment $\overline{M_iM_j}$, the line supporting the diagonal of the corner rectangle determined by M_i and M_j (other than $\overline{M_iM_j}$). Suppose that the line currently drawn follows the diagonal of $\overline{M_iM_j}$ downwards. When we encounter the diagonal of $\overline{M_jM_k}$, we cut through it and replace it by the diagonal of $\overline{M_iM_k}$ (Fig. 9b). By doing so, we recompute the LL-diagram of S locally around the path being thus drawn. If we iterate on this process i.e., drawing, hitting and stitching, we will produce a path $\mathbf{z}$, which is obviously monotone with respect to both the x and y axes. This is

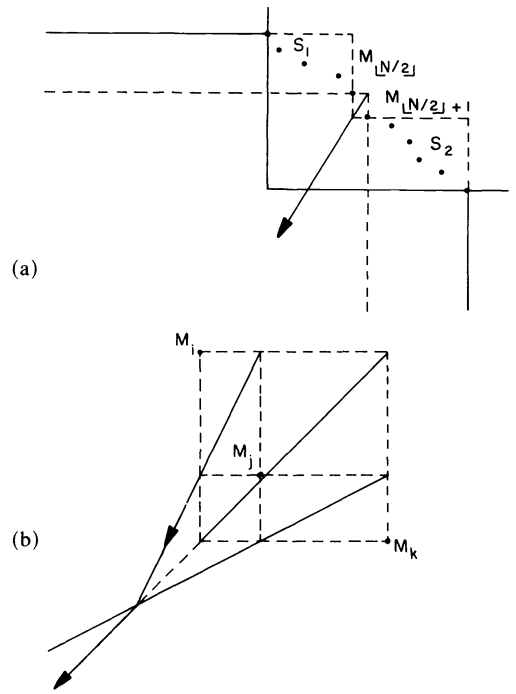


FIG. 9. *Stitching operation of two LL-diagrams.*

due to the fact that z is made of chunks of diagonal, which all have positive slopes. To ensure that stitching the two *LL*-diagrams L_1 and L_2 takes $O(N)$ time, we do the following. At all times, we keep track of the face in L_1 and the face in L_2 we are currently in. The stitching operation corresponds to leaving one face of, say L_1, for another face of L_1. At this point, we know the direction to follow, and we must compute the next hitting edge. To do so, we maintain a pointer p_1 (respectively p_2) to go around the current face of L_1 counterclockwise (respectively L_2 clockwise). Since all slopes are positive, pointers will always be descending; therefore we can move them in a simple round-robin fashion so as to detect the first intersection with the new drawing direction without ever backtracking. As usual, we remove all parts of L_1 and L_2 which have been cut and lie to the right and left, respectively, of the path z. We omit the details and conclude that the entire computation of $LL(S)$ takes $O(N \log N)$ time. Interested readers are referred to, for example, [10], [11], [15] for details of the merge concept.

LEMMA 6. *The LL-diagram of a set of N points can be computed in* $O(N \log N)$ *time.*

Proof. It suffices to show that in the recursive step of the above procedure each point to the left and to the right, respectively, of z has its farthest neighbor in S_1 and in S_2. Assume that this is not the case, and that, for example, a point M to the left of z has its farthest neighbor in S_2. Let us draw a horizontal line through M. This line will necessarily intersect the path z in exactly one point P. Let P^* be a point immediately to the left of z (Fig. 10). P is, by construction, in the region of a point of S_1, since we have already seen that the *LL*-diagram has been correctly constructed around z locally. This is a contradiction to Lemma 5, which completes the proof.

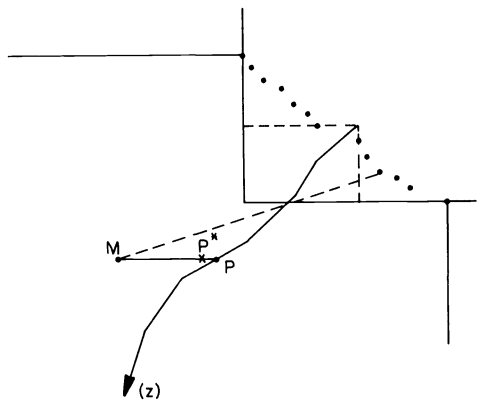


FIG. 10. *Illustration for the proof of Lemma* 6.

With the *LL*-diagrams in mind let us consider how they can be effectively used in finding the point of CR_1 to pair with each point in CL_1. Consider the complete binary tree T that has $M_1, M_2, \cdots, M_u$ of CR_1 for leaves, from left to right. Letting SL be the set of sequences of leaves of each subtree of T, we can use the standard segment-tree technique [2] to rewrite any contiguous subsequences of $M_1, M_2, \cdots, M_u$ as the concatenation of $O(\log u)$ sequences in SL (Fig. 11). It is clear that for any interval $[M_i, M_j]$ of consecutive leaves such decomposition can be obtained in $O(\log u)$ time by a simple search for M_i and M_j in T. We omit the details (see Bentley and Wood [2]). The purpose of this decomposition is to compute efficiently the farthest neighbor in $[M_{l(P)}, M_{r(P)}]$ of each point P in CL_2. To do so, we precompute the *LL*-diagram of each sequence of points in SL so that we may decompose $[M_{l(P)}, M_{r(P)}]$

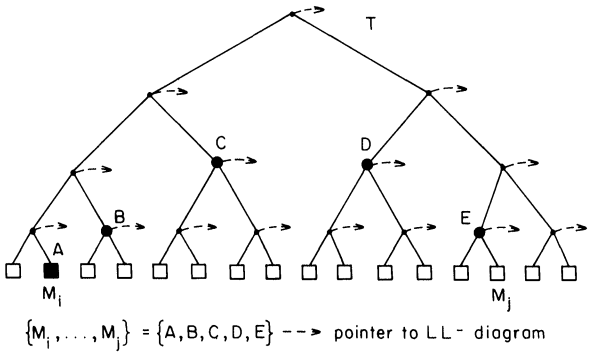


FIG. 11. *Decomposition of an interval* $[M_i, M_j]$ *into subintervals.*

into sequences in SL and search for the farthest neighbor of P in each of the sequences. A similar technique can be found in Gowda et al. [6]. Searching for the farthest neighbor is, in this case, equivalent to determining in which region of the LL-diagram P lies. This can be accomplished in logarithmic time, using only linear space with Kirkpatrick's planar point location algorithm [8].

To summarize, the preprocessing of the recursive step consists of:

1. Trimming CL_1, CL_2, CR_1 and CR_2 in $O(N)$ time and space.
2. Precomputing the functions l, and r in $O(N)$ time snd space.
3. Setting up the tree T for CR_1, and computing the LL-diagram of each sequence of points in SL. SL consists of one u-sequence, two $u/2$-sequences, $\cdots$, 2^k $u/2^k$-sequences, etc. It follows that computing all the LL-diagrams requires time $T(u) = 2T(u/2) + O(u) = O(u \log u)$ and space $S(u) = 2S(u/2) + O(u) = O(u \log u)$. (See Lemma 6 or [6].)
4. Setting up the preprocessing required by Kirkpatrick's planar point location algorithm for each LL-diagram computed, which requires $O(k)$ time and space for a set of k points.

Consequently, the total cost of the preprocessing amounts to $O(u \log u)$ time and space.

The computation of the farthest neighbor in $\{M_{l(P)}, \cdots, M_{r(P)}\}$ of each point P in CL_2 is done by performing $O(\log u)$ planar point searches, each requiring $O(\log u)$ time. Putting our results together, we conclude that it is possible to find the LECR with corners in CR_1 and CL_2 in time $E(N) = O(N \log^2 N)$ and space $O(N \log N)$. From the relations (1) and (2) we therefore have the following.

LEMMA 7. *The largest empty corner rectangle with corner points in CL and CR can be computed in* $D(N) = O(N \log^3 N)$ *time and* $O(N \log N)$ *space.*

THEOREM 1. *The largest empty rectangle problem for N points in the plane can be solved in* $T(N) = O(N \log^4 N)$ *time and* $O(N \log N)$ *space.*

4.3. An improved algorithm for computing LECR. The result of the previous section can still be improved using a less redundant representation. The redundancy comes partly from the horizontal recursion, since it is likely to entail repeated computations of the same LL-diagrams. Instead, we will set up a global data structure for the entire right half, namely the set CR.

Let $M_1, \cdots, M_u$ be the points of CR. We will arrange the points to be the nodes of a rooted tree T_{CR} that is constructed as follows. The root is an imaginary point situated entirely above and to the left of CR. First of all, the points with the same y-coordinates are connected to form chains, ordered in y-coordinate. Then for each chain we connect the leftmost point to the point in a higher chain that is directly above

it (Fig. 12). We note that if P is an arbitrary point in CL and M_i is the lowest (rightmost) point of CR higher than P, the path from M_i to the root of the tree contains the only points which can be paired with P to form an LECR (Fig. 12).

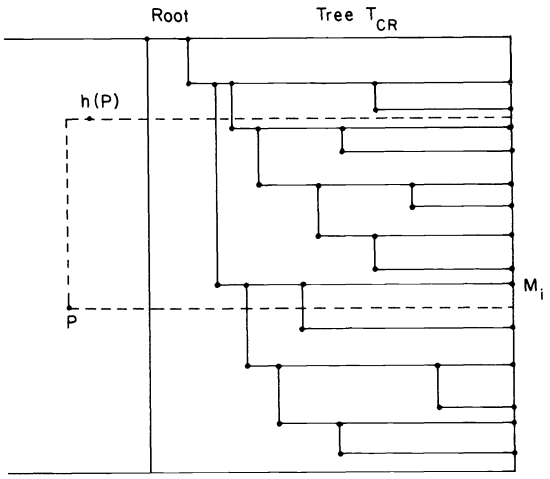


FIG. 12. *Construction of the tree* T_{CR}.

By analogy with the previous section, we will precompute for each point in CL the functions l and r, which will now point to nodes in T_{CR}. Notice that $M_{r(P)}$ is simply the lowest (rightmost) point of CR higher than P, and can be computed in linear time for each P in CL. Similarly we can compute the "vertical" obstacle $h(P) = \text{upper}(P)$, in CL for each point P in linear time. Next we precompute the function l as follows: consider each point P in descending y-coordinate and 1) if P is in CR, ensure that all the points and only the points on the path of T_{CR} from the root to P are arranged consecutively in a stack A, 2) if P is in CL, do a binary search in A in order to find the highest point below $h(P)$. Note that this point is exactly $M_{l(P)}$. Since T_{CR} is a tree, operation 1 consists simply of updating a stack, which will take a total of $O(N)$ time. Of course, each point of CL may cause an $O(\log N)$ search time. Therefore the total time complexity of the procedure is $O(N \log N)$.

Computing path functions in a free tree. Let T be a free tree with N vertices, each of degree at most 3. Recall that a free tree is a connected acyclic graph. We wish to compute "decomposable" functions over tree-paths very efficiently. We consider functions of the form $F(v, w)$, defined for any pair of vertices (v, w) of T. These functions are assumed to have the following property: there exists an associative operator OP computable in constant time, such that for any vertex z on the path from v to w, we have

$$F(v, w) = F(w, v) = OP(F(v, z), F(z, w)).$$

One trivial example of such a function is the distance between two nodes of the tree. For the application at hand, (v, w) will be a pair of the form $(l(P), r(P))$ and $F(v, w)$ will be the maximum d-distance between P and any point on the chain from $M_{l(P)}$ to $M_{r(P)}$.

Suppose now that in order to compute $F(v, w)$ we can use a data structure $L(v, w)$ so that $F(v, w)$ can then be evaluated in $O(f(N))$ time. We assume that $L(v, w)$ requires $O(t)$ space and can be computed in $O(t \log t)$ time, where t is the number of nodes on the path between v and w. Instead of precomputing all possible structures

$L(v, w)$ for all pairs of nodes (v, w), which would require $O(N^3)$ storage, we will compute only a subset of them $R = L(v_i, w_i)$, which takes only $O(N \log N)$ space, and has the property that the path between any pair of nodes in T is the concatenation of disjoint paths between $O(\log N)$ pairs (v_i, w_i). The availability of R clearly allows us to evaluate $F(v, w)$ in $O(f(N) \log N)$ time, assuming that we can express the path (v, w) as a sequence in R in $O(\log N)$ time.

The construction of R relies on the fact that it is possible in linear time to find a vertex (the centroid) of an N-vertex tree whose removal from the tree leaves subtrees with at most $N/2$ nodes [9]. We will compute R recursively. To do so, we will represent T, as a rooted ternary tree G defined as follows: let C, the centroid of T be the root of G. For the sake of simplicity we may assume without loss of generality that we have exactly 3 subtrees, T_1, T_2, T_3, rooted at C. We then proceed to compute their centroids, C_1, C_2, C_3, which we insert in G as the sons of the root C. We iterate on this process for each subtree, labeling the nodes in the following manner. Assume that T_i has exactly N_i vertices ($N_1 + N_2 + N_3 = N - 1$). The root C is labeled N and the general rule is that T_1 will be labeled recursively with the integers $\{1, \cdots, N_1\}$, T_2 with $\{N_1 + 1, \cdots, N_1 + N_2\}$, and T_3 with $\{N_1 + N_2 + 1, \cdots, N - 1\}$. To be consistent we will label the root of each subtree with the highest label available.

Our next task is to augment G with new edges, called *extra-edges*. For each vertex v of G we, in turn, apply the following procedure to all the vertices which are adjacent to v in T and are labeled lower than v in G. Let w be such a vertex. Since it clearly must lie in the subtree of G rooted at v, we link to v every node, including w, on the path in G from v to w, thus adding the so-called extra-edges. Figure 13a shows the labeling of the tree and Fig. 13b the ternary tree representation G. The dotted lines in Fig. 13b indicate the extra-edges that are introduced when the root node (labeled 26) is considered.

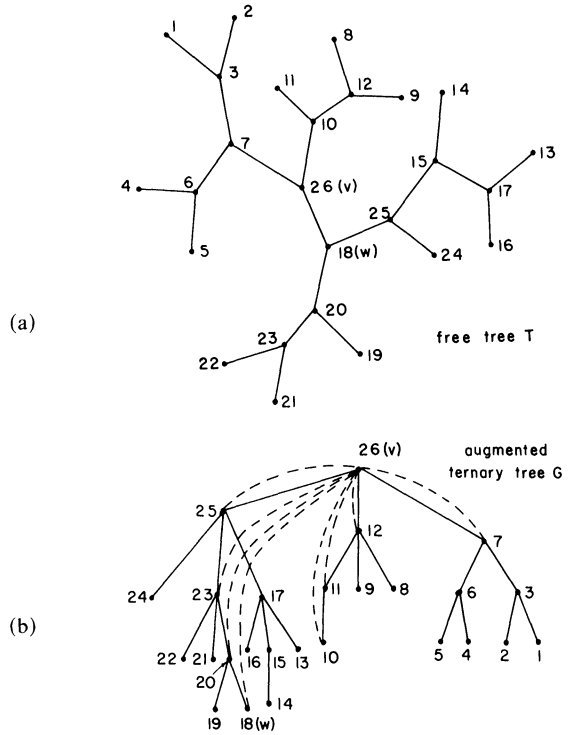


FIG. 13. *Converting a free tree to an augmented ternary tree.*

Since we can compute centroids in linear time and G clearly has height $O(\log N)$, it is easy to construct the tree augmented with extra-edges in time $\text{Max}(O(N \log N), H(N))$, where the first term accounts for the time needed for augmenting the extra-edges, and $H(N)$, the time for constructing G, is given by $H(N) = H(N_1) + H(N_2) + H(N_3) + O(N)$, $N_1 + N_2 + N_3 = N - 1$, and $N_1, N_2, N_3 \leqq N/2$, i.e., $H(N) = O(N \log N)$. The final addition to G is to set pointers from each edge (u, v) of G to the structure $L(u, v)$.

It is not difficult to evaluate the overall time and space complexities, $T(N)$ and $S(N)$, respectively, of the preprocessing we have just described. Since an edge in G from the root to a vertex k levels deeper gives rise to a path in T of length at most $N/2^k$, the time and space complexities, respectively, for computing all the structures of the form $L(\text{root}, x)$ required by our algorithm is $O(N \log N)$ and $O(N)$. (Recall that the structure $L(v, w)$ is assumed to take $O(t)$ space and $O(t \log t)$ time to construct with t being the number of nodes on the path in T from v to w.) Thus,

$$T(N) = T(N_1) + T(N_2) + T(N_3) + O(N \log N)$$

and

$$S(N) = S(N_1) + S(N_2) + S(N_3) + O(N)$$

with $N_1 + N_2 + N_3 = N - 1$, and $N_1, N_2, N_3 \leqq N/2$. This gives rise to the following complexity bounds for computing the preprocessing structure R: $T(N) = O(N \log^2 N)$ and $S(N) = O(N \log N)$.

We can now show that this preprocessing allows us to evaluate $F(u, v)$ for any pair (u, v) of nodes in T in time $O(f(N) \log N)$. To do so, we walk up the path in G from u towards the root, stopping at the first node w with a label exceeding that of v. We know that v lies in one of the subtrees of w. Note also that the labeling of G allows us to go down from w to v in time proportional to the length of the path. At this point we perform the same operations for u and v in turn, so we may describe the procedure for u only. Let $(w, u_1, \cdots, u_k)$ $(u_k = u)$ be the path in G from w to u. If this path has extra-edges connecting w to some u_i, we note that G must have a set of consecutive extra-edges $wu_1, wu_2, \cdots, wu_j$ between w and u. We collect the last extra-edges wu_j and iterate on this process, restarting at u_j (Fig. 14). If w does not have an extra-edge on its path to u, we simply collect the edges wu_1 and iterate from u_1. Note that this *collecting* operation takes only $O(\text{length of path from } w \text{ to } u) = O(\log N)$. Finally we compute $F(u, v)$ by evaluating $OP(\cdots, F(u_i, v_i), \cdots)$, where the (u_i, v_i) are all the edges collected by the above procedure.

To show the correctness of our method it suffices to notice that the path formed by the edges collected from u to w, along with the path collected similarly from w to v, gives an exact partition of the path in T from u to v. There again, the reader can supply an easy proof of the fact. We conclude as follows.

LEMMA 8. *With $O(N \log N)$-space, $O(N \log^2 N)$-time preprocessing, it is possible to evaluate $F(u, v)$ in $O(f(N) \log N)$ time for any pair of vertices (u, v) in T.*

Note that the idea of decomposing tree-paths into canonical pieces is one aspect of a general mapping principle between linear lists and trees developed in [4].

Computing the LECR *efficiently.* A simple application of the previous paragraph provides an improved algorithm for computing the LECR of the sets $S = CL \cup CR$. Clearly T is our tree T_{CR}, the structure $L(u, v)$ is the LL-diagram of the points of CR on the path between u and v, preprocessed so as to allow for Kirkpatrick's planar point location algorithm, the function $F(u, v)$ simply returns the regions in $L(u, v)$ where a given point P of CL lies; its complexity is therefore $f(N) = O(\log N)$. Finally,

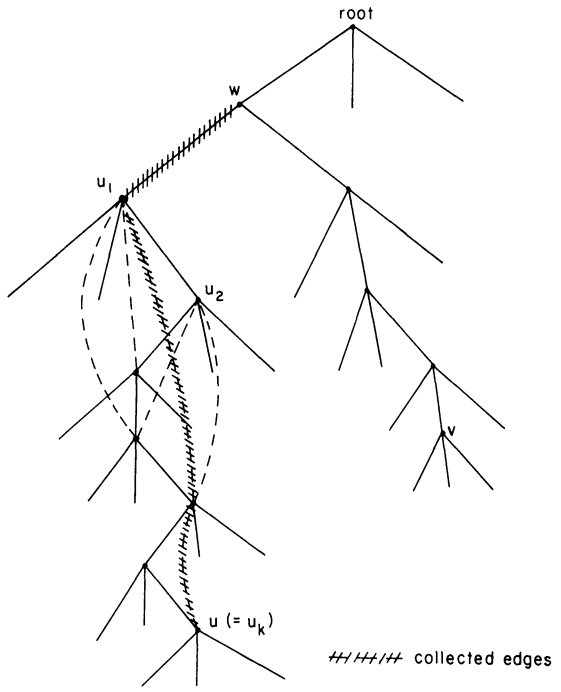


FIG. 14. *Collecting operation of a path from w to u.*

the operator $OP(M, M')$ returns the point (M or M') that forms the LECR with P. Note that the only discrepancy comes from the fact that T_{CR} is not necessarily a tree with degree $\leqq 3$. There is an easy fix, however. We simply introduce dummy vertices to reduce any excessive degree to 3. This adds only $O(N)$ vertices and thus does not affect the complexity of the algorithm. We are now in a position to compute the LECR in S. To do so, compute $F(M_{l(P)}, M_{r(P)})$ for all P in CL, and return the point which gives the largest area along with its upper right neighbor. We omit the details of this straightforward transformation. Thus, with Lemma 8 and the above discussion we have the following.

LEMMA 9. *It is possible to compute the* LECR *determined by a pair of points, each of which is in one of the sets CL and CR respectively in* $D(N) = O(N \log^2 N)$ *time and in* $O(N \log N)$ *space.*

Using relation (1) we can state our main result.

THEOREM 2. *The largest empty rectangle problem for a set of N points can be computed in* $O(N \log^3 N)$ *time and* $O(N \log N)$ *space.*

5. Conclusion. We have presented an $O(N \log^3 N)$-time and $O(N \log N)$ space algorithm for computing the largest area rectangle which contains none of the N points in its interior. A simpler version of the algorithm with running time $O(N \log^4 N)$ and space $O(N \log N)$ has also been given. The algorithms are primarily based on the divide-and-conquer strategy. We have addressed only the problem of locating a rectangle whose sides are parallel to those of the bounding rectangle of the given set of points. If the rectangle sought is arbitrarily oriented, the problem becomes much more difficult.

Naturally one may ask for an arbitrary polygon instead of a rectangle within a bounded region or generalize the problem to higher dimensions. We remark that if

the largest empty triangle is sought and the directions of the sides of the triangle have been predetermined, then the largest empty triangle can be found in $O(n \log n)$ time and $O(n)$ space using a divide-and-conquer techique similar to the one used in § 3.

REFERENCES

[1] J. L. BENTLEY, *Divide-and-conquer algorithms for closest point problems in multidimensional space*, Ph.D. thesis, Dept. Computer Science, Univ. North Carolina, Chapel Hill, NC, 1976.

[2] J. L. BENTLEY AND D. WOOD, *An optimal worst case algorithm for reporting intersections of rectangles*, IEEE Trans. Comput. (1980), pp. 571-577.

[3] J. E. BOYCE, D. P. DOBKIN, R. L. DRYSDALE III AND L. J. GUIBAS, *Finding extremal polygons*, Proc. ACM Symposium on Theory of Computing, 1982, pp. 282-289.

[4] B. M. CHAZELLE, *Computing on a free tree via complexity-preserving mappings*, Proc. 25th IEEE Symposium on Foundations of Computer Science, 1984, to appear.

[5] D. P. DOBKIN, R. L. DRYSDALE III AND L. J. GUIBAS, *Finding smallest polygons*, to appear in Advances of Computing Research, F. P. Preparata, ed., Jai Press.

[6] I. G. GOWDA, D. G. KIRKPATRICK, D. T. LEE AND A. NAAMAD, *Dynamic Voronoi diagrams*, IEEE Trans. Inform. Theory, IT-29 (1983), pp. 724-731.

[7] F. K. HWANG, *An $O(n \log n)$ algorithm for rectilinear minimal spanning trees*, J. ACM, 26 (1979), pp. 177-182.

[8] D. G. KIRKPATRICK, *Optimal search in planar subdivisions*, this Journal, 12 (Feb. 1983), pp. 28-35.

[9] D. E. KNUTH, *The Art of Computer Programming, Vol.* I: *Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.

[10] D. T. LEE, *Two dimensional Voronoi diagrams in the L_p-metric*, J. ACM, 27 (1980), pp. 604-618.

[11] D. T. LEE AND R. L. DRYSDALE III, *Generalization of Voronoi diagrams in the plane*, this Journal, 10 (1981), pp. 73-87.

[12] D. T. LEE AND C. K. WONG, *Voronoi diagrams in L_1-(L_∞-)metrics with 2-dimensional storage applications*, this Journal, 9 (1980), pp. 200-211.

[13] A. NAAMAD, W. L. HSU AND D. T. LEE, *On maximum empty rectangle problem*, Appl. Disc. Math., 8 (1984), pp. 267-277.

[14] M. I. SHAMOS, *Computational geometry*, Ph.D. dissertation, Dept. Computer Sciences, Yale Univ., New Haven, CT, 1978.

[15] M. I. SHAMOS AND D. HOEY, *Closest-point problem*, Proc. 16th IEEE Symposium on Foundations of Computer Science, 1975, pp. 151-162.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics

OPTIMAL POINT LOCATION IN A MONOTONE SUBDIVISION*

HERBERT EDELSBRUNNER†, LEONIDAS J. GUIBAS‡ AND JORGE STOLFI§

Abstract. Point location, often known in graphics as "hit detection," is one of the fundamental problems of computational geometry. In a point location query we want to identify which of a given collection of geometric objects contains a particular point. Let $\mathcal{S}$ denote a subdivision of the Euclidean plane into monotone regions by a straight-line graph of m edges. In this paper we exhibit a substantial refinement of the technique of Lee and Preparata [SIAM J. Comput., 6 (1977), pp. 594–606] for locating a point in $\mathcal{S}$ based on separating chains. The new data structure, called a *layered dag*, can be built in $O(m)$ time, uses $O(m)$ storage, and makes possible point location in $O(\log m)$ time. Unlike previous structures that attain these optimal bounds, the layered dag can be implemented in a simple and practical way, and is extensible to subdivisions with edges more general than straight-line segments.

Key words. point location, monotone polygons, planar graphs, partial order, graph traversal, computational geometry

1. Introduction. *Point location* is one of the fundamental problems in computational geometry. In the two-dimensional case, we are given a subdivision $\mathcal{S}$ of the plane into two or more regions, and then asked to determine which of those regions contains a given query point p. If the same subdivision is to be used for a large number of queries, as is often the case, we can reduce the total cost of this task by preprocessing the subdivision into a data structure suitable for the search. We will measure the performance of a proposed solution to this problem by three quantities, the preprocessing cost P, the storage cost S, and the query cost Q.

Optimal solutions for this search problem have been known since Lipton and Tarjan [LT] and Kirkpatrick [Ki]. For a subdivision $\mathcal{S}$ with m edges these optimal solutions simultaneously attain $S = O(m)$, $Q = O(\log m)$ and, under certain assumptions, also $P = O(m)$. The Lipton–Tarjan method is based on their graph separator theorem, and so far has been only of theoretical interest. Kirkpatrick's method, which consists of building a hierarchy of coarser and coarser subdivisions, is implementable, but still the implied constants are so large as to make current implementations of little practical interest. In addition, neither of these techniques seems to extend in a natural way to curved-edge subdivisions.

Historically, Dobkin and Lipton [DL] were the first to achieve $O(\log m)$ query time, while using $O(m^2)$ space. Their method was subsequently refined by Preparata [P] so that $O(m \log m)$ space suffices. Later Bilardi and Preparata [BP] again gave a refinement that achieves $O(m)$ space in the expected case, while retaining $O(m \log m)$ space and $O(\log m)$ query time in the worst case. These solutions are applicable to curved-edge subdivisions and seem to admit of efficient implementations.

A substantially different approach was taken by Shamos [S] and led to the well-known point location paper of Lee and Preparata [LP], based on the construction

* Received by the editors October 13, 1983, and in final revised form January 28, 1985.

† Institutes for Information Processing, Technical University of Graz, A-8010 Graz, Austria. The work of this author was supported by the Austrian Fonds zur Förderung der wissenschaftlichen Forschung.

‡ Digital Equipment Corporation, Systems Research Center, Palo Alto, California 94301.

§ Digital Equipment Corporation Systems, Research Center, on leave from the University of São Paulo, São Paulo, Brazil. Part of this research was conducted while this author was at the Xerox Palo Alto Research Center, Palo Alto, California. The work of this author was supported in part by a grant from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

of separating chains. This data structure attains $P = O(m \log m)$, $S = O(m)$, and $Q = O(\log^2 m)$. The constants of proportionality in these expressions are quite small, and the algorithms, particularly the query one, are simpler than those of Kirkpatrick. While Kirkpatrick's algorithm requires the regions to be triangular, that of Lee and Preparata works for regions of a more general shape (monotone polygons, which include the convex ones). Recently Chazelle [C2] described a variant of this solution as a special instance of a general method for "searching in history". His structure needs the same amount of space and time. These techniques again are applicable to curved-edge subdivisions, although this possibility was not examined.

In a separate development, Edelsbrunner and Maurer [EM] came up with a space-optimal solution that works for general subdivisions, and even for sets of arbitrary nonoverlapping regions. The query time is $Q = O(\log^3 m)$, but it can be improved to $Q = O(\log^2 m)$ for rectangular subdivisions, where the structure becomes especially simple. For this reason a generalization to rectangular point location in higher dimensions has also succeeded; see Edelsbrunner, Haring and Hilbert [EH].

The purpose of this paper is to show an elegant modification to the separating chain method of Lee and Preparata that yields a new optimal point location algorithm for monotone subdivisions of the plane. The algorithm is based on a new data structure called the *layered dag*. In this new structure the separating chains built into a binary tree by Lee and Preparata are refined so that (1) once a point has been discriminated against a chain, it can be discriminated against a child of that chain with constant extra effort, and (2) the overall storage only doubles. The layered dag simultaneously attains $S = O(m)$ and $Q = O(\log m)$.[1] An additional insight allows us to build this dag (as well as the original Lee-Preparata tree) in only $O(m)$ time. Not only is this new structure optimal with respect to all of preprocessing, space, and query costs, but in fact a simple implementation, with small constants of proportionality, is possible. Like its Lee-Preparata predecessor, it also extends to curved-edge subdivisions.

In the organization of the paper we have adopted the policy that each new data structure is first introduced by how it is to be used, and then by how it is to be constructed. We have placed emphasis throughout on implementation considerations, as well as the underlying theory. Section 2 describes the basic notions surrounding monotone polygons and subdivisions. Section 3 shows how nonmonotone subdivisions can be made monotone. Sections 4, 5 and 6 introduce a partial ordering of the regions and its use in getting a complete family of separators. Section 7 reviews the Lee-Preparata structure, while §§ 8 and 9 introduce the layered dag, and explain its use in point location and its construction. Section 10 gives an algorithm for constructing a complete family of separators based on a traversal of the subdivision. Two implementation issues are taken up in §§ 11 and 12; these may be omitted on a first reading. Section 11 describes some bit-twiddling trickery used to give us linear preprocessing time, while § 12 discusses how subdivisions can be traversed without auxiliary storage. Finally § 13 contains some further applications of the layered dag to problems in computational geometry.

2. Monotone polygons and subdivisions. An *interval* is a convex subset of a straight line, i.e., the whole line, a ray, a segment, a single point, or the empty set. An interval is *proper* if it contains more than one point, and is *open* if it does not contain its endpoints (if any). A subset of the plane is said to be *monotone* if its intersection with any line parallel to the y axis is a single interval (possibly empty).

[1] This refinement is similar to a technique proposed by Cole [C] and, in a different context, by Chazelle [C3] (the hive graph).

We define a *polygon* as an open, connected, and simply connected subset of the plane whose boundary can be partitioned into finitely many points (*vertices*) and open intervals (*edges*). Note that, according to these definitions, polygons may have infinite extent. A *subdivision* is a partition of the plane into a finite number of disjoint polygons (*regions*), edges, and vertices; these parts are collectively called the *elements* of the subdivision. It follows from the definitions that every edge is on the boundary between two regions (not necessarily distinct), that every vertex is incident to some edge, that every endpoint of an edge is a vertex, and that every region (unless there is only one) has some edge on its boundary. From these facts and from Euler's theorem, we can conclude that a subdivision with m edges has $O(m)$ vertices and regions, thus justifying our use of m as the measure of problem size.

A subdivision is said to be *monotone* if all its regions are monotone and it has no vertical edges. The last condition is a technical one: it is imposed only to simplify the proofs and algorithms and can be removed with some care. Figure 1 illustrates a monotone subdivision (arrowheads denote edges extending to infinity).

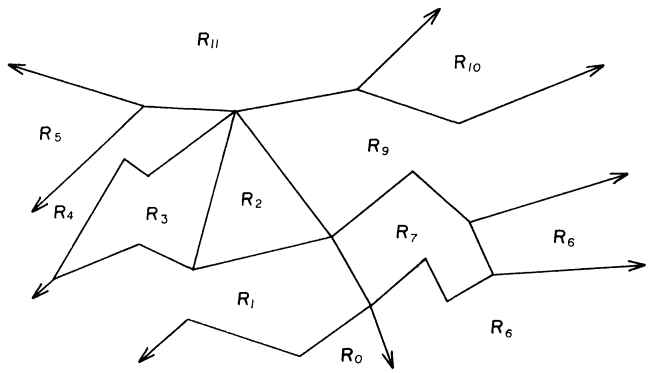


FIG. 1. *A monotone subdivision.*

With minor caveats and modifications, monotone subdivisions include many interesting subcases, such as triangulations, subdivisions of the plane into convex pieces, and so forth. The lemma below shows that monotone subdivisions are precisely the "regular planar straight-line graphs" as defined by Lee and Preparata [LP]:

LEMMA 1. *A Subdivision with no vertical edges is monotone if and only if every vertex is incident to at least two distinct edges, one at its left and one at its right.*

Proof. The "only if" part is easy, since if, for example, all edges incident to a vertex v pointed to the right, then the region to the left of v would have a disconnected intersection with a vertical line through v.

For the converse, assume R is a region that is not monotone, and let l be a vertical line whose intersection with R consists of two or more components, as in Fig. 2. Since R is connected, any two components I_1, I_2 of the intersection are connected by a simple path π in R. We can always choose I_1, I_2, and π so that I_1 is above I_2, and π does not meet l except at its endpoints $p_1 \in I_1$ and $p_2 \in I_2$. Then π and the interval $[p_1, p_2]$ delimit a closed and bounded region S of the plane.

Suppose π lies to the left of l. The boundary of R must enter S at the lower endpoint of I_1 (and also at the upper endpoint of I_2). Since the boundary cannot meet the path π, there must be some vertex of the subdivision in the interior of S. Let v be the *leftmost* such vertex; clearly all edges incident to v lie to the right of it, and we have proved the lemma. A similar argument holds if π lies to the right of l. □

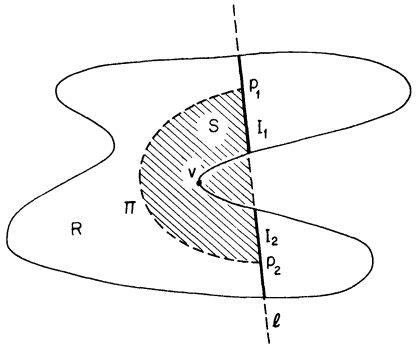


FIG. 2

Therefore, we can check whether any subdivision is monotone by enumerating all the edges incident to each vertex, and checking whether they satisfy Lemma 1. Each edge will be examined at most twice, and each vertex once, so this algorithm runs on $O(m)$ time.

3. Regularization. Lee and Preparata [LP] have shown that an arbitrary subdivision with m edges can be refined into a monotone subdivision having $\leqq 2m$ edges in $O(m \log m)$ time, a process they termed *regularization.* (They define a vertex as being *regular* if it satisfies the condition of Lemma 1.) For the sake of completeness we reproduce their algorithm here, in a slightly different setting.

The task of the regularization process is to add new edges to the subdivision so as to make each vertex regular. Each new edge must connect two existing vertices (or extend from an existing vertex to infinity), and may not intersect any other edges. In other words, we can connect two vertices only if they are *visible* from each other. See Fig. 3 for an example.

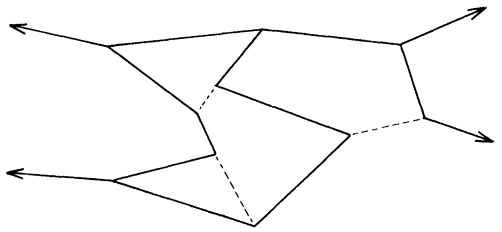


FIG. 3. *Regularizing a nonmonotone subdivision.*

The regularization algorithm is based on the *sweeping line* paradigm that has been used extensively in computational geometry. We imagine that a vertical line sweeps the plane from left to right. We call *active* those vertices, edges, and regions currently intersecting the sweeping line. The list of active elements changes only when the line passes through a vertex, at which time some edges may end and others may start. Except at those moments, the active edges have an obvious vertical ordering, and cut the sweeping line into one or more *active intervals.* To each active interval we assign a *generator,* which is the last vertex swept over that happened to lie on or between the two active edges bounding that interval. This vertex may be a left endpoint of those edges, as Fig. 4 shows, or it may be an irregular vertex with no right-going edge, as Fig. 5 shows.

LEMMA 2. *The generator of an active interval is visible from any point in that interval.*

Proof. By definition, the hatched region in Figs. 4 and 5 is free of vertices and edges. □

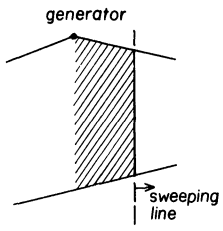


FIG. 4

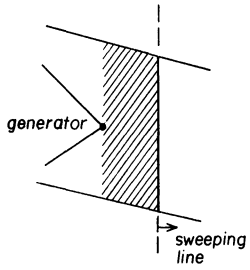


FIG. 5

The regularization algorithm simulates this line sweep. We start by sorting all vertices of the subdivision by their x-coordinates. We will assume these coordinates are all distinct; if necessary, we can enforce this condition by rotating the subdivision through a sufficiently small angle. We also augment the subdivision with two dummy vertices at $x = \pm\infty$, to which are incident all edges having infinite left and right extent, respectively. During the sweep, we maintain a balanced search tree whose leaves represent the active intervals of the current sweep line (and the associated generators), ordered by y-coordinate. Conceptually, we imagine that the sweep line jumps from one corridor between successive vertices to the next, because within a corridor neither the ordering nor the generators of the active intervals can change.

Consider what happens when we pass a vertex v with l left-going and r right-going edges. At that moment we must delete the $l+1$ intervals bounded by edges that end at v, and insert in their place the $r+1$ intervals bounded by edges that start at v. In particular, if $l=0$, we must delete the currently active interval in which v lies, which is bounded by the edges immediately above and below v. Similarly, if $r=0$, we add one new interval bounded by those two edges. Updating the generators is straightforward: the vertex v simply becomes the generator of the $r+1$ new intervals.

When an active interval is about to be deleted, we check whether a new regularizing edge should be added to the subdivision. If the generator u of that interval has no right-going edges, or the new vertex v has no left-going ones, we add a new edge between u and v. See Fig. 6. By Lemma 2, this edge does not intersect any others.

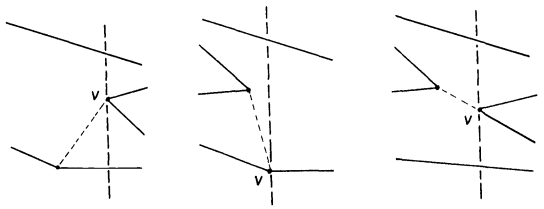


FIG. 6. *Adding new edges.*

When the sweep line reaches the vertex at $x = +\infty$, all irregular vertices will have been fixed in this way. The number of new edges is at most the original number of irregular vertices, which in turn is at most m. The running time is dominated by the cost of sorting of the vertices and manipulating the balanced tree, which is $O(m \log m)$ in the worst case. The problem of whether regularization can be done in time faster than $O(m \log m)$ remains open.

4. The vertical ordering. We denote by ΠA the orthogonal projection of a set A on the x-axis. The projection ΠR of a monotone polygon is an open interval of the

x-axis, whose endpoints, if any, are the projections of at most two *extremal vertices* of R. The *frontier* of R is the boundary of R minus its extremal vertices. This frontier consists of two disjoint pieces, the *top* and the *bottom* of the region R. Each is either empty, or a connected polygonal line, which may extend to infinity in one or both directions.

Given two subsets A and B of the plane, we say that A *is above* B (and write $A \gg B$) if for every pair of vertically aligned points (x, y_a) of A and (x, y_b) of B we have $y_a \geqq y_b$, with strict inequality holding at least once. In this case we also say that B *is below* A (and write $B \ll A$). Some of these concepts are illustrated in Fig. 7. For the rest of this section, we will restrict $\gg$ to the elements of a fixed monotone subdivision $\mathcal{S}$, with n regions and m edges. Then $\gg$ has several interesting properties listed in the lemmas below (straightforward proofs will be omitted).

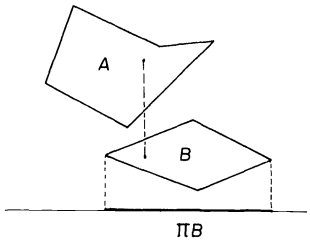


FIG. 7. $A \gg B, \Pi B$.

LEMMA 3. *For any two elements A and B of a monotone subdivision, $A \gg B$ if and only if $A \neq B$ and there is some vertical line segment with lower endpoint in B and upper endpoint in A.* □

LEMMA 4. *Any two elements A and B of a monotone subdivision satisfy exactly one of $A = B$, $A \ll B$, $A \gg B$, or $\Pi A \cap \Pi B = \varnothing$.* □

If three elements A, B and C are intercepted by a common vertical line, then from $A \ll B$ and $B \ll C$ we can conclude $A \ll C$. Therefore, the relation $\ll$ is transitive (and in fact a linear order) when restricted to all the elements intercepted by a given vertical line. Transitivity may not hold without this restriction, but the following property will be true in any case:

LEMMA 5. *The relation $\ll$ is acyclic.*

Proof. Suppose not. Let $E_0 \ll E_1 \ll E_2 \ll \cdots \ll E_k = E_0$ be a cycle of $\ll$ with minimum length, where each E_i is an element of the subdivision $\mathcal{S}$. From Lemma 4, it follows immediately that $k > 2$.

The x-projections of any two consecutive elements E_i and E_{i+1} in this cycle must have some abscissa x_i in common. If for some i we had $\Pi E_i \subset \Pi E_{i-1}$, then the vertical line through x_i would meet E_{i-1}, E_i, and E_{i+1}; transitivity would hold, and we would

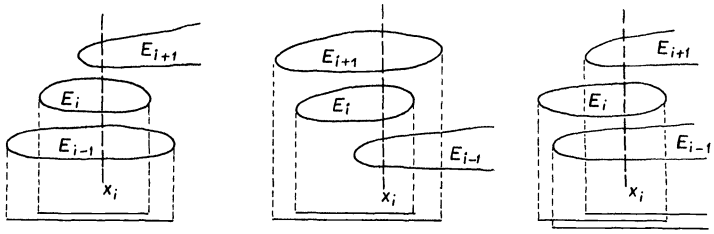


FIG. 8

have $E_{i-1} \ll E_{i+1}$. See Fig. 8. But then we could omit E_i and still get a cycle, contradicting the assumption that k is minimum. For analogous reasons, we cannot have $\Pi E_i \subset \Pi E_{i+1}$.

Let E_i be one of the "leftmost" cycle elements, i.e., such that none of the intervals ΠE_j contains a point to the left of ΠE_i. The projections ΠE_{i-1} and ΠE_{i+1} of its neighbors must both meet ΠE_i, and, by what we have just said, extend beyond its right endpoint. But then there would be again a vertical line meeting all three elements, implying $E_{i-1} \ll E_{i+1}$ and k is not minimum. We conclude that no such cycle exists. □

We say that a region A is *immediately above* a region B (and write $A > B$) if $A \gg B$ and the frontiers of the two regions have at least one common edge; see Fig. 9.

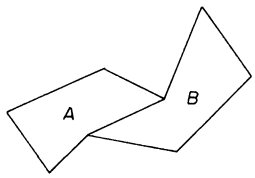


FIG. 9. $A > B$.

In general, the $>$ relation is stronger than $\gg$, but the following is easily seen to be true:

LEMMA 6. *The transitive closure of* $\gg$ (*restricted to the regions of a subdivision*) *is the same as that of* $>$. □

5. Separators. A *separator* for a subdivision $\mathcal{S}$ is a polygonal line s, consisting of vertices and edges of $\mathcal{S}$, with the property that it meets every vertical line at exactly one point. Since s extends from $x = -\infty$ to $x = +\infty$, any element of the subdivision that is not part of s is either above or below it. See Fig. 10. The elements of s have pairwise disjoint projections on the x-axis, and so can be ordered from left to right; the first and last elements are infinite edges.

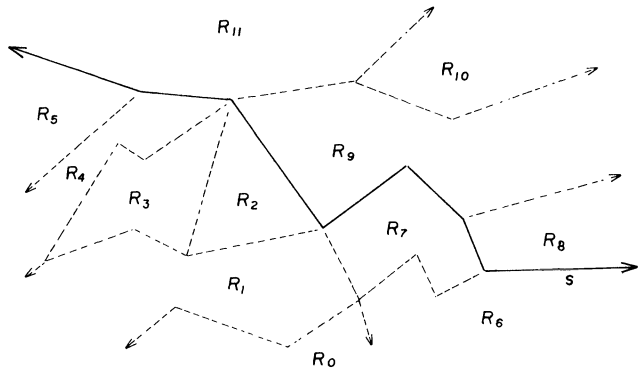


FIG. 10. *A separator.*

A *complete family of separators* for a monotone subdivision $\mathcal{S}$ with n regions is a sequence of $n-1$ distinct separators $s_1 \ll s_2 \ll \cdots \ll s_{n-1}$. There must be at least one region between any two consecutive separators, and also one below s_1 and one above s_{n-1}. If a region is below a separator s_i, it must also be below any separator s_j with $j > i$. We can conclude that, if $\mathcal{S}$ admits a complete family of separators, its regions can be enumerated as $R_0, R_1, \cdots, R_{n-1}$ in such a way that $R_i \ll s_j$ if and only if $i < j$;

in particular,

$$R_0 \ll s_1 \ll R_1 \ll s_2 \ll \cdots \ll s_{n-1} \ll R_{n-1}. \tag{1}$$

Given a complete family of separators and an enumeration of the regions as in (1), let us denote by index (R) the index of a region R in the enumeration. Then $s_{\text{index}(R)} \ll R \ll s_{\text{index}(R)+1}$ (whenever those separators are defined). Also, if we let above (e) be the region above the edge or vertex e, and below (e) the one below it, the following holds:

LEMMA 7. *If* $i = \text{index (below } (e))$ *and* $j = \text{index (above } (e))$, *then the separators containing e will be exactly* $s_{i+1}, s_{i+2}, \cdots, s_j$. □

6. Existence of a complete family of separators. It is a well-known result that any acyclic relation over a finite set can be extended to a linear (that is, total) order. Therefore, by using Lemma 5 we conclude that there is an enumeration $R_0, R_1, \cdots, R_{n-1}$ of the regions of $\mathcal{S}$ that is compatible with $\ll$, i.e., such that $R_i \ll R_j$ implies $i < j$. Furthermore, any enumeration $R_0, R_1, \cdots, R_{n-1}$ of the regions that is compatible with $\prec$ is also compatible with $\ll$, and vice-versa.

Since a region with nonempty bottom frontier is immediately above some other region, and therefore not minimal under $\prec$, the first region in any such enumeration has no bottom frontier, and extends to infinity in the $-y$ direction. Since vertical edges are not allowed, its x-projection is the whole x-axis. Similarly, the last region R_{n-1} is the only one with no top frontier, and also projects onto the whole x-axis. Therefore, we always have $R_0 \ll R_i \ll R_{n-1}$ for $0 < i < n-1$.

We are ready to prove the main result of this section:

THEOREM 8. *Every monotone subdivision* $\mathcal{S}$ *admits a complete family of separators.*

Proof. Let $R_0, R_1, \cdots, R_{n-1}$ be a linear ordering of the regions of $\mathcal{S}$ that is compatible with the $\ll$ relation, i.e. $R_i \ll R_j$ only if $i < j$. For $i = 1, 2, \cdots, n-1$, let s_i be the collection of all edges and vertices that are on the frontier between regions with indices $< i$ and regions with indices $\geq i$. For example, Fig. 10 shows the separator s_8.

Now consider any vertical line l, and let $R_{i_1}, R_{i_2}, \cdots, R_{i_q}$ be the regions it meets, from bottom to top. Since l meets R_0 and R_{n-1}, and $R_{i_1} \ll R_{i_2} \ll \cdots \ll R_{i_q}$, we will have $0 = i_1 < i_2 < \cdots < i_q = n-1$. Therefore, there is exactly one point on l that is on the frontier between a region with index $< i$ and a region with index $\geq i$, that is, on s_i. Furthermore, the intersection of l with s_i will be equal to or above that with s_{i-1}, and for some such lines (those that meet R_{i-1}) the two intersections will be distinct. So, we have $s_1 \ll s_2 \ll \cdots \ll s_{n-1}$.

Clearly, the elements of s_i have disjoint x-projections, and therefore can be ordered from left to right; they must be alternately edges and vertices, the first and last being infinite edges. To prove that s_i is a separator, it remains only to show that s_i is connected; if that were not the case, we would have some vertex v of s_i that is distinct from, but has the same x-coordinate as, one endpoint of an adjacent edge e of s_i; see Fig. 11.

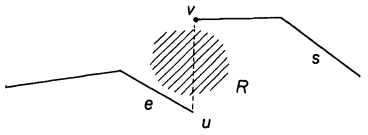


FIG. 11

But then we would have $u \ll R \ll v$ (or vice-versa), for some region R; and therefore $e \ll R \ll v$ (or vice versa), contradicting the construction of s_i. Therefore, each s_i is a separator, and $s_1, s_2, \cdots, s_{n-1}$ is a complete family of them. □

7. A point location algorithm. Later on we will tackle the problem of efficiently computing a complete family of separators for a monotone subdivision. Let us therefore assume for now that we have such a family $s_1, s_2, \cdots, s_{n-1}$, with a corresponding enumeration of the regions $R_0, R_1, \cdots, R_{n-1}$ satisfying (1); we will show next how they can be used to determine, in time $O(\log^2 m)$, the unique element of $\mathcal{S}$ that contains a given point p.

The algorithm we will describe is essentially that of Lee and Preparata [LP], and uses two levels of binary search. The inner loop takes a separator s_i (as a linear array of edges and vertices, sorted by x-coordinate), and determines by binary search an edge or vertex e of s_i whose x-projection contains the abscissa p_x of p. By testing p against e, we will know whether p is above s_i or below s_i (or, possibly, on e itself, in which case the search terminates). The outer loop performs binary search on i, so as to locate p between two consecutive separators s_i and s_{i+1}, that is to say in a region R_i.

Besides the separators, the algorithm assumes the index, index (R) can be obtained for any given region R, and similarly the adjacent regions above (e) and below (e) can be obtained from an edge e, all in constant time. We will see that the construction of these tables is part of the process of constructing the family of separators. The search algorithm uses these tables to reduce substantially the storage requirements (and also speed up the search a little bit).

Let **T** be the infinite, complete binary search tree with internal nodes $1, 2, 3, \cdots$ and leaves $0, 1, 2, 3, \cdots$, as in Fig. 12. The tree **T** is used as a flowchart for the outer loop of the search algorithm, with the convention that each internal node i represents a test of p against the separator s_i, and each leaf j represents the output "p is in R_j". While reading the algorithm it should be borne in mind that "left" in the tree corresponds to "down" in the subdivision. The left and right children of an internal node k of **T** will be denoted by $l(k)$ and $r(k)$, respectively. We let lca (i, j) be the *lowest common ancestor* in **T** of the leaves i and j, that is, the root of the smallest subtree of **T** that contains both i and j.

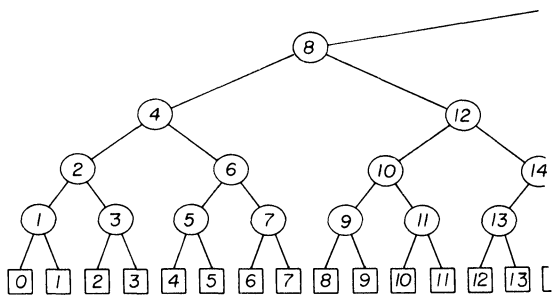


FIG. 12. *The tree* **T**.

When testing p against a separator, we adopt the convention that each edge contains its right endpoint but not its left. This is unambiguous since there are no vertical edges. If the algorithm detects that p lies on some edge e during a discrimination against a separator, it can terminate the search and, by comparing p with the right endpoint of e, determine if our point is a vertex of the subdivision.

ALGORITHM 1. *Point location in a monotone subdivision.*

{*The algorithm returns in the variable loc a reference to the vertex, edge, or region of* $\mathcal{S}$ *containing p.*}

1. Set $i \leftarrow 0$, $j \leftarrow n-1$, $k \leftarrow \text{lca}(0, n-1)$.
2. While $i < j$, do:
   {*At this point p is above the separator* s_i *and below* s_{j+1} (*whenever those separators exist*). *That is, p is in one of the regions* R_i, $R_{i+1}, \cdots, R_j$ (*or in some edge or vertex between two of these regions*). *At each iteration of this loop, either i increases or j decreases, and k* (*a common ancestor of i and j*) *moves down one level in the tree* **T**}.
   3. If $i < k \leqq j$ then
      4. Find (by binary search) an edge e in s_k such that $p_x \in \Pi e$.
         Let $a \leftarrow \text{index (above } (e))$, $b \leftarrow \text{index (below } (e))$.
         {*By Lemma 7, e belongs to the separators* $s_{b+1}, s_{b+2}, \cdots, s_a$. *Therefore, by testing p against e we conclude it is either on e, above* s_a, *or below* s_{b+1}.}
      5. If p is on e, set $loc \leftarrow e$ and terminate the search.
      6. If p is above e, set $i \leftarrow a$; else set $j \leftarrow b$.
   7. Else
      8. If $k > j$ set $k \leftarrow l(k)$; else (if $k \leqq i$) set $k \leftarrow r(k)$.
9. Set $loc \leftarrow R_i$ and terminate the search.

The binary search along each separator s_k can be performed in $O(\log m)$ time if the edges of s_k are stored in a linear array or balanced binary search tree sorted from left to right. By the first iteration, the variable $k = \text{lca}(0, n-1)$ points to an internal node of **T** at level $\lceil \log n \rceil$; at each iteration it descends one level, so we have $O(\log n)$ iterations, and a total time bound of $O(\log n \log m) = O(\log^2 m)$. This bound is tight: Fig. 13 shows a subdivision with m edges that realizes it. This example has $\sqrt{m}+1$ regions and a family of $\sqrt{m}$ disjoint separators with $\sqrt{m}$ edges each.

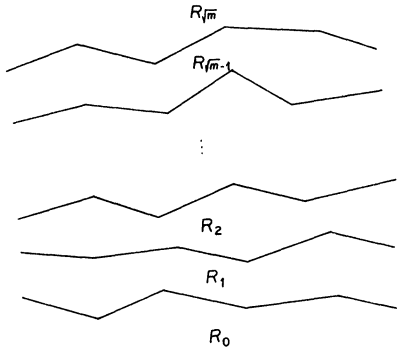


FIG. 13. *A subdivision that is bad for Algorithm* 1.

Note that by keeping track of the variables i and j we are sometimes able to skip the binary search for p_x in some separators. This optimization may improve the average running time of the algorithm in practice, but does not affect the worst-case bound. It was included primarily to make algorithm 1 more similar to the variants developed further on.

If we were to independently represent each separator as a linear array, with all its edges and vertices, we would have to store $n-1$ separators, whose average length can be as large as $\Theta(m)$ for some classes of subdivisions. So, the storage requirement

of this basic method is $\Theta(m^2)$ in the worst case. However, after p has been tested against the edge e in step 6, i and j are updated in such a way that we will never again look at the edges of any other separator that contains e. Therefore, an edge need only be stored in the *first* separator containing it that would be encountered in a search down the tree **T**. Specifically, if the edge e is in the common frontier of regions R_i (below) and R_j (above), by Lemma 7 it belongs to separators $s_{i+1} \cdots s_j$ and so it suffices to store it in s_k, where k is the least common ancestor of i and j. This is the highest node in **T** whose separator contains e.

Note that only those edges assigned to s_k according to the above rule are actually stored in such a structure. In general these will form a proper subset of all the original edges of s_k, so between successive stored edges of s_k there may be *gaps*. See Fig. 14. Actually, it may happen that all the edges of s_k are stored higher up in the tree, so that s_k is reduced to a single gap, extending from $x = -\infty$ to $x = +\infty$. The ordered list of stored edges and gaps corresponding to separator s_k will be termed the *chain* c_k. Note that to each subtree of **T** rooted at a node k there corresponds a "partial subdivision" consisting of a range of regions and the edges and vertices between them. The separator s_k splits this partial subdivision into two others, each having half the regions; the gaps of c_k correspond to edges of s_k that lie on the upper or lower boundary of the partial subdivision, as shown in Fig. 14.

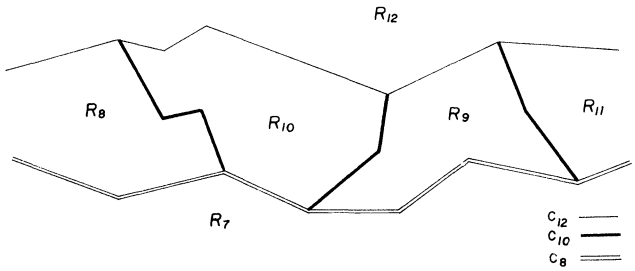


FIG. 14. *Stored edges and gaps.*

The total storage required to represent the chains is only $O(m)$; in § 10 we show how they can be constructed in $O(m)$ time. The derivation of this bound is contingent on our ability to compute the least common ancestor of any two leaves of **T** in $O(1)$ time. This is made possible by the fixed, regular structure of the search tree **T** (see § 11). The point location phase proper could easily be adapted to search a more conventional linked tree structure, but such structures seem to admit no simple $O(1)$ algorithm for lca determination.

8. A faster point location method. Our hope to obtain a faster algorithm for point location comes from the fact that there is some obvious information loss in the method of § 7. Specifically, when we discriminate a point against a chain c_k, we must localize it in the x coordinate to within an edge or a gap of c_k. Yet when we continue the search down some child of k, we start this localization process all over again. It would be nice if each edge or gap in a chain pointed to the place on each child chain where the x search on that child will finish. The trouble is that an edge or gap of the parent can cover many edges or gaps of the child.

The novel idea in the technique we present in this section is to refine the chains so that the localization of p_x in one chain allows us to do the same localization in its children with only *constant* extra effort. In its ultimate form, this idea leads to breaking up each chain at the x coordinates of all vertices of the subdivision. A bit of thought will convince the reader, however, that such an extensive refinement will require

quadratic storage for its representation. Instead, we describe below a refinement that produces for each chain c_k a list L_k of x-values, defining a partitioning of the x axis into *x-intervals.* Each such interval of L_k overlaps the x-projection of exactly one edge or gap of c_k and *at most two* x-intervals of the lists $L_{l(k)}$ and $L_{r(k)}$. As we will see in § 9, this last condition is compatible with keeping the overall storage linear.

The lists L_k and their interconnections can be conveniently represented by a linked data structure that we call the *layered dag.* This is a directed acyclic graph whose nodes correspond to tests of three kinds: *x-tests, edge tests,* and *gap tests.* A list L_k is represented in the dag by a collection of such nodes: each x-value of L_k gives rise to an x-test, and each interval between successive x-values to an edge or gap test. See Fig. 15.

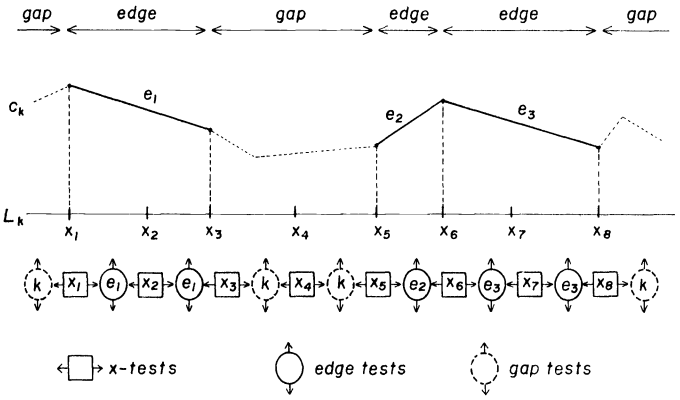


FIG. 15. *The dag nodes for list* L_k.

An x-test node t contains the corresponding x-value of L_k, denoted by xval (t), and two pointers left (t) and right (t) to the adjacent edge or gap nodes of L_k. An edge or gap test node t contains two links down (t) and up (t) to appropriate nodes of $L_{l(k)}$ and $L_{r(k)}$. In addition, an edge test contains a reference edge (t) to the edge of c_k whose projection covers the x-interval represented by t. A gap test node contains instead the chain number chain $(t) = k$. The various types of nodes present are illustrated in Fig. 16.

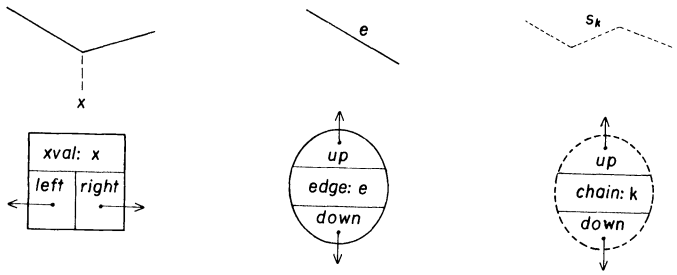


FIG. 16. *The nodes of the layered dag.*

Let us define more precisely the meaning of the links down (t) and up (t). The properties of the refined lists ensure that the x-interval of L_k corresponding to an edge or gap test t covers either one x-interval I of $L_{l(k)}$, or two such intervals I_1, I_2 separated by some element x_k of $L_{l(k)}$. In the first case, down (t) points to the edge or gap test of $L_{l(k)}$ corresponding to the interval I; in the second case, down (t) points to the x-test corresponding to the separating abscissa x_k. Similarly, the link up (t) points to

a node of $L_{r(k)}$ defined in an analogous manner. In the special case when $r(k)$ and $l(k)$ are leaves of **T**, we let down $(t) = \text{up}(t) = nil$.

The layered dag then consists of the test nodes for all lists $L_1, L_2, \cdots, L_{n-1}$, linked together in this fashion. We can use this dag to simulate algorithm 1; the main difference is that each time we move from a separator to one of its children, the down or up pointers (possibly together with an x-test) allow us to locate x_p in the new chain in $O(1)$ time. As before, the variables i and j keep track of the interval of separators in which the point p is known to lie, namely above s_i and below s_{j+1}. This interval is updated after each edge test exactly as in the previous algorithm, and is used during a gap test. When we come to a gap test, we know that the chain number of the gap is not interior to the interval in question. This gives us an unambiguous test as to whether we are above or below the chain of the gap. By the time the search algorithm gets to a null up or down link, the interval will have been narrowed down to a single region.

We have now presented enough details about the structure of the layered dag that we can give the code for the point-location algorithm. The layered dag contains a distinguished node *root* where the point location search begins. This node is the root of a balanced tree of x-tests whose leaves are the edge tests corresponding to the list for the root node of **T**.

ALGORITHM 2. *Fast point location in a monotone subdivision.*

{*This algorithm takes as input the root node of the layered dag and the number n of regions. Its output, as in algorithm* 1, *is placed in the variable loc.*}

1. Set $i \leftarrow 0$, $j \leftarrow n-1$, $t \leftarrow root$.
2. While $i < j$ do:
   {*At this point we know p is above the separator* s_i *and below* s_{j+1} (*whenever those separators exist*). *That is, p is in one of the regions* $R_i, R_{i+1}, \cdots, R_j$ (*or in some edge or vertex between two of these regions*). *The variable t points to a test node in the layered dag, which together with its descendants will allow us to locate the point p among those regions.*}
   3. If t is an edge test then let $e \leftarrow \text{edge}(t)$ and do:
      {*At this point we know* p_x *lies within* Πe.}
      4. If p is on e, set $loc \leftarrow e$ and terminate the algorithm.
      5. If p is above e, set $t \leftarrow \text{up}(t)$ and $i \leftarrow \text{index}(\text{above}(e))$.
         Else set $t \leftarrow \text{down}(t)$ and $j \leftarrow \text{index}(\text{below}(e))$.
   6. Else if t is an x-test then do:
      {*The following x-test routes us to the appropriate edge of the next chain we need to test against.*}
      7. If $p_x \leqq \text{xval}(t)$ then $t \leftarrow \text{left}(t)$ else $t \leftarrow \text{right}(t)$.
   8. Else t is a gap test; do
      {*We have already compared p against the appropriate edge of the chain of the gap test. We just need to recontruct how that comparison went.*}
      9. If $j < \text{chain}(t)$ then $t \leftarrow \text{down}(t)$ else $t \leftarrow \text{up}(t)$.
10. Set $loc \leftarrow R_i$ and terminate the search.

9. Computing the layered dag. Now that we understand how the layered dag is to be used, we will describe how it is to be constructed. Our starting point will be the tree **T** and the chains c_k defined in § 7; recall that c_k consists of those edges of s_k that do not belong to any ancestor of s_k (that is, to any separator whose index is an ancestor of k in **T**).

Our construction of the layered dag proceeds from the bottom up and happens simultaneously with the refinement of the chains c_k. We first describe how the x values

in L_k are obtained. Note that we already have at our disposal three sorted lists of x values: those corresponding to $L_{l(k)}$, to $L_{r(k)}$, and also to the endpoints of the edges stored with the chain c_k associated with node k in the chain tree. The x values in L_k are a merge of those in c_k, and *every other one* of those present in each of $L_{l(k)}$ and $L_{r(k)}$. By convention, if k is a terminal node of the chain tree (so it corresponds to a region), or $k \geqq n$, then L_k is empty. We imagine now that, in a bottom-up fashion, this is done for every node k of the chain tree. The propagation of every other value from the children to the father constitutes the chain refinement we had mentioned in § 8. This refinement has two desirable properties:

LEMMA 9. *An interval between two successive x values in L_k overlaps at most two intervals in $L_{l(k)}$, or $L_{r(k)}$, respectively.* □

LEMMA 10. *The total number of x values in the lists L_k, summed over all k, is at most 4m.*

Proof. If a_k denotes the number of edges in c_k, then

$$\sum_{k \in T} a_k = m,$$

since each edge of the subdivision occurs in exactly one chain c_k. Let b_k denote the number of x values in L_k, and A_k (resp. B_k) denote the sum of a_i (resp. b_i) over all nodes i in the subtree rooted at k. To prove the lemma it suffices to show that

$$B_r \leqq 4A_r = 4m$$

for the root node $r = \text{lca}\,(0, n-1)$.

As an inductive hypothesis now assume that

$$B_i + b_i \leqq 4A_i \tag{2}$$

for $i = l(k)$ or $i = r(k)$. This hypothesis is trivially true for the leaves of **T**. Observe now that

$$B_k = B_{l(k)} + B_{r(k)} + b_k,$$

and that

$$b_k \leqq 2a_k + (b_{l(k)} + b_{r(k)})/2,$$

since each edge in c_k contributes at most two x-values to L_k. Applying the inductive hypothesis (2) yields

$$B_k + b_k \leqq 4A_k,$$

which proves the same conclusion for k. This concludes the proof of the lemma. □

Intuitively, half of the x values of c_k propagate to the father chain, a quarter to the grandfather, an eighth to the great-grandfather, and so on. Although this argument is not strictly correct it illustrates why we can expect the combined length of the L lists to remain linear in m, and in fact just twice as great as the total number of x values in the chains c_k.

The construction of the x-test, edge test, and gap test nodes representing the list L_k in the layered dag is now straightforward, as well as the setting of the left and right fields of the x-tests. The down links of L_k can be set according to the rules stated in § 8, by traversing simultaneously the two lists L_k and $L_{l(k)}$ from left to right. The up pointers are analogous. In fact, it is possible to build L_k and link it to the rest of the dag in a single simultaneous traversal of c_k, $L_{l(k)}$, and $L_{r(k)}$. This bottom-up process terminates with the construction of the root chain L_r, where $r = \text{lca}\,(0, n-1)$. As a final

step we produce a tree of x-test nodes corresponding in a natural way to a binary search on the x-values of the list L_r. The leaves of the tree are made to point to the appropriate edge test nodes of L_r. All nodes of the layered dag can be reached from the root of that tree.

ALGORITHM 3. *Construction of the layered dag.*

1. Set $i \leftarrow 1$. While $i < n$ do:
   {*Construct one more level of the tree* **T**. *The first node in this level is* i *and the difference between successive nodes is* $2i$. *A node* k *on this level is the common ancestor of the leaves* $k-i$ *through* $k+i-1$; *its children* (*if* $i>1$) *are the internal nodes* $k-i/2$ *and* $k+i/2$.}
2. Set $k \leftarrow i$. While $k - i < n$ do:
   {*Create the list* L_k *from the chain* c_k *and the lists* $L' = L_{l(k)}$ *and* $L'' = L_{r(k)}$ (*if they exist*).}
   3. If $i = 1$, set $L' \leftarrow \varnothing$, else set $L' \leftarrow L_{k-i/2}$.
   4. If $i = 1$ or $k + i/2 \geqq n$, set $L'' \leftarrow \varnothing$, else set $L'' \leftarrow L_{k+i/2}$.
   5. If $k \geqq n$, let c_k be a single gap from $x = -\infty$ to $x = +\infty$.
      Split the edges and gaps of c_k at every other x-value of L' and L''.
   6. Set $L_k \leftarrow \varnothing$. For each edge or gap e in c_k, do:
      7. Append to L_k an edge or gap test t representing e.
         Set edge $(t) \leftarrow e$ or chain $(t) \leftarrow k$, as appropriate.
      8. If $L' = \phi$, let down $(t) \leftarrow$ *nil*.
         Else, if Πe overlaps only one x-interval of L', let down (t) point to the corresponding edge or gap test of L'.
         Else Πe overlaps two x-intervals of L'; create a new x-test node t' that chooses between the two corresponding edge or gap tests of L', and let down $(t) \leftarrow t'$.
      9. Similarly, set up (t) to *nil*, to an edge or gap test of L'', or to a new x-test that chooses between two tests of L''.
   10. Set $k \leftarrow k + 2i$.
   11. Set $i \leftarrow 2i$.
12. Let $r \leftarrow \text{lca}(0, n-1)$. Construct a binary tree of x-tests for the list L_r, and let *root* point to this tree.

Note that several different nodes of L_k may point to the same node of a child; an example is shown in Fig. 17. Thus the resulting dags are not trees. We remark that

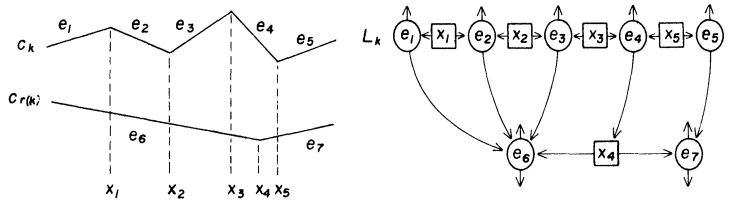


FIG. 17. *Convergence in the dag.*

the structure built by Kirkpatrick [Ki] also corresponds to a dag. This "sharing" of subtrees seems to be an essential feature of algorithms for point location that simultaneously attain optimal space and query time.

To cut down on the number of links, we may consider storing the edge and gap test nodes of each list L_k in consecutive memory locations, in their natural left-to-right

order, with the x-tests between them. The initial location of x_p in L_r could then be carried out by standard binary search. Also, in the construction of L_k we would be able to scan the lists $L_{l(k)}$ and $L_{r(k)}$ without any auxiliary pointers or tables. Finally, this sequential allocation would eliminate the need for the left and right links of x-tests, a fact that may reduce the total storage used by pointers in the dag by about one third.

In any case, the initial location of x_p in the root list L_r can be determined in $O(\log m)$ time. After that, Algorithm 2 executes exactly $\lceil \lg n \rceil$ edge or gap tests (one at each level of **T**), and at most that many x-tests. So the total query time is $O(\log m)$. A list L_k with t x-values is represented in the layered dag by at most t x-tests and $t+1$ edge/gap tests and, as we have seen, it can be constructed in $O(t)$ time. Using Lemma 10, we conclude that the layered dag contains at most $4m$ x-tests and $4m+n-1$ edge and gap tests, and can be built in total time $O(m+n)$ from the chain tree **T**. In summary, we have shown that

THEOREM 11. *Assuming that the chain tree for a subdivision with m edges is given, the layered dag data structure can be constructed in $O(m)$ time, takes $O(m)$ space, and allows point location to be done in $O(\log m)$ time.* □

10. Constructing a complete family of separators. We proceed now to the description of an efficient algorithm that constructs the chain tree **T** representing a complete family of separators for a given monotone subdivision $\mathcal{S}$. As suggested by the proof of Theorem 8, the first pass of this algorithm enumerates the regions in a linear order compatible with $\ll$. A second pass enumerates the edges and vertices of $\mathcal{S}$, in such a way that the edges and vertices of each separator are visited in order of increasing x-coordinate (even though this may not be true for elements belonging to different separators). Therefore, by just appending each visited element to its appropriate separator, we will get the sorted lists required in Algorithms 1 and 3.

As in § 3, we will add to S two dummy vertices at $x=-\infty$ and $x=+\infty$, which are endpoints of all edges with infinite left or right extent, respectively. If we orient every edge of $\mathcal{S}$ from right to left, we obtain a planar embedding of a directed, acyclic graph S with exactly one source and one sink (solid lines in Fig. 18). The enumeration we need in the second pass is basically a *compatible traversal* of this graph, in which we visit a vertex only after visiting all its outgoing edges, and we visit an edge only after visiting its destination. This is a form of topological sorting, as discussed in [Kn].

Consider now the dual graph S^* whose vertices are the regions of $\mathcal{S}$, and where for each edge e of $\mathcal{S}$ there is an edge e' from above (e) to below (e). By what we have seen in §§ 4–6, S^* too is acyclic and has exactly one source and one sink. We can draw S^* on top of S (dotted lines in Fig. 18) in such a way that each of its edges e' crosses only the corresponding edge e of S, and that exactly once (going down).

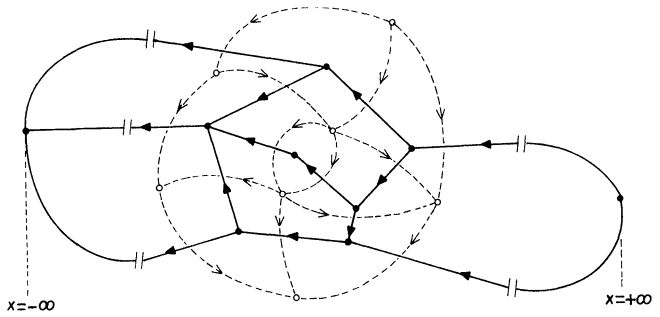


FIG. 18. *The graphs S and S**.

Therefore, S^* is planar, and corresponds to the topological dual of S. It turns out that S^* represents for the first pass what S does for the second: a compatible traversal of S^* will visit the regions in an order consistent with $\prec$ and $\ll$.

Therefore, both passes reduce to a generic graph traversal algorithm, applied first to S^* and then to S. In the first pass, as each region R is visited, we store in a table (or in a field of the region's record) its serial number index (R) in the enumeration. In the second pass, as each edge or vertex e is visited, we obtain the indices $a =$ index (above (e)) and $b =$ index (below (e)) of the regions immediately above and below it, and append e to the separating chain c_k where $k = \text{lca}(a, b)$. With an appropriate representation for subdivisions, it is possible to accomplish this graph traversal without any auxiliary storage. This topic is discussed in § 12.

11. The lowest common ancestor function. If the construction of the separating chains c_i as described above is to run in $O(m)$ time, it is essential that the total time to compute lca (index (below (e)), index (above (e))) for all edges e be $O(m)$. This rules out the straightforward algorithm that starts at the leaves i and j, and moves up one level of **T** at a time, following "parent" links, until the two paths join at a common node. This naïve algorithm has running time $\Omega(\log |i-j|)$, and it is possible to have subdivisions in which $|\text{index (below } (e)) - \text{index (above } (e))| = \Omega(\sqrt{m})$ for $\Omega(m)$ edges e, thus giving an overall running time of $\Omega(m \log m)$.

Algorithms for computing in $O(1)$ time the least common ancestor function on general binary trees have been published by Harel [H]. His algorithms are probably too complex to be of practical use here, but they can be considerably simplified thanks to the regular structure of our search tree **T**. On this tree, the value of lca (i, j) has a simple interpretation in terms of the binary representations of i and j. Let $\nu = \lceil \lg n \rceil$ be the number of bits needed to represent any number from 0 through $n-1$, and let

$$i = (a_{\nu-1}a_{\nu-2} \cdots a_{r+2}\ 0\ a'_r \cdots a'_1 a'_0)_2,$$
$$j = (a_{\nu-1}a_{\nu-2} \cdots a_{r+2}\ 1\ a''_r \cdots a''_1 a''_0)_2.$$

Then

$$\text{lca}(i, j) = (a_{\nu-1}a_{\nu-1} \cdots a_{r+2}\ 1\ 0\ \cdots 0\ 0)_2.$$

Loosely speaking, lca (i, j) is the longest common prefix of i and j, followed by 1 and padded with zeros.

An efficient formula for computing lca (i, j) is based on the function $\text{msb}(k) = \text{lca}(0, k) = 2^{\lceil \lg k \rceil - 1}$, the *most significant bit of* k. Its values for $k = 1, 2, \cdots, n-1$ are 1, 2, 2, 4, 4, 4, 4, 8, 8, $\cdots$, $2^{\nu-1}$; these numbers can be easily precomputed in $O(n)$ time and stored in a table with $n-1$ entries, so we can compute msb (k) in $O(1)$ time. Then we can express the lca function as

$$\text{lca}(i, j) = j \wedge \neg(\text{msb}(i \oplus j) - 1)$$

where $\oplus$, $\wedge$, and $\neg$ are the boolean operations of bitwise "exclusive or", "and", and complement. We assume these boolean operations can be computed in $O(1)$ time, like addition and subtraction. We feel their inclusion in the model is justified, since their theoretical complexity is no greater than that of addition, and most computers have such instructions.[2] For use in this formula, it is preferable to tabulate $\neg(\text{msb}(k) - 1)$ instead of msb (k).

[2] In fact, some machines can even compute $\lceil \lg k \rceil - 1$ from k ("find first 1 bit") and 2^q from q ("shift left q bits"), in a single instruction cycle. Under a more rigorous log-cost complexity model, all time and space bounds in this paper and in the main references should be multiplied by an extra log m factor.

Another way of computing lca is based on the *bit-reversal* function rev (k), that reverses the order of the last ν bits in the binary expansion of k. For example, for $n=16$ and $k=0, 1, \cdots, 15$, the values of rev (k) are 0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15. Using this function we get the formula

$$\text{lca}\,(i,j) = \text{rev}\,(k \oplus (k-1)) \wedge j,$$

where $k = \text{rev}\,(i \oplus j)$.

12. Compatible traversal of an acyclic graph. The input to the compatible graph traversal routine we mentioned in § 10 is a planar embedding G of a directed, acyclic, and connected graph with exactly one sink and one source, both on the exterior face of G. See Fig. 19.

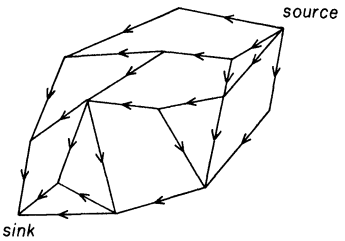


FIG. 19. *The input to Algorithm* 4.

Such an embedding defines a "counterclockwise" circular ordering of the edges incident to any given vertex u. The *post-order traversal* of G is defined as a listing of all its vertices and edges such that

(i) an edge is listed (or *visited*) only after its destination,
(ii) a vertex is visited only after all its outgoing edges, and
(iii) edges with same origin are visited in counterclockwise order.

This is clearly a compatible traversal as defined in § 10. The post-order traversal is unique, and is a particular case of the general depth-first graph traversal described by Tarjan [Ta]. This problem admits a straightforward recursive solution. Given a vertex u (initially the source of G), we enumerate the edges out of u, in counterclockwise order. For each edge e, we first recursively apply the procedure to its destination v (unless it has been previously visited). We then visit e and proceed to the next edge. After all edges out of u have been visited, we visit u and exit. This algorithm runs in $O(m)$ time and requires only $O(m)$ auxiliary storage, the latter consisting of the recursion stack and one mark bit per vertex (to distinguish the nodes that have already been visited).

In the rest of this section we will show that this post-order traversal can be performed without an auxiliary stack or any mark bits on the vertices, provided the data structure used to represent the subdivision is rich enough. This improvement is of significant practical interest, even though it does not affect the $O(m)$ space bound.

As we observed, the embedding of G in the plane defines a counterclockwise ordering of the edges incident to a given vertex u. In this ordering all outgoing edges occur in consecutive positions, and the same is true of the incoming ones. To see why, consider any two edges e_1, e_2 entering u, and any two edges g_1, g_2 leaving u. Let π_1, π_2 be two paths from the source vertex of G that end with the edges e_1 and e_2, and let σ_1, σ_2 be the two paths to the sink vertex that begin with g_1, g_2. See Fig. 20.

Since G is acyclic, both π_1 and π_2 are disjoint from σ_1 and σ_2 (except for u itself). Now, the paths π_1 and π_2 together divide the plane in two (or more) regions. If the two pairs of edges were interleaved, at least one of the paths σ_1 and σ_2 would have

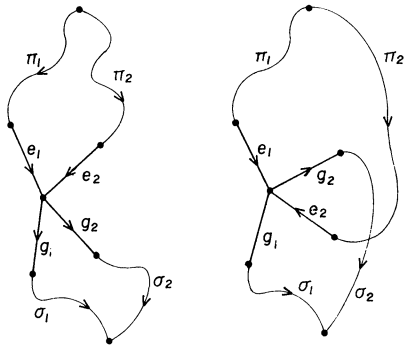


FIG. 20

to cross π_1 or π_2, since they start on different regions but have a common destination. This proves the above assertion.

If u has both incoming and outgoing edges, this result establishes a *linear* order for each class with well-defined "first" and "last" elements, which will be denoted by first in (u), last in (u), first out (u), and last out (u). See Fig. 21. To make this definition meaningful also for the source and sink of G, we will introduce a dummy edge base (G) that connects the sink back to the source across the exterior face of G. We may consider the resulting graph G' as embedded on the upper half of a sphere, with base (G) running across the bottom half, as in Fig. 22. In the graph G', the first and last outgoing edges of the source of G will be those incident to the exterior face of G. A similar statement applies to the sink of G.

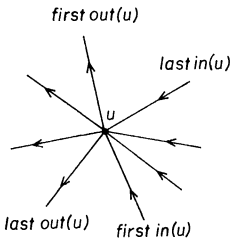


FIG. 21

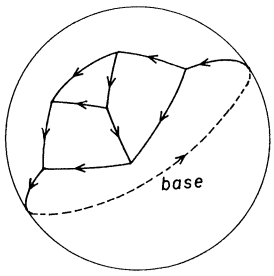


FIG. 22

Let us mechanically translate the recursive post-order algorithm into an iterative one, using an explicit stack Q to save the value of e when simulating recursive calls (the value of u need not be saved, since it is always the origin of e).

ALGORITHM 4. *Post-order traversal of an acyclic planar graph with one source and one sink.*

1. Set $Q \leftarrow \emptyset$, $u \leftarrow$ source of G, $e \leftarrow$ first out (u).
2. Repeat the following steps:
   {*At this point u is unvisited, and e is the first unvisited edge out of u.*}
   3. While $e \neq$ base (G) and the destination v of e has not been visited yet,
      4. Set $u \leftarrow v$, push e onto the stack Q, and set $e \leftarrow$ first out (u).
   5. If $e \neq$ base (G), visit e.
   6. While $e =$ last out (u) do
      7. Visit u.
      8. If Q is empty, the algorithm terminates. Else,
         9. Pop the topmost edge of Q, and assign it to e.
            Set u to the origin of e.
      10. Visit e.
   11. Set e to the next edge out of u.

The following claims about Algorithm 4 are a direct consequence of the planarity and acyclicity of G, and can easily be proved by induction on the number of executed steps:

(I) Before every step, the edges in the stack Q form a path in G from the source to u;
(II) an edge is stacked onto Q (step 4) only if its destination is still unvisited;
(III) an edge is unstacked (step 9) only after its destination has been visited;
(IV) an edge is visited only after its destination has been visited;
(V) a vertex is visited only after its last outgoing edge has been visited;
(VI) every edge is stacked at most once;
(VII) for any given vertex, at most one incoming edge ever gets stacked; and
(VIII) an edge is visited only after the previous edge with same origin (if any) is visited.

In particular, from (IV), (V), and (VIII) we conclude that all vertices and edges of G are visited, and conditions (i)-(iii) are satisfied. Algorithm 4 defines a subgraph H of G, consisting of all the edges that ever get stacked onto Q. Every vertex of G is reachable from the source via a directed path in H (the contents of the Q at any instant when the variable u is that vertex), and has at most one incoming edge in H. It follows that H is an oriented spanning tree of G, as illustrated in Fig. 23. We remark

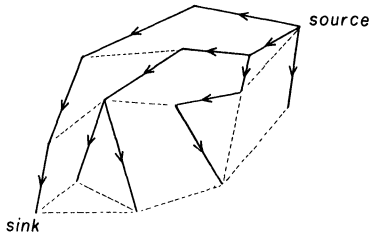


FIG. 23. *The spanning tree H.*

that the order in which the vertices of G are visited corresponds to the post-order traversal of the tree H, as defined by Knuth [Kn]. We will show now that the only edge of H (if any) entering a vertex u is last in (u). More precisely, the following lemma holds:

LEMMA 12. *Before any step of algorithm* 4, *if the stack Q is not empty, its topmost edge is last in* (u); *otherwise u is the source of G.*

Proof. The second part of the lemma is obvious, since the contents of Q is always a path in G from the source to u. Let us then assume Q is not empty. Let u_1 be the current value of u, π be the path in Q, and e_1 be the last edge of π (i.e., the top of Q). See Fig. 24. Suppose e_2 is an edge distinct from e_1 but having the same destination u_1. From assertions (II) and (III) above we conclude that e_1 and e_2 have yet to be

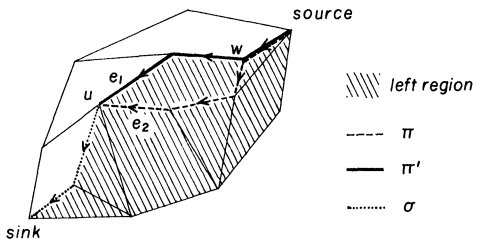


FIG. 24

visited. By the time e_2 is visited by Algorithm 4, the variable u will contain the origin of e_2; the contents of Q at that time, plus the edge e_2, will define another directed path π' from the source to u_1. Since G is acyclic, the path π' neither contains nor is contained in π; in fact, because of property (VII) the paths π and π' must diverge exactly once, at some vertex $w \neq u_1$, and converge exactly once at u_1. Therefore all edges of π between w and u_1 must be unstacked (and visited) before e_2 is visited. In particular, the edge α through which π leaves w is visited before the corresponding edge α' of π', and therefore α' must follow α in counterclockwise order around w.

Let R be the complement of the outer face of G, and let σ be any directed path from u_1 to the sink of G. The concatenation of π and σ divides R in two (not necessarily connected) regions, which we call *left* and *right* (with the direction of $\pi\sigma$ being taken as "forwards"). The path π' cannot cross σ (otherwise the two would give rise to a cycle), and its first edge α' lies in the left region; therefore, after leaving w the path π' must lie entirely in the left region, and in particular e_2 precedes e_1 in the counterclockwise order around u_1. By repeating this argument for all possible edges $e_2 \neq e_1$ into u_1, we conclude that $e_1 = \text{last in}(u)$. □

Therefore, instead of retrieving e from the stack in step 9, we can simply set $e \leftarrow \text{last in}(u)$. Note also that the test "$e \neq \text{base}(G)$ and v has not been visited yet" in step 3 is evaluated and yields true if and only if e is pushed into Q by the following step, so we can replace that test by the condition "$e \neq \text{base}(G)$ and $e = \text{last in}(v)$". These observations enable us to dispense with the recursion stack and the "visited" bits on the vertices.

A representation for the embedded graph G that allows the efficient computation of first out and its companions is the *quad-edge* data structure [GS]. This representation is similar to the well-known "winged edge" and DCEL data structures [B], [MP], but has over them the important advantage of encoding simultaneously both G and its dual embedding, in precisely the same format, at a negligible extra cost in storage. This allows the post-order traversals of both S and S^* to be performed by the same procedure, applied to the same data structure.

The quad-edge structure by itself can only represent an *undirected* embedded graph, such as the undirected subdivision $\mathcal{S}$. However, every edge $\bar{e}$ of $\mathcal{S}$ is represented by two distinct records in the structure, corresponding to the two possible orientations of $\bar{e}$. Therefore, to refer to the edge $\bar{e}$ of the structure we must actually refer to a specific *directed* version of $\bar{e}$. Given such a directed edge e, the quad-edge data structure gives immediate access to:

org (e) the origin vertex of e,
dest (e) the destination vertex of e,
onext (e) the next counterclockwise directed edge with the same origin,
dnext (e) the next counterclockwise directed edge with the same destination,
sym (e) the same edge directed the other way, and
rot (e) the dual of the edge e, directed from the right to the left faces of e.

A *directed graph* G, such as S or S^*, can be represented by the quad-edge encoding of the corresponding undirected graph, plus a predicate forward (e, G) that tells whether the directed edge e of the structure is actually an edge of G. Clearly, e is in G if and only if sym (e) is not in G, so forward (e, G) $\equiv \neg$forward (sym (e), G). In our case, forward (e, S) is simply the test of whether the x-coordinate of dest (e) is smaller than that of org (e). Similarly, in the dual graph S^* the predicate forward (e, S^*) tests whether the region dest (e) is immediately below the region org (e). This turns out to be the same as $\neg$forward (rot (e), S). The dummy edges that we must add to S and S^* are the only exception: we have rot (base (S^*)) = base (S), and yet both are

forward. For both graphs, we also have

$$e = \text{last in}(\text{dest}(e)) \Leftrightarrow \text{forward}(e, G) \wedge \neg\text{forward}(\text{dnext}(e), G),$$

$$e = \text{last out}(\text{org}(e)) \Leftrightarrow \text{forward}(e, G) \wedge \neg\text{forward}(\text{onext}(e), G),$$

$$e = \text{last in}(u) \Rightarrow \text{sym}(\text{dnext}(e)) = \text{first out}(u).$$

These identities allow us to remove also the calls to first out and its companions from Algorithm 4. Algorithm 5 below incorporates all these modifications.

ALGORITHM 5. *Post-order traversal of an acyclic planar graph with one source and one sink using $O(1)$ auxiliary storage.*

1. Set $u \leftarrow \text{dest}(\text{base}(G))$ and $e \leftarrow \text{sym}(\text{dnext}(\text{base}(G)))$.
2. Repeat the following steps:
   {*At this point u is unvisited,* forward (e, G) *is true, and e is the first unvisited edge out of u.*}
   3. While $e \neq \text{base}(G) \wedge \neg\text{forward}(\text{dnext}(e), G)$, do
      {*Here e is the last edge into its destination v, so v has not been visited yet.*}
      4. Set $u \leftarrow \text{dest}(e)$, and set $e \leftarrow \text{sym}(\text{dnext}(e))$.
      {*This sets e to* first out (u).}
   {*Now e is the dummy edge, or its destination has already been visited.*}
   5. If $e \neq \text{base}(G)$, visit e.
   6. While $\neg\text{forward}(\text{onext}(e), G)$ do
      {*Here e has been visited and is* last out (u).}
      7. Visit u.
      8. If $u = \text{dest}(\text{base}(G))$, the algorithm terminates. Else,
         {*Compute* last in (u), *and backtrack through it.*}
         9. Set $e \leftarrow \text{sym}(\text{onext}(e))$.
            While $\text{forward}(\text{dnext}(e), G)$, do $e \leftarrow \text{dnext}(e)$.
            Set $u \leftarrow \text{org}(e)$.
      10. Visit e.
   11. Set $e \leftarrow \text{onext}(e)$.

The equivalence between Algorithms 4 and 5 is straightforward. It is also easy to show that the latter runs in $O(m)$ time for a subdivision with m edges. Every edge of the graph is assigned to e at most twice: once in the enumeration of the edges out of a vertex v (step 11), and once in the determination of last in (v) (step 9).

13. Conclusions and applications. We have introduced a new data structure, the layered dag, which solves the point location problem for a monotone subdivision of the plane in optimal time and space. The main idea has been to refine the chains introduced by Lee and Preparata and connect the refined chains by links. The latter concept originates with Willard [W] and has found frequent application since then. The layered dag can be built from standard subdivision representations in linear time, as follows. We use the graph traversal algorithm of § 12 to enumerate the regions of the subdivision in a way compatible with the vertical ordering presented in § 4. Another traversal of the subdivision then allows us to build the chain tree representing a complete family of separators, as in § 10. Finally, the layered dag is built from the chain tree, as explained in § 9. The point location algorithm using this structure has

been given in § 8. Compared to previous optimal solutions, the advantage of the layered dag is that

- it admits a simple, practical implementation, and
- it can be extended to subdivisions with curved edges.

We will not discuss in detail here how to generalize our method to work for curved-edge subdivisions. Certain requirements for such a generalization to work are clear. We must be able to break up edges into monotone pieces, to introduce the additional edges required by regularization, and to test on what side of (a monotone segment of) an edge a point lies. Our time bounds will be maintained as long as we are able to in constant time:

- cut an edge into monotone pieces,
- add a monotone regularization edge between two existing monotone edges, and
- test if a point p is above or below a monotone edge e.

The layered dag also yields improved solutions for several other problems in computational geometry. All these problems are reduced to the subdivision search problem treated earlier. For example, subdivisions with circular edges occur in the *weighted Voronoi diagram* of a point set [AE]. There, each point p in a finite set U has associated a positive weight $w(p)$ and the region $R(p)=\{x \mid d(x, p)/w(p) \leqq d(x, q)/w(q)$, for all $q \in U\}$. The layered dag offers the first optimal method for locating a point in the diagram defined by these regions.

Finally, certain problems related to windowing a two-dimensional picture given as a collection of line segments have been reduced to subdivision search by Edelsbrunner, Overmars, and Seidel [EO]. The layered dag provides a way to extend their methods to more general curves without losing efficiency.

Acknowledgments. We would like to thank Andrei Broder, Dan Greene, Mary-Claire van Leunen, Greg Nelson, Lyle Ramshaw, and F. Frances Yao, whose comments and suggestions have greatly improved the readability of this paper.

REFERENCES

[AE] F. AURENHAMMER AND H. EDELSBRUNNER, *An optimal algorithm for constructing the weighted Voronoi diagram in the plane*, Pattern Recognition, 17 (1984), pp. 251-257.

[BM] J. L. BENTLEY AND H. A. MAURER, *A note on Euclidean near neighbor searching in the plane*, Inform. Proc. Letters, 8 (1979), pp. 133-136.

[BP] G. BILARDI AND F. P. PREPARATA, *Probabilistic analysis of a new geometric searching technique*, Manuscript, Dept. EECS, Univ. Illinois, Urbana, 1982.

[B] I. C. BRAID, *Notes on a geometric modeller*, C.A.D. Group Document No. 101, Computer Laboratory, Univ. Cambridge, England, July 1979.

[C] R. COLE, *Searching and storing similar lists*, Tech. Rep. 83 New York Univ., New York, 1983.

[C1] B. M. CHAZELLE, *An improved algorithm for the fixed-radius neighbor problem*, Inform. Proc. Letters, 16 (1983), pp. 193-198.

[C2] ———, *How to search in history*, Report CS-83-08, Dept. Computer Science, Brown Univ., Providence, RI, 1983; also Proc. International Symposium on Fundamental Computer Theory, Springer-Verlag, Berlin, 1983.

[C3] ———, *Filtering search: A new approach to query-answering*, Proc. 24th Symposium on the Foundations of Computer Science, 1983, pp. 122-132.

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

[EH] H. EDELSBRUNNER, G. HARING AND D. HILBERT, *Rectangular point location in d dimensions with applications*, Comput. J., to appear.

[EM] H. EDELSBRUNNER AND H. A. MAURER, *A space-optimal solution of general region location*, Theoret. Comput. Sci., 16 (1981), pp. 329-336.

[EO] H. EDELSBRUNNER, M. H. OVERMARS AND R. SEIDEL, *Some methods of computational geometry applied to computer graphics*, Computer Vision, Graphics and Image Processing, 28 (1984), pp. 92-108.

[GS] L. J. GUIBAS AND J. STOLFI, *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*, Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 221-234.

[H] D. HAREL, *A linear time algorithm for the lowest common ancestor problem*, Proc. 21st IEEE Symposium on Foundations of Computer Science, 1980, pp. 308-319.

[Ki] D. G. KIRKPATRICK, *Optimal search in planar subdivisions*, this Journal, 12 (1983), pp. 28-35.

[Kn] D. E. KNUTH, *The Art of Computer Programming, Vol.* 1: *Fundamental Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1975.

[LP] 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.

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

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

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

[PS] F. P. PREPARATA AND K. J. SUPOWIT, *Testing a simple polygon for monotonicity*, Inform. Proc. Lett., 12 (1981), pp. 161-164.

[S] M.I. SHAMOS, *Geometric complexity*, Proc. 7th ACM Symposium on Theory of Computing, 1975, pp. 224-233.

[Ta] R. E. TARJAN, *Depth first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146-160.

[W] D. E. WILLARD. *New data structures for orthogonal queries*, this Journal, 14 (1985), pp. 242-253.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics

CONSTRUCTING ARRANGEMENTS OF LINES AND HYPERPLANES WITH APPLICATIONS*

H. EDELSBRUNNER†, J. O'ROURKE‡, AND R. SEIDEL§

Abstract. A finite set of lines partitions the Euclidean plane into a cell complex. Similarly, a finite set of $(d-1)$-dimensional hyperplanes partitions d-dimensional Euclidean space. An algorithm is presented that constructs a representation for the cell complex defined by n hyperplanes in optimal $O(n^d)$ time in d dimensions. It relies on a combinatorial result that is of interest in its own right. The algorithm is shown to lead to new methods for computing λ-matrices, constructing all higher-order Voronoi diagrams, halfspatial range estimation, degeneracy testing, and finding minimum measure simplices. In all five applications, the new algorithms are asymptotically faster than previous results, and in several cases are the only known methods that generalize to arbitrary dimensions. The algorithm also implies an upper bound of 2^{cn^d}, c a positive constant, for the number of combinatorially distinct arrangements of n hyperplanes in E^d.

Key words. arrangements, configurations, geometric transformation, combinatorial geometry, computational geometry, optimal algorithm

1. Introduction. Let H denote a finite set of lines in the Euclidean plane E^2. H determines a partition of the plane called the *arrangement $A(H)$ of H* or the *cell complex induced by H*. $A(H)$ consists of *vertices* (intersections of lines), *edges* (maximal connected components of the lines containing no vertex), and *regions* (maximal connected components of E^2 containing no edge or vertex). All geometric entities of this paper will be defined in Euclidean space which should make clear that our discussion does not take the projective view. However, no essential use is made of the concept of distance which implies that all results, but the ones on minimum measure simplices in § 4.5, also hold in real affine space.

Arrangements of lines have been studied from various mathematical points of view since Steiner [St] in 1826. The first attempt to provide a systematic exposition of the subject was made in 1967 in Grünbaum [G1], and to a more exhaustive extent five years later in Grünbaum [G2]. In spite of the extensive literature on arrangements of lines, there is a host of easily formulated but unsolved questions in this area. The interested reader is referred to [G2] where many open conjectures are stated. Recent advances on questions posed in [G2] are reported e.g. in Goodman and Pollack [GP1] and Edelsbrunner and Welzl [EW3].

The notion of a two-dimensional arrangement is easily generalized to three and higher dimensions. There, H is a finite set of $((d-1)$-dimensional) hyperplanes in the d-dimensional Euclidean space E^d. The arrangement $A(H)$ consists of open convex d-dimensional polyhedra and various relatively open convex k-dimensional polyhedra bounding them, for $0 \leqq k \leqq d-1$.

Arrangements in E^d, for $d \geqq 3$, have received considerably less attention in the mathematical literature than arrangements in E^2. We refer to Grünbaum [G1, Chap. 18], and Grünbaum [G3] for surveys of d-dimensional arrangements. A reason for the lack

* Received by the editors September 15, 1983, and in revised form November 5, 1984.

† Institutes for Information Processing, Technical University of Graz, A-8010 Graz, Austria. The work of this author was supported by the Austrian Fonds zur Förderung der wissenschaftlichen Forschung.

‡ Department of Electrical Engineering and Computer Science, The Johns Hopkins University, Baltimore, Maryland 21218. This research was conducted while this author visited the Technical University of Graz. The work of this author was supported in part by the National Science Foundation under grant MSC-8117424.

§ Computer Science Department, Cornell University, Ithaca, New York 14583. This research was conducted while this author visited the Technical University of Graz.

of attention is probably the difficulty of visualizing arrangements even in E^3. Furthermore, Goodman and Pollack [GP2] demonstrated that a tool that is useful in E^2 (the "Levi enlargement lemma," see e.g. [G2]) does not generalize to E^3.

Much of the significance of arrangements in E^d is due to a dual correspondence to configurations of points in E^d. Many problems for sets of points are more conveniently solved for the corresponding arrangement. Examples for this thesis are the computational geometry problems discussed in § 4. Additional significance stems from the correspondence of arrangements in E^d to a special type of polytopes, called zonotopes, in E^{d+1} that can be defined as the Minkowski sum of segments (see e.g. [G3]).

The purpose of this work is to describe an optimal algorithm for constructing arrangements in E^d, for $d \geqq 2$. The optimality of the algorithm follows from the fact that the time required to construct an arrangement does not exceed asymptotically the space needed to store it. To be more specific: An arrangement of n hyperplanes in E^d, for $d \geqq 2$, is constructed in $O(n^d)$ time, and the arrangement actually needs space proportional to n^d unless it is highly degenerate; see e.g. Grünbaum [G1], [G3], Zaslavsky [Z], and Alexanderson and Wetzel [AW]. (For $d = 1$, the arrangement is essentially a sorted set of points on a line and cannot be constructed faster than in $O(n \log n)$ time.)

The optimality of the algorithm relies heavily on a nontrivial combinatorial fact that appears to be new (Thms. 2.7 and 2.8). This fact and other geometric preliminaries are demonstrated in § 2. Section 3 outlines the algorithm for constructing arrangements. In § 4, applications of the algorithm to λ-matrices, halfspatial range estimation, Voronoi diagrams, degeneracy tests, and minimum measure simplices are demonstrated. Finally, § 5 reviews the main results and lists some open problems.

2. Geometric fundamentals. This section discusses properties of arrangements of hyperplanes and configurations of points. It falls into three parts. Section 2.1 is devoted to a geometric transform that realizes the duality between arrangements and configurations; that will be exploited in the applications of § 4. Section 2.2 lists rather straightforward properties of arrangements that are relevant for the algorithmic part of this paper, §§ 3 and 4. Finally, § 2.3 presents a combinatorial result that is the key to the optimality of the algorithm outlined in § 3.

2.1. Arrangements and configurations. Let h be a nonvertical hyperplane in E^d, for $d \geqq 2$, that is, h is a $(d-1)$-dimensional hyperplane that intersects the dth coordinate axis in a unique point. Then the points on h with coordinates $x_1, \cdots, x_d$ satisfy a relation of the form $x_d = h_1 x_1 + \cdots + h_{d-1} x_{d-1} + h_d$. Let $p = (p_1, \cdots, p_d)$ be a point in E^d. We say that p is *above, on,* and *below* h if p_d is greater than, equal to, and smaller than $h_1 p_1 + \cdots + h_{d-1} p_{d-1} + h_d$. Let T be the geometric transform that maps the hyperplane h into the point $T(h) = (h_1, \cdots h_d)$ and the point p into the hyperplane $T(p)$ whose points $(x_1, \cdots x_d)$ satisfy $x_d = -p_1 x_1 - \cdots - p_{d-1} x_{d-1} + p_d$. Where convenient, we will also use the natural extension of T to sets of hyperplanes and sets of points. One of the significant properties of T is that it preserves the relative positions of h and p.

Observation 2.1. If p is above, on, or below h then $T(h)$ is below, on, or above $T(p)$ respectively.

This observation establishes that T leads to dual and order preserving arrangements of hyperplanes if applied to configurations of points and vice versa. This duality of T has found applications to computing intersections of halfspaces (Brown [B]) and other tasks [EMPRWW], [EW2], [O].

We continue the development with an implication of Observation 2.1. First, some definitions are introduced. A set in E^d is called a *subspace of dimension k* (or a *k-flat*), for $0 \leqq k \leqq d$, if there are $d-k$ hyperplanes (and no fewer) such that the set is the intersection of these hyperplanes. Thus, E^d is a d-flat, each hyperplane is a $(d-1)$-flat, and, for convenience, the empty set is said to be a (-1)-flat. (The terms "points," "lines," and "planes" are used to designate 0-flats, 1-flats, and 2-flats in E^2 or E^3.)

Observation 2.2. Let S be a set of points in E^d, and let H be the set of all hyperplanes containing S. Then $T(H)$ is the intersection of all hyperplanes $T(p)$, for p in S.

For the reader particularly interested in the transform T, we note that a rather extensive list of similar implications restricted to E^2 can be found in Goodman and Pollack [GP3].

2.2 Properties of arrangements. Let $H = \{h_1, \cdots, h_n\}$ denote a set of n nonvertical hyperplanes in E^d, for $d \geqq 2$; since they are nonvertical, none contains a 1-flat parallel to the dth coordinate axis. Let h_i^+ and h_i^- denote the open halfspaces above and below h_i, for $1 \leqq i \leqq n$. The arrangement $A(H)$ consists of *faces f* with

$$(*) \qquad f = \bigcap_{1 \leqq i \leqq n} s_f(h_i),$$

where $s_f(h_i)$ is either h_i, h_i^+, or h_i^-. Thus, each face f can be assigned its *intersection word* $w(f) = w_1 \cdots w_n$, with $w_i = 0$, $+$, or $-$ depending on whether $s_f(h_i)$ is h_i, h_i^+, or h_i^-. f is called a *k-face*, for $0 \leqq k \leqq d$, if the affine hull of f is a k-flat. (The *affine hull of* a set X is the collection of points of the form $\sum_{i=1}^m a_i x_i$ with a_i real, $\sum_{i=1}^m a_i = 1$, and x_i in X, for $0 \leqq i \leqq m$.) The terms "vertices," "edges," and "regions" are synonymous with 0-faces, 1-faces, and 2-faces for arrangements in E^2 and E^3. If f is a k-face, then $w(f)$ contains $d-k$ 0's, for $k = d-1, d$, and at least $d-k$, for $0 \leqq k \leqq d-2$. A k-face g and a $(k-1)$-face f are said to be *incident* if f is contained in the closure of g, for $1 \leqq k \leqq d$. Thus, $w(f)$ matches $w(g)$ up to a positive number of letters which are 0 in $w(f)$. Also g is termed a *superface of f* and f is called a *subface of g*. (To avoid confusion, we say that f is a subface of g (or g a superface of f) only if the dimensions of f and g differ by one. A synonym for subface is facet.)

$A(H)$ is called *simple* if the intersection of any k hyperplanes is a $(d-k)$-flat, for $1 \leqq k \leqq d+1$. Observe that this condition excludes parallelism between any two subspaces unless one contains the other. If $A(H)$ is simple, then f is a k-face if and only if $w(f)$ contains exactly $d-k$ 0's for $0 \leqq k \leqq d$. Thus, a face f is subface of a face g if and only if $w(f)$ and $w(g)$ differ in exactly one letter which is 0 in $w(f)$.

It will be necessary to have a system of notation to describe the relationship between the faces of an arrangement and a new hyperplane not part of the arrangement. The reader who is less interested in the algorithm for constructing arrangements may skip the introduction of this notation as well as Lemmas 2.3 and 2.4. Let $H = \{h_1, \cdots, h_n\}$ denote a set of n nonvertical hyperplanes in E^d, and let h denote a nonvertical hyperplane not in H. We assign to each face f of $A(H)$ one of the colours white, red, black, and grey, depending on its relationship to h:

f is *black* if h contains f,

f is *red* if h intersects f but does not contain f,

f is *grey* if h does not intersect f but intersects the closure of f, and

f is *white* if h does not intersect the closure of f.

The nonwhite faces in $A(H)$ are exactly those that are involved in updates if h is to be added to the arrangement. Using the introduced notation, we present a few basic properties of faces in arrangements.

LEMMA 2.3. *Let $A(H)$ be an arrangement in E^d, f a subface of face g in $A(H)$, and h a nonvertical hyperplane not in H.*

(i) *Only the pairs of colours indicated by* 1*'s in Table* 2-1 *can occur.*

(ii) *If g has two black subfaces, then g is black.*

(iii) *g is red if and only if it is a* 1*-face that intersects h, it has a red subface, or it has grey subfaces on both sides of h.*

(iv) *If g is a* 1*-face, then only the circled* 1*'s in Table* 2.1 *hold.*

TABLE 2.1
Matching colours.

f \ g	white	grey	red	black
white	①	①	①	0
grey	0	1	1	0
red	0	0	1	0
black	0	①	0	①

The proof of Lemma 2.3 is omitted as straightforward arguments using (*) and the definitions imply the assertion. The same is true for the proof of the next lemma. Let $A(H)$ and h be as above. We call a face f in $A(H \cup \{h\})$ *blue* if it is contained in h but was not present in $A(H)$.

LEMMA 2.4. *Let $A(H)$ and h be as in Lemma* 2.3 *and let g be a red k-face in $A(H)$, for some k with $1 \leqq k \leqq d$.*

(i) *$g \cap h^+$ and $g \cap h^-$ are k-faces in $A(H \cup \{h\})$, and $g \cap h$ is a blue $(k-1)$-face in $A(H \cup \{h\})$.*

(ii) *A $(k-1)$-face f of $A(H \cup \{h\})$ is a subface of $g \cap h^+$ if and only if either* (1) *f is a white or grey subface of g above h,* (2) *$f = f' \cap h^+$, for a red subface f' of g, or* (3) *$f = g \cap h$. The symmetric statements hold for $g \cap h^-$.*

(iii) *A $(k-2)$-face f'' is incident with $g \cap h$ if and only if either* (1) *$f'' = f' \cap h$, for a red subface f' of g, or* (2) *f'' is a black face in $A(H)$ incident with a grey subface of g.*

For the analysis of the algorithms to follow in § 3, the cardinalities of several sets of faces and incidences are of interest. Let $C_k(H)$ denote the number of k-faces of $A(H)$, for $0 \leqq k \leqq d$, and let $I_k(H)$ be the number of incidences between k-faces and $(k+1)$-faces of $A(H)$, for $0 \leqq k \leqq d-1$. We prove below that both $C_k(H)$ and $I_k(H)$ are in $O(n^d)$, if n denotes the number of hyperplanes in H.

LEMMA 2.5. *Let H be a set of n hyperplanes in E^d. Then*

(i) $C_k(H) \leqq \sum_{i=d-k}^{d} \binom{i}{d-k}\binom{n}{i}$, *for $0 \leqq k \leqq d$, and*

(ii) $I_k(H) \leqq 2(d-k) \sum_{i=d-k}^{d} \binom{i}{d-k}\binom{n}{i}$, *for $0 \leqq k \leqq d-1$.*

Equality occurs if $A(H)$ is simple.

Proof. Part (i) of the assertion follows from the exact formula for simple arrangements, e.g. given in Zaslavsky [Z] or Alexanderson and Wetzel [AW], and the fact that the number of k-faces is maximized when $A(H)$ is simple.

Observe now that a k-face f that is contained in i hyperplanes (so $i \geqq d-k$) has at most $2\binom{i}{d-k-1}$ incident $(k+1)$-faces. This follows from the fact that the i hyperplanes define at most $\binom{i}{d-k-1}$ $(k+1)$-flats each containing two superfaces of f. But as f represents $\binom{i}{d-k}$ k-faces (the maximal number of k-faces created by i hyperplanes), there are at most $2(d-k)$ $(k+1)$-faces for each k-face that f represents. The maximum $2(d-k)$ is achieved if $i = d-k$, which implies that $I_k(H)$ is maximal if $A(H)$ is simple. This completes the argument.

Let now v be a 0-face in an arrangement $A(H)$. Then the number of 1-faces incident with v is called the *degree* $\deg(v)$ *of* v.

LEMMA 2.6. *Let H be a set of n hyperplanes in E^d and let V denote the set of 0-faces contained in a 1-flat of $A(H)$. Then $\sum_{v\in V} \deg(v) = O(n^{d-1})$.*

Proof. There are at most $\binom{n}{d-1}$ 1-flats in $A(H)$. One of these 1-flats can intersect each of the other 1-flats at most once. Hence, $\sum_{v\in V} \deg(v) < 2\binom{n}{d-1} = O(n^{d-1})$.

2.3. Combinatorial results. The combinatorial results demonstrated in this subsection are crucial for the algorithms in §§ 3 and 4. They appear to be new and are of independent interest. We start with the introduction of some notation.

Let $H = \{h_0, h_1, \cdots, h_n\}$ denote a set of $n+1$ nonvertical hyperplanes in E^d. For convenience, h_0 is assumed to coincide with the hyperplane spanned by the first $d-1$ coordinate axes. In § 3, h_0 will play the role of the new hyperplane added to the existing arrangement formed by $h_1, \cdots, h_n$. A d-face g in $A(H)$ is said to be *active* (*with respect to* h_0) if g is above h_0 and the closure of g intersects h_0. Note that g is active if and only if it is contained in a grey or red face (with respect to h_0) in $A(H-\{h_0\})$. Extending the notion of an incidence, a k-face f, for $0 \leqq k \leqq d-1$, is said to *bound* d-face g if f is contained in the closure of g. We call the pair (f, g) a *k-border* (*of* g); often the d-face g will not be explicitly mentioned when it is irrelevant or clear from the context. Where convenient, a flat is said to contain (f, g) if it contains f, and also f is said to contain (f, g). The intersection of all open halfspaces containing g that are defined by hyperplanes in H containing f is termed the *cone of* (f, g). In a two-dimensional arrangement, the cone of a vertex is a wedge with apex at the vertex, and the cone of an edge is a halfplane with the edge on its boundary. The *k-degree* $\deg_k(g)$ of g is now defined as the number of k-faces that bound g, for $0 \leqq k \leqq d-1$. The sum of the k-degrees of all active d-faces in $A(H)$ is denoted by $S_k^d(H, h_0)$. These definitions are illustrated in Fig. 2.1, which shows the regions active with respect to the horizontal line h_0. In the arrangement depicted, $S_0^2(H, h_0) = 17$ and $S_1^2(H, h_0) = 19$.

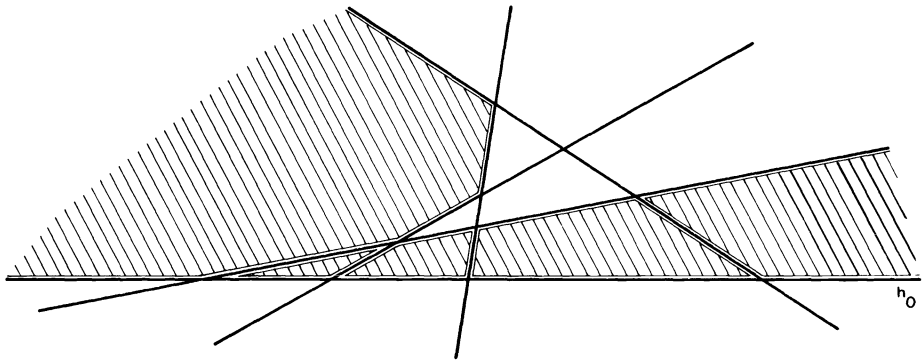


FIG. 2.1. *Regions active with respect to h_0.*

We call a k-border (f, g) *active* if g is active. So $S_k^d(H, h_0)$ counts the number of active k-borders (rather than the number of k-faces that bound active d-faces). For $0 \leqq k \leqq d-1$, define $S_k^d(n) = \max\{S_k^d(H, h_0)$: H a set of $n+1$ nonvertical hyperplanes in E^d and h_0 in $H\}$. It is easy to see that there are simple arrangements $A(H)$ of $n+1$ hyperplanes that achieve $S_k^d(n)$. Thus, for deriving upper bounds it suffices to examine simple arrangements.

We prove below that $S_k^d(n)$ is in $O(n^{d-1})$, which permits the insertion of h_0 into $A(H-\{h_0\})$ in $O(n^{d-1})$ time. The algorithm for performing the insertion will be

described in § 3. Since the result is easiest to understand in E^2, this case is considered first and generalized to higher dimensions later. In both cases, the main technique is to sweep the arrangement with a unidirected hyperplane initially coincident with h_0. During the sweep, faces in the hyperplane are classified into three states that change over time. The rules obeyed by these changes are finally exploited to infer bounds on $S_k^d(n)$.

THEOREM 2.7. *$S_0^2(n) = 5n-3$ and $S_1^2(n) = 5n-1$.*

Proof. Let $H = \{h_0, \cdots, h_n\}$ denote a set of $n+1$ nonvertical lines in E^2 such that h_0 coincides with the x_1-axis and $A(H)$ is simple. We first show that $5n-1$ is an upper bound for $S_1^2(n)$ and demonstrate that it is tight. Then we argue that $S_0^2(n) = S_1^2(n) - 2$.

Observe first that the number of active 1-borders contained in h_0 is $n+1$. It remains to show that $4n-2$ is the maximum number of active 1-borders that are not contained in h_0. To this end, we perform a continuous upwards sweep with a horizontal line h. Initially $h = h_0$, and at each point in time h intersects $A(H)$ in a one-dimensional arrangement $A_h(H)$. Let p_i denote the intersection of h with h_i, and let v_i^L and v_i^R denote the 0-borders on p_i in $A_h(H)$. The superscript L indicates that $v_i^L = (p_i, e_L)$, for e_L the segment in $A_h(H)$ to the left of p_i; the superscript R indicates the symmetric situation to the right. Consult Fig. 2.2 for an illustration.

At each point in time, a 0-border v in $A_h(H)$ is in one of three *states*. Let e denote the 1-border in $A(H)$ such that the cone of v in $A_h(H)$ is the intersection of h and the cone of e in $A(H)$. The cone of v in $A_h(H)$ is a horizontal ray within h with endpoint v; the cone of e in $A(H)$ is a halfplane with e on its boundary. Thus e must contain v for the intersection of h and the cone of e to be the cone of v. Define the state of v as follows:

v is *alive* or *live* if e is active.

v is *dead* if there are two lines h_i and h_j in $H - \{h_0\}$ such that the intersection of h_i and h_j is between h_0 and h, e is contained in h_i or h_j, and the cone of e contains the wedge between h_i and h_j that lies entirely above h_0. (Note that death is irreversible.)

Otherwise, v is *sleeping*.

Intuitively, v is sleeping when it traverses a "dead sector" of $A(H)$ and still has the chance to leave it and become alive. In Fig. 2.2, v_3^L and v_1^R are alive, v_3^R, v_2^L, v_2^R, v_4^R, and v_1^L are dead, and v_4^L is sleeping. In the argument below, we watch the states of 0-borders changing from live to dead which allows us to infer results on the number of active 1-borders in $A(H)$. During the sweep of h, the states of the 0-borders in

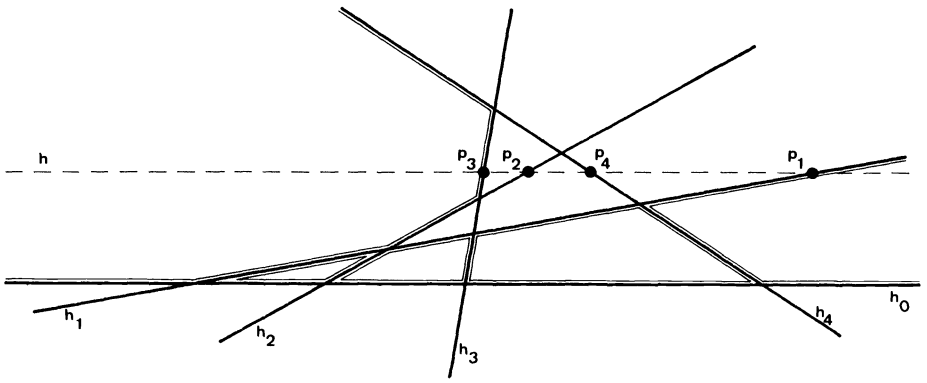


FIG. 2.2. *The bottom-up sweep.*

$A_h(H)$ change only when two points p_i and p_j switch. The following rules can be observed:

R1. All $2n$ 0-borders are alive in the beginning of the sweep.

R2. At least two 0-borders are alive when h has passed all vertices of $A(H)$.

Let now h pass the intersection of h_i and h_j such that p_i is to the left of p_j before they switch. The states of v_i^L and v_i^L after the switch depend only on their states before the switch, and similarly for v_i^R and v_j^R. As the rules for the states of v_i^R and v_j^R are strictly symmetric, we consider only those for v_i^L and v_j^L. The rules observed in each of five cases follow immediately from the definitions of the states "alive," "sleeping," and "dead." Table 2.2 indicates the possible states before and after the switch.

TABLE 2.2

Rule	Before		After	
	v_i^L	v_i^L	v_i^L	v_i^L
R3	alive	alive	dead	alive
R4	dead	alive	dead	sleeping
R5	alive	sleeping	dead	alive
R6	dead	sleeping	dead	sleeping
	dead	dead	dead	dead
R7	sleeping	sleeping	dead	sleeping

The two cases where v_i^L is alive or sleeping and v_j^L is dead have not been enumerated as they cannot occur. Consult Fig. 2.3 for an illustration of rules R3-R7. Living 0-borders are indicated by solid lines, sleeping 0-borders by dashed lines, and dead 0-borders by dotted lines.

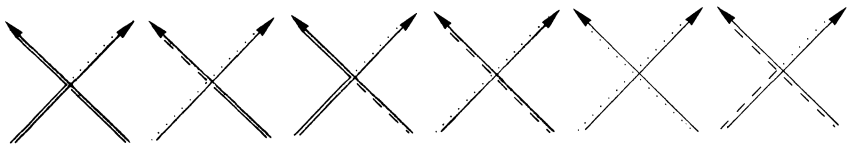


FIG. 2.3. *Illustration of rules* R3-R7.

These rules are exploited for deriving an upper bound on the number of active 1-borders in $A(H)$ that are not contained in h_0. To this end, four counters are used: ACT, to designate the current number of active 1-borders in $A(H)$ that are entirely below h and not contained in h_0, and A, S, and D, to designate the current number of 0-borders in $A_h(H)$ that are alive, sleeping, and dead, respectively. Initially, $ACT = 0$, $A = 2n$, $S = 0$, and $D = 0$ by R1; ultimately, $A \geqq 2$ by R2. Application of the rules R3-R7 causes the following changes to ACT, A, S, and D:

R3: $ACT = ACT + 2$, $A = A - 1$, $S = S$, $D = D + 1$.
R4: $ACT = ACT + 1$, $A = A - 1$, $S = S + 1$, $D = D$.
R5: $ACT = ACT + 1$, $A = A$, $S = S - 1$, $D = D + 1$.
R6: $ACT = ACT$, $A = A$, $S = S$, $D = D$.
R7: $ACT = ACT$, $A = A$, $S = S - 1$, $D = D + 1$.

The transitions flow only from alive to dead (R3) or sleeping (R4), and from sleeping to dead (R5 and R7). Both the alive → dead and the alive → sleeping → dead paths give rise to at most two active 1-borders in $A(H)$. Since $A = 2n$ initially, $4n$ active borders

could be generated. But $A \geqq 2$ after the complete sweep, and each of the remaining live 0-borders is contained in an unbounded active 1-border of $A(H)$. Therefore, $4n-2$ is an upper bound on the number of active 1-borders in $A(H)$ that are not contained in h_0. This shows that $(4n-2)+(n+1)=5n-1$ is an upper bound on $S_1^2(n)$. The arrangement shown in Fig. 2.1 actually realizes equality for $n=4$ and can be generalized to arbitrary n in an obvious way. This shows $S_1^2(n)=5n-1$.

To establish $S_0^2(n) \leqq S_1^2(n)-2$, observe that $\deg_1(r)=\deg_0(r)$ for each bounded region r in $A(H)$, and that $\deg_1(r)=\deg_0(r)+1$ for each unbounded region r in $A(H)$. At least two of the active regions are unbounded, so $S_0^2(n) \leqq S_1^2(n)-2$. In fact, the arrangement (shown in Fig. 2.1) that realizes $S_1^2(n)$ has exactly two unbounded active regions, which implies $S_0^2(n)=S_1^2(n)-2=5n-3$. This completes the proof.

We recently learned that Theorem 2.7 was independently discovered by Chazelle, Guibas, and Lee [CGL]. The proof given in [CGL] is considerably simpler than ours; however, it does not seem to generalize to three and higher dimensions. In fact, the motivation for presenting the proof given above (out of a number of possible proofs) is its generalizability.

It is worth mentioning that the assertion of Theorem 2.7 also holds for families of pseudo-lines. (A *pseudo-line* is an unbounded and connected curve in E^2 such that any two in a given arrangement intersect in exactly one point and cross there.) Consult Grünbaum [G2] for an account of this natural generalization of lines. The proof of Theorem 2.7 for arrangements of pseudo-lines is the same as that for lines except that the sweep is performed with a pseudo-line.

Next, the analogue of Theorem 2.7 in $d \geqq 3$ dimensions will be established. The essential idea in the proof is the same as in the proof of Theorem 2.7: the halfspace above h_0 is swept by a hyperplane h parallel to h_0. The switches of pairs of points are now replaced by switches of d-tuples of $(d-2)$-flats in h, that is, the $(d-1)$-dimensional bounded simplex defined by d $(d-2)$-flats collapses and reappears in mirrored shape. Consult Fig. 2.4, which depicts a switch when h is a plane.

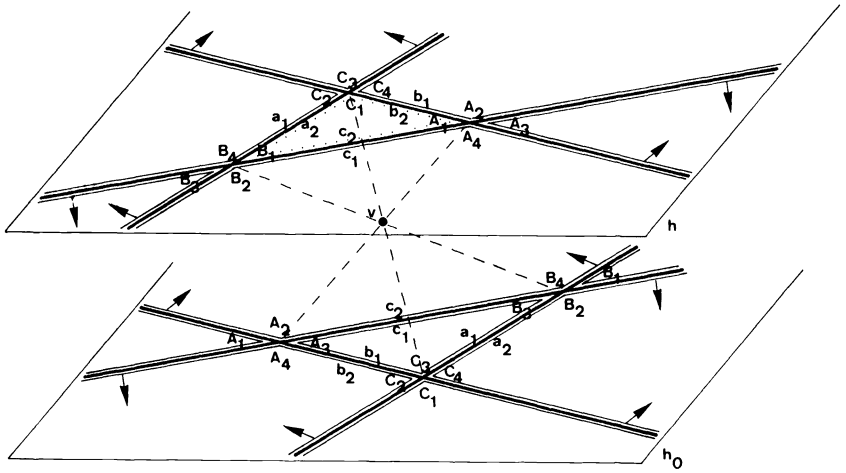


FIG. 2.4. *Switch in* E^3.

THEOREM 2.8. $S_k^d(n)=\theta(n^{d-1})$, *for* $d \geqq 3$ *and* $0 \leqq k \leqq d-1$.

Proof. Let $H=\{h_0, \cdots, h_n\}$ denote a set of $n+1$ nonvertical hyperplanes in E^d such that h_0 coincides with the hyperplane spanned by the first $d-1$ coordinate axes and such that $A(H)$ is simple. The intersection of $A(H-\{h_0\})$ and h_0 is isomorphic

to a simple arrangement of n hyperplanes in E^{d-1}. There are $\theta(n^{d-1})$ k-faces ($0 \leqq k \leqq d-1$) in this arrangement (see Lemma 2.5) and these k-faces bound d-faces of $A(H)$ that are active with respect to h_0. Thus, $S_k^d(n) = \theta(n^{d-1})$, for $0 \leqq k \leqq d-1$, which establishes the asymptotic lower bound of the assertion.

For a proof of the upper bound, we perform an upwards sweep with a hyperplane h parallel to h_0, that is, h sweeps in the direction of the x_d-axis. Initially, $h = h_0$, and at each point in time, h intersects $A(H)$ in a $(d-1)$-dimensional arrangement $A_h(H)$. We define a relation R between the faces of $A(H)$ such that (g, g') in R, for $(k+1)$-faces g and g' and $0 \leqq k \leqq d-1$, if

(i) there is a $(k+1)$-flat p (defined by $d-k-1$ hyperplanes in H) that contains g and g',

(ii) g and g' share a bounding 0-face, and

(iii) there is no hyperplane parallel to h_0 that intersects both g and g'.

We call two faces g and g' *equivalent* if they are in the same equivalence class induced by the transitive closure of R. So all 1-faces of a 1-flat are equivalent, and h intersects exactly one face of each equivalence class unless it contains a 0-face of $A(H)$. The notion of "equivalence" can be extended to borders of $A(H)$ such that two $(k+1)$-borders c and c' are *equivalent* if

(i) the two $(k+1)$-faces g and g' that contain c and c' are equivalent, and

(ii) the cones of c and c' are the same.

Let now b and b' be two k-borders in $A_h(H)$, for $0 \leqq k \leqq d-2$, at different points in time. Let c and c' be the $(k+1)$-borders in $A(H)$ such that the cone of b (and b') is the intersection of h and the cone of c (and c'). We *identify* b and b' if c and c' are equivalent. Consult Fig. 2.4 for an illustration of this identification of borders which is natural when the sweep of h is considered as a process in time.

At each point in time, a k-border b ($0 \leqq k \leqq d-2$) in $A_h(H)$ is in one of three states. Let c denote the $(k+1)$-border in $A(H)$ such that the cone of b (in $A_h(H)$) is the intersection of h and the cone of c (in $A(H)$. Then the state of b is defined as follows:

b is *alive* if c is active.

b is *dead* if there are d hyperplanes in $H - \{h_0\}$ such that their common 0-face lies between h_0 and h, c is contained in the intersection of $d-k-1$ of these hyperplanes, and the cone of c contains the unique sector defined by the d hyperplanes that lies above h_0. (Death is thus irreversible.)

Otherwise, b is *sleeping*.

During the sweep of h, the states of the k-borders in $A_h(H)$ change only when d $(d-2)$-flats in $A_h(H)$ switch (see Fig. 2.4 for a switch of three lines (1-flats) in E^3). The rules for the changes of the states that are observed can be related and reduced to the rules described in the proof of Theorem 2.7: All $\Theta(n^{d-1})$ k-borders in $A_h(H)$, for $0 \leqq k \leqq d-2$, are alive in the beginning of the sweep. Let now h pass the common 0-face v of d hyperplanes $h_{i1}, \cdots, h_{id}$ in H (see Fig. 2.4). There are certain k-borders in $A_h(H)$, for $0 \leqq k \leqq d-2$, that collapse into v as h comes closer to v. We call such a k-border *collapsing*. A collapsing i-border B is *paired* with a collapsing $(d-i-2)$-border b, for $0 \leqq i \leqq d-2$, if the following holds:

(i) there is no hyperplane in H that contains B and b (before they collapse), and

(ii) there is a proper halfspace in h that contains the cone of B and the cone of b.

In the arrangement shown in Fig. 2.4, the following 0-borders and 1-borders are paired: (A_1, a_1), (A_3, a_2), (B_1, b_1), (B_3, b_2), (C_1, c_1), and (C_3, c_2). Note that there are collapsing i-borders in $A_h(H)$ that are not paired. However, each (collapsing) i-border, for $0 \leqq i \leqq d-2$, of the collapsing $(d-1)$-simplex s in $A_h(H)$ is paired with a $(d-i-2)$-border whose cone does not contain s. Let f be a $(d-1)$-face in $A_h(H)$ that shares

a bounding i-face with s, with i maximal and $0 \leqq i \leqq d-2$. Then the i-border b of f contained in that common i-face is paired with some $(d-i-2)$-border B of s. Paired borders play the same role in this proof as v_i^L and v_j^L have done in the proof of Theorem 2.7. Let B and b be two paired borders such that the cone of B contains s and the cone of b contains f (before s collapses). After the collapse, B becomes a border of f, and b becomes a border of s. Hence, b dies in any case, and the new state of B depends on the old states of B and b. The changes of the states of B and b follow exactly the rules R3–R7 (see Table 2.2 and Fig. 2.3). For instance, if b was alive before the collapse then B stays or becomes alive as it belongs now to a $(d-1)$-face (in $A_h(H)$) that is the intersection of h and an active d-face in $A(H)$. If b was not alive before the collapse, then it cannot change the state of B unless B is alive in which case B falls asleep.

Let us now exploit this property for deriving an upper bound on the number of active k-borders in $A(H)$, for $0 \leqq k \leqq d-1$. Note first that an active k-border is created during the sweep of h (that is, the upper end of the k-face that contains the active k-border is passed by h) only when a switch occurs in $A_h(H)$ such that some of the collapsing borders are alive. Furthermore, each collapse creates at most a constant number of active k-borders.

Let g be the d-face in $A(H)$ such that the collapsing $(d-1)$-complex s in $A_h(H)$ is the intersection of g and h. We distinguish two cases: First assume that g is not active. Then there is another $(d-1)$-face f in $A_h(H)$ with the following properties:

(i) Let g' be the d-face in $A(H)$ such that $f = g' \cap h$. Then g' is active and f shares a bounding i-face, (i maximal and $0 \leqq i \leqq d-2$) with s.

(ii) There is a (living) i-border b of f contained in that i-face that is paired with a (nonliving) $(d-i-2)$-border of s.

By rule R3 or R5, b dies. However, this implies that this case can occur only $O(n^{d-1})$ times as each occurrence increases the number of dead borders in $A_h(H)$ by at least one. Second, assume that g is active. As h is passing the topmost point of g (since s is collapsing) and there are only $O(n^{d-1})$ active d-faces in $A(H)$, this case can also occur only $O(n^{d-1})$ times. This completes the proof.

In order to keep the proof of Theorem 2.8 short, we have refrained from deriving more accurate than only asymptotic upper bounds. Nevertheless, we conjecture that the applied proof technique is well suited for calculating more accurate bounds as well. It is worthwhile to note here that Theorem 2.8 also holds for arrangements of pseudo-hyperplanes appropriately defined (see e.g. [GP2]). In this more general setting, the proof of Theorem 2.8 can be adapted by performing a sweep with a pseudo-hyperplane.

There is an interesting consequence of Theorems 2.7 and 2.8:

COROLLARY 2.9. *Let H be a set of n hyperplanes in E^d, let f be a d-face in $A(H)$, and let $\deg_k(f)$ denote the number of k-faces $(0 \leqq k \leqq d-1)$ bounding f. Then the sum of the products $\deg_{d-1}(f)\deg_k(f)$, for all d-faces f in $A(H)$, is in $O(n^d)$.*

Proof. The sum of $S_k^d(H, h)$, for all h in H, is in $O(n^d)$ by Theorems 2.7 and 2.8. Turning $A(H)$ upside-down and repeating the evaluation of $S_k^d(H, h)$ gives again $O(n^d)$. But now, each k-face in $A(H)$ has been counted $\deg_{d-1}(f)$ times for each d-face f that is bounded by the k-face.

3. Constructing arrangements. This section describes algorithms for constructing arrangements in Euclidean spaces. For expository reasons, the algorithm working in E^2 is presented first and the general algorithm later. The next subsection presents the overall structure of the algorithm and the data structure used for representing arrangements.

3.1. The overall structure. The construction of an arrangement proceeds incrementally, that is, the arrangement is built by adding hyperplanes one at a time to an already existing arrangement. The order in which the hyperplanes are added is irrelevant. To avoid tedious special cases that occur for sets of hyperplanes whose normal-vectors do not span E^d, we start with a carefully chosen subcollection of the given set.

Let H denote a set of n hyperplanes $h_1, \cdots, h_n$ in E^d and define $H_i = \{h_1, \cdots, h_i\}$, for $1 \leqq i \leqq n$. $D(H)$ denotes the data structure to be described that represents the arrangement $A(H)$. We assume that the normal-vectors of the hyperplanes in H span E^d. Otherwise, each hyperplane is intersected with the k-dimensional subspace of E^d (with $k < d$) spanned by the normal-vectors, and the resulting k-dimensional arrangement, which captures the essential information of the d-dimensional arrangement, is constructed. We will see that this preprocessing phase only requires $O(n)$ time if d is considered a constant. Now, the overall structure of the algorithm can be described as follows:

Without loss of generality, assume the normal-vectors of $h_1, \cdots, h_d$ span E^d. Construct $D(H_d)$.

For i running from $d+1$ to n, construct $D(H_i)$ from $D(H_{i-1})$ by insertion of h_i.

Finally, $D(H) = D(H_n)$.

Some comments are in order to clarify the preprocessing phase that computes the space spanned by the normal-vectors of the hyperplanes. It is readily seen that this action can be performed in $O(n)$ time by successively testing whether the normal-vector of the current hyperplane is contained in the subspace spanned so far. Let k denote the dimension of the spanned subspace. Then this strategy can also be used to identify k hyperplanes whose normal-vectors span the subspace. Without loss of generality let this subspace be spanned by the last k coordinate axes. Then each hyperplane is replaced by its intersection with this subspace and the arrangement of the resulting hyperplanes in k dimensions is constructed. The method for constructing $D(H_d)$ (or $D(H_k)$) is demonstrated in the next subsection.

3.2. The representation of arrangements. For storing an arrangement $A(H)$, we basically use the incidence lattice of $A(H)$ defined for polytopes in Gruenbaum [G1]. By convention, $A(H)$ is called a $(d+1)$-face and the empty set is called a (-1)-face of $A(H)$. Also $A(H)$ is said to be incident with all its d-faces, and the empty set is said to be incident with all 0-faces. The incidence lattice of $A(H)$ represents each k-face by a node, for $-1 \leqq k \leqq d+1$, and contains connections between nodes of incident faces. Where convenient in the subsequent discussion, no distinction will be made between a node and its corresponding face. See Fig. 3.1 for an arrangement of two lines in E^2 and its incidence lattice.

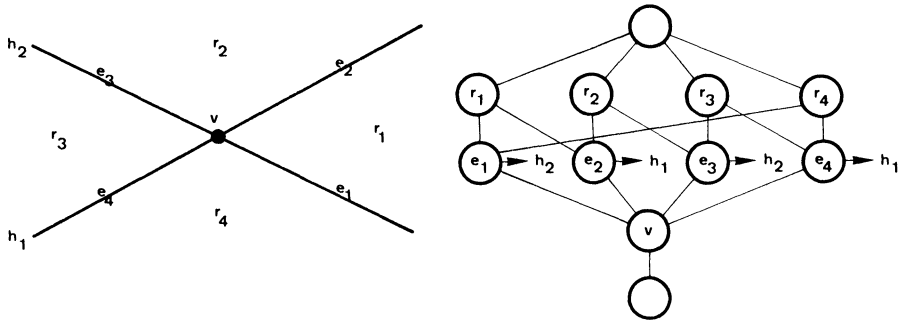


FIG. 3.1. *Arrangement and incidence lattice.*

A point $p(f)$ in f is also associated with each k-face f, for $0 \leqq k \leqq d$. If f is a 0-face, then $p(f) = f$. To be precise, we use the following definition for $p(f)$:

(1) If f is an unbounded 1-face then $p(f)$ is the unique point of f with distance 1 from the only incident vertex.

(2) If f is a bounded 1-face or f is a k-face with $k \geqq 2$, then $p(f) = (\sum_{i=1}^{m} p(f_i))/m$, for $f_1, \cdots, f_m$ the subfaces of f.

In addition, each $(d-1)$-face is associated with its supporting hyperplane. The data structure $D(H)$ is thus the incidence lattice of $A(H)$ augmented with some auxiliary information as described. Without confusion, we will use $A(H)$ and $D(H)$ interchangeably. The auxiliary information used above is not the only choice: it could easily be replaced by equivalent information, such as lists of supporting hyperplanes. The preferred structure depends on the particular application for which the arrangement is being used, and indeed we will further augment the structure when discussing specific problems in § 4.

Before proceeding to the algorithms for building $D(H)$, we discuss the construction of $D(H_d)$ as required in the initial step of the algorithm. We make use of the special structure of $D(H_d)$, which results from the assumption that the normal-vectors of H_d span E^d. Recall that $C_k(H)$ denotes the number of k-faces in $A(H)$, for $0 \leqq k \leqq d$. By definition $C_{-1}(H_d) = C_{d+1}(H_d) = 1$.

LEMMA 3.1. $C_k(H_d) = 2^k\binom{d}{k}$, *for* $0 \leqq k \leqq d$.

The assertion follows from the duality of $A(H_d)$ and the d-dimensional cube in conjunction with Theorem 4.4.2 in [G1]. By "duality" we mean that the incidence lattices of $A(H_d)$ and the cube are isomorphic. The assertion can also be verified directly from the observation that the subarrangement of $A(H_d)$ in one of the hyperplanes is isomorphic to an arrangement defined by $d-1$ hyperplanes in E^{d-1} whose normal-vectors span E^{d-1}. Both facts can be exploited to find the connections to be established between the nodes, thus determining the incidence lattice of $A(H_d)$. The following gives a simple and constructive description of the incidence lattice of $A(H_d)$. We will use the intersection words of the faces as defined in § 2.2.

$A(H_d)$ is a simple arrangement, so for each word w in $\{0, +, -\}^d$ there is a face f with $w(f) = w$. If there are $d-k$ 0's in w then f is a k-face. For a proof of this observe that there are exactly $2^k\binom{d}{k}$ words of length d which contain $d-k$ 0's. But Lemma 3.1 tells us that there are also exactly that many k-faces in $A(H_d)$.

Thus, the incidence lattice of $A(H_d)$ can be set up by creating a node for each word in $\{0, +, -\}^d$. In addition, two nodes representing $A(H_d)$ and the empty set are created. Figure 3.2 shows the incidence lattice of $A(H_2)$ with the intersection word of each node marked.

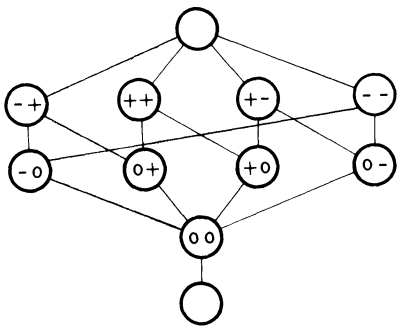


FIG. 3.2. *Incidence lattice of* $A(H_2)$.

Two nodes are connected if their words differ in only one letter, which is 0 in one word. In addition, all d-faces are connected to the $(d+1)$-face and the 0-face is connected to the (-1)-face. It is easy to then augment $D(H_d)$ with the necessary auxiliary information.

3.3. Constructing arrangements of lines. This section describes an algorithm that inserts a line h into an arrangement $A(H)$ of a set H of n lines in E^2. The assumptions on $H \cup \{h\}$ are that no line is vertical and that $A(H \cup \{h\})$ is simple. The strategy for inserting h into $A(H)$ is presented on a rather intuitive level. The full details, including the handling of degenerate cases, can be derived from the general strategy presented in § 3.4. The main purpose of this section is to provide intuition for the explanations in § 3.4.

The insertion of h into $A(H)$ is accomplished in three steps:

Step 1. An edge of $A(H)$ that intersects h is identified.
Step 2. All edges and regions that intersect h are marked red.
Step 3. The marked edges and regions are updated and the new vertices and edges contained in h are integrated.

The three steps are now explained in more detail.

Step 1. To identify an edge e_0 that intersects h, the edges on an arbitrary line of H are visited and tested. The process starts at an arbitrary edge e and proceeds edge by edge closer to h until e_0 is reached.

Step 2. Starting with e_0, all edges and regions that intersect h are marked and remembered in separate storage. To initialize the process, e_0 is marked red and remembered. In addition, the incident regions of e_0 are also marked red, remembered, and put into an empty queue Q. While Q is not empty, the first region r is deleted from Q, and its incident white edges are tested for intersection with h. Those that intersect h are marked red and remembered. Also, if they have an incident region that is not yet marked red then it is marked red and put into Q to await its computation.

Step 3. This step concentrates on splitting each red edge and each red region into two, establishing their new incidences, and integrating the new vertices and edges contained in h into the data structure.

Each red edge e is replaced by two new red edges e_a and e_b representing the parts of e above and below h. Next, the incidences of e_a and e_b are established in the appropriate way. That is: (1) Both are connected to the incident regions of e, and (2) e_a (e_b) is connected to the vertex of e above (below) h. In addition, a blue node v is created that represents $e \cap h$ and is thus connected to e_a, e_b, and the (-1)-face. See Fig. 3.3, where the red nodes are shaded and the blue node cross-hatched.

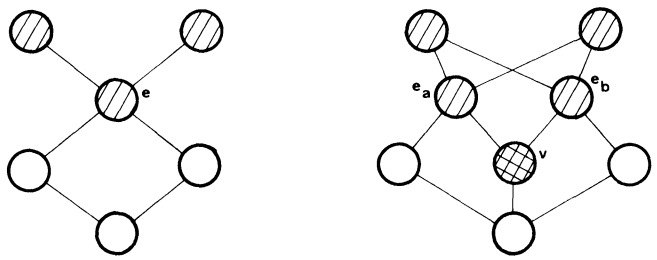


FIG. 3.3. *Updating a red edge e.*

Additional adjustment of incidences is carried out when the red regions are updated. Instead of discussing the update of a red region (which is similar to that of a red edge), we refer to Fig. 3.4, which depicts the actions to be taken in order to split a red region r. Finally, all marked vertices, edges, and regions are unmarked by coloring them white.

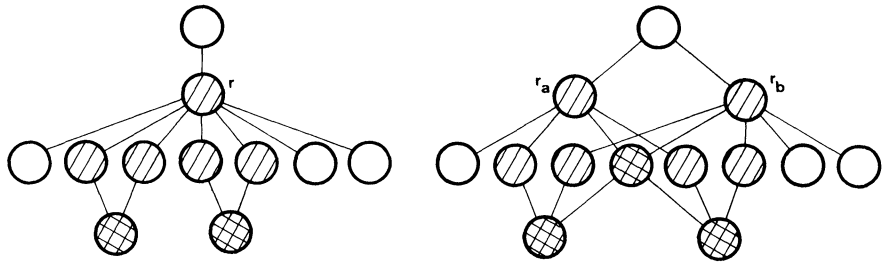


FIG. 3.4. *Updating a red region r.*

We only mention that $O(n)$ time suffices to insert h into $A(H)$; see also Lemma 3.2. Thus, $O(n^2)$ time suffices to set up the arrangement for n lines in E^2; see also Theorem 3.3.

3.4. Constructing arrangements in *d* dimensions. In this section, the insertion of a hyperplane in E^d into an arrangement $A(H)$ of a set H of n hyperplanes is discussed. It is assumed that the normal-vectors of the hyperplanes in H span E^d, as discussed in § 3.1. No restriction on the position of the hyperplanes is assumed except for the exclusion of multiple hyperplanes and of vertical hyperplanes; in particular, the arrangement is not assumed to be simple.

The algorithm that inserts h into $A(H)$ proceeds in three steps:

Step 1. A 1-face in $A(H)$ is identified whose closure intersects h.
Step 2. All faces in $A(H)$ whose closures intersect h are marked black, red, or grey.
Step 3. The marked faces are updated and the new faces contained in h are integrated.

The remainder of this section describes in detail the actions taken in each step and finally gives an analysis of the time and space requirements.

We apply the following strategy to determine a 1-face e_0 of $A(H)$ whose closure intersects h:

Step 1.1. Let w be an arbitrary 0-face and e an incident 1-face not parallel to h. If the closure of e intersects h, then e is the 1-face e_0 required. Otherwise, set v to w if e has only one incident 0-face, and set v to the incident 0-face nearer to h if there are two.

Step 1.2. Let e' (distinct from e) denote the superface of v collinear with e. If the closure of e' intersects h then e' is the 1-face e_0. Otherwise, let v' be the subface of e' distinct from v. Step 1.2 is now repeated with e and v replaced with e' and v'.

Starting with the 1-face e_0, all k-faces in $A(H)$, for $0 \leqq k \leqq d$, whose closures intersect h are marked and remembered in queue Q_k. In addition to the queues $Q_0, \cdots, Q_d$, we use a queue Q to store temporarily those 2-faces that are awaiting examination.

Step 2.1. The 1-face e_0 is marked red if it intersects h, and grey, otherwise. (Note that due to the method of choosing e_0, e_0 is not contained in h.) In addition, e_0 is put

into Q_1. If e_0 is red then all incident 2-faces are marked red and put into Q and Q_2 (Table 2.1, third row and column). If e_0 is grey then its incident 0-face contained in h is marked black and put into Q_0. The incident 2-faces are marked grey, for the moment, and put into Q and Q_2 (Table 2.1, second row and column).

Step 2.2. While Q is not empty, the first 2-face r is deleted from Q. All incident white 1-faces e are tested for intersection with h.

Case 2.2.1. If e is contained in h, then e is marked black and put into Q_1. The white subfaces of e are marked black and put into Q_0. The white superfaces of e are marked grey, for the moment, and put into Q and Q_2 (Table 2.1, fourth row and column).

Case 2.2.2. If e intersects h but is not contained in h, then e is marked red and put into Q_1. All white superfaces are marked red and put into Q and Q_2 (Table 2.1, third row and column).

Case 2.2.3. If e does not intersect h but its closure does then e is marked grey and put into Q_1. The white subface contained in h (if it exists) is marked black and put into Q_0. The white superfaces are marked grey, for the moment, and put into Q and Q_2 (Table 2.1, second row and column).

Case 2.2.4. No action is taken if the closure of e does not intersect h (Table 2.1, first column).

Step 2.3. All grey 2-faces in Q_2 that have either a red subface or grey subfaces above and below h are marked red (Lemma 2.3(iii)). All grey 2-faces which have at least two black subfaces are marked black (Lemma 2.3(ii)).

Step 2.4. For k running from 3 to d and for all faces f in Q_{k-1}, the following actions are taken for the white superfaces of f:

Case 2.4.1. If f is red then they are marked red (Table 2.1, third row).

Case 2.4.2. If f is black, then they are marked black if they have at least two black subfaces, and grey otherwise (Lemma 2.3(ii)).

Case 2.4.3. If f is grey, then they are marked red if they have also red subfaces or grey subfaces above and below h, and grey otherwise (Lemma 2.3(iii)).

In any of the three cases, the examined superfaces of f are put into Q_k.

In Step 2, the nodes which are relevant for structural changes in $D(H)$ have been colored appropriately and stored in queues $Q_0, \cdots, Q_d$. Step 3 performs these changes by replacing each red node by two new ones, establishing their incidences, and integrating the blue faces of $A(H \cup \{h\})$. The steps for updating the auxiliary information in the incidence lattice are not discussed in detail. The only action that is not completely trivial is to provide a blue and unbounded 1-face e with $p(e)$. For this action note that e is contained in a red 2-face r in $A(H)$ that has at least two subfaces e_1 and e_2. The intersection of h with the two 1-flats through $p(e_1)$ and $p(e_2)$, and $2p(e_1)$ and $2p(e_2)$, gives two points on the 1-flat that contains e. $p(e)$ is easily derived from these two points.

Step 3.1. For k running from 1 to d and for each red face f in Q_k the following actions are taken:

Step 3.1.1. In $D(S)$ and in Q_k, f is replaced by two new red faces f_a and f_b representing the parts above and below h (Lemma 2.4(i)). f_a is called an above-node and f_b is called a below-node (see Fig. 3.5).

Step 3.1.2. Each superface of f is connected to both f_a and f_b, as in Fig. 3.5.

Step 3.1.3. Each white or grey subface of f is tested for lying above or below h. (It is convenient to use the auxiliary information for this test.) In the former

case, it is connected to f_a, in the latter case to f_b. Each red subfacial above-node is connected to f_a, and each red subfacial below-node is connected to f_b (Lemma 2.4(ii)); again see Fig. 3.5.

Step 3.1.4. A new blue node f' is created and put into Q_{k-1}. f' represents the new $(k-1)$-face $f \cap h$, and thus, f' is connected to f_a and f_b (Lemma 2.4(ii.3)). In addition, it is connected to the blue subfaces of the red subfaces of f and to the black subfaces of the grey subfaces of f (Lemma 2.4(iii)); see Fig. 3.5. (If f' is a 0-face, then it is connected to the (-1)-face.)

Step 3.2. Finally, $Q_0, \cdots, Q_d$ are emptied and all black, red, grey, and blue nodes are unmarked by coloring them white.

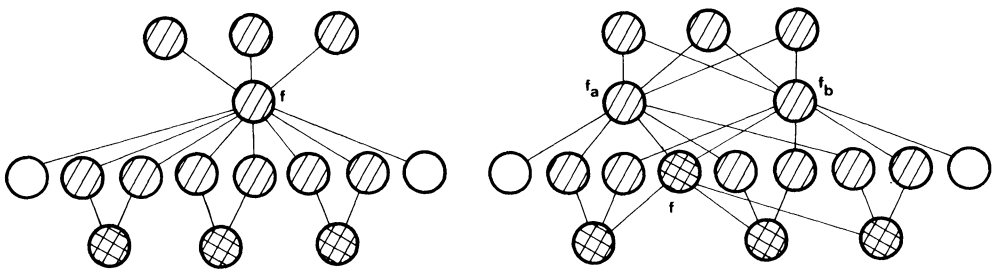


FIG. 3.5. *Updating a red face f.*

This completes the description of the algorithm. It is worthwhile to note that the same algorithm can also be used to construct cell complexes defined by a set of pseudo-hyperplanes. In this case, however, assumptions on the computability of intersections of the pseudo-hyperplanes must be made. We now turn to the analysis of the time requirements.

LEMMA 3.2. *Let H be a set of n hyperplanes in E^d and h be a hyperplane not in H. Then the above algorithm constructs $D(H \cup \{h\})$ in $O(n^{d-1})$ time from $D(H)$.*

Proof. It is trivial to implement Step 1 such that the time needed is proportional to the sum of deg (v), for all 0-faces v examined. This sum is in $O(n^{d-1})$ by Lemma 2.6.

A tedious look at Steps 2 and 3 reveals that the time required is proportional to the number of incidences of all faces in $A(H \cup \{h\})$ whose closures intersect h. Let inc be such an incidence between a k-face g and a $(k-1)$-face f, for some k with $0 \leqq k \leqq d$. Observe that the closure of g intersects h whereas the closure of f may not (Table 2.1, first row). Assume first that g is contained in h, that is, g and f are contained in h. As $h \cap A(H)$ is an arrangement in $d-1$ dimensions, there are at most $O(n^{d-1})$ incidences of this kind according to Lemma 2.5. Now assume that g is not contained in h and without loss of generality that g is above h. We will show that there are at most $O(n^{d-1})$ incidences of this kind.

For counting purposes, inc is attributed to the unique k-flat p that contains g. Define $H_p = \{h^*: h^* = h' \cap p$, for h' in H such that h' does not contain $p\}$ and define $h_p = h \cap p$. Then the faces in p whose closures intersect h are exactly the faces in $A(H_p \cup \{h_p\})$ that are active with respect to h_p. Thus, the number of incidences attributed to p is at most $S^k_{k-1}(n) = O(n^{k-1})$ by Theorems 2.7 and 2.8. However, there are at most $\binom{n}{d-k} = O(n^{d-k})$ k-flats defined by H, which implies that there are at most $O(n^{d-1})$ incidences attributed to all k-flats in $A(H)$. Summing up for k running from 0 to d gives again $O(n^{d-1})$, which completes the argument.

As shown in the beginning of § 3, the strategy to set up $A(H)$ for a set H of n hyperplanes in E^d is to successively insert the hyperplanes. Thus, we state the main result of this section, which follows directly from Lemma 3.2 and Lemma 2.5.

THEOREM 3.3. *Let H be a set of n hyperplanes in E^d, for $d \geqq 2$. Then the outlined algorithm constructs $A(H)$ in $O(n^d)$ time, and this is optimal.*

4. Applications. The problem of constructing an arrangement of hyperplanes is an underlying task for several applications, five of which are demonstrated in this section. The algorithm introduced in § 3 leads to optimal methods for computing λ-matrices and Voronoi diagrams. It also leads to methods for halfspatial range estimation, degeneracy testing, and finding minimum measure simplices that are faster than those previously known.

It turns out that the first three applications are closely related to the concept of "levels" in arrangements. It is for this reason that we introduce what we call the "ranked representation" of an arrangement, which is essentially the incidence lattice augmented with some additional information stored in the nodes.

Let H be a set of n nonvertical hyperplanes in E^d and let f be an arbitrary k-face in $A(H)$. The *ranks* $a(f)$, $o(f)$, and $b(f)$ *of* f denote the number of hyperplanes strictly above f, containing f, and strictly below f. Clearly, $a(f)+o(f)+b(f)=n$, $o(f)=d-k$, for $k=d-1, d$, and $o(f) \geqq d-k$, for $0 \leqq k \leqq d-2$. $D(S)$ augmented with the ranks of its faces is called the *ranked representation of* $A(H)$. In what now follows, an algorithm is outlined that computes the ranks of each k-face, for $0 \leqq k \leqq d$. The algorithm proceeds in three steps and uses a queue Q.

Step 1. The d-face f_{top} that has no hyperplane above it is identified. This can be done by testing, for each d-face f, whether there is an incident $(d-1)$-face whose supporting hyperplane is above $p(f)$.

Step 2. For each d-face f, the numbers $a(f)$, $o(f)$, and $b(f)$ are computed as follows:

Step 2.1. $a(f_{\text{top}})=o(f_{\text{top}})=0$ and $b(f_{\text{top}})=n$. f_{top} is marked and put into Q.

Step 2.2. If Q is not empty, then the first d-face f is removed from Q and the following actions are taken for each subface f^* of f: Let g denote the superface of f^* different from f. Unless g is already marked, the ranks of g are computed as follows: If f is above and g is below the hyperplane that supports f^*, then $a(g)=a(f)+1$, $o(g)=0$, and $b(g)=b(f)-1$. Otherwise, $a(g)=a(f)-1$, $o(g)=0$, and $b(g)=b(f)+1$. Finally, g is put into Q and Step 2.2 is repeated.

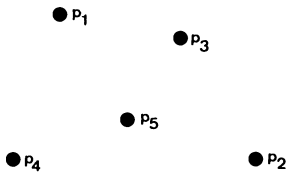
Step 3. For k running from $d-1$ to 0 and for each k-face f, the numbers $a(f)$, $o(f)$, and $b(f)$ are calculated as follows: $a(f)=\min\{a(g)\colon g \text{ superface of } f\}$, and $b(f)=\min\{b(g)\colon g \text{ superface of } f\}$. Finally, $o(f)=n-a(f)-b(f)$.

It is readily seen that this algorithm requires constant time per incidence, which implies that it is in $O(n^d)$ by Lemma 2.5.

4.1. The λ-matrix. Goodman and Pollack [GP4] introduced the λ-matrix of a finite set of points as a generalization of sorting to arbitrary dimensions. Among the applications, they suggest it can be used as a tool in pattern recognition, as it characterizes the set with respect to convexity properties.

Let $(p_1, \cdots, p_{d+1})$ denote a sequence of $d+1$ points in E^d, for $d \geqq 2$, with $p_i=(p_{i,1}, \cdots, p_{i,d})$, for $1 \leqq i \leqq d+1$. The sequence is said to have *positive orientation* $((1, \cdots, d+1)>0)$ if $\det(p_{i,j})>0$, where $p_{i,d+1}=1$, for $1 \leqq i,j \leqq d+1$. $(1, \cdots, d+1)=0$ and $(1, \cdots, d+1)<0$ are analogously defined. As noted in [GP4], $(1,2,3)>0$ if (p_1, p_2, p_3) is oriented counterclockwise, $(1,2,3)=0$ if the points lie on a common line, and $(1,2,3)<0$ if the sequence is oriented clockwise. Let now $S=\{p_1, \cdots, p_n\}$ denote a set of n points in E^d. Then $\lambda(i_1, \cdots, i_d)$ denotes the number of points p_j in S such

that $(i_1, \cdots, i_d, j) > 0$. The *λ-matrix* $\lambda(S)$ *of* S is the d-dimensional matrix with $\lambda(i_1, \cdots, i_d)$ as the element with indices $i_1, \cdots, i_d$ provided the points with indices $i_1, \cdots, i_d$ determine a unique hyperplane containing them. Otherwise, the element is not defined. For $d = 2$, the λ-matrix is a two-dimensional array with entry (i, j) filled with the number of points that fall to the left of the directed line from p_i to p_j. Figure 4.1 shows a set of points in E^2 and the corresponding λ-matrix. (The undefined elements are denoted *w*.) For $d = 3$, the λ-matrix is a three-dimensional array with entry (i, j, k) filled with the number of points that fall to the "positive" side of the plane determined by p_i, p_j, and p_k.



$$\lambda(S) = \begin{pmatrix} w & 1 & 0 & 3 & 2 \\ 2 & w & 3 & 0 & 1 \\ 3 & 0 & w & 2 & 1 \\ 0 & 3 & 1 & w & 2 \\ 1 & 2 & 2 & 1 & w \end{pmatrix}$$

FIG. 4.1. *Point-set and λ-matrix.*

Let $H = T(S)$ using the geometric transform T defined in § 2.1. Furthermore, let $h_i = T(p_i)$, for $1 \leqq i \leqq n$. By Observation 2.2, the points with indices $i_1, \cdots, i_d$ define a unique hyperplane if and only if the intersection of the hyperplanes in H with the same indices is a 0-face v of $A(H)$. In addition, $\lambda(i_1, \cdots, i_d) = a(v)$ if the positive side defined by the points is above $T^{-1}(v)$, and $\lambda(i_1, \cdots, i_d) = b(v)$, otherwise. These explanations suggest that $\lambda(S)$ be computed as follows:

Step 1. Construct the ranked representation of $A(H)$.

Step 2. Associate with each 0-face in $A(H)$ the list of hyperplanes in H which contain it.

Step 3. Derive the elements of $\lambda(S)$ from the ranks of the vertices of $A(H)$.

By now, the details of this strategy should be obvious. Due to Theorem 3.3 and the fact that $\lambda(S)$ consists of n^d elements, we conclude:

THEOREM 4.1. *Let S denote a set of n points in* E^d, *for* $d \geqq 2$. *Then there exists an algorithm which computes* $\lambda(S)$ *in* $O(n^d)$ *time, which is optimal.*

This is an improvement over the $O(n^d \log n)$ time algorithm presented in [GP4].

4.2. Halfspatial range estimation. Let S denote a set of n points in E^3 and let h denote a nonvertical plane. Let $a(h)$ denote the number of points strictly above h. The *halfspatial range search problem* requires that S be stored in a data structure such that for any nonvertical plane h, $a(h)$ can be computed easily. This problem is a generalization of the halfplanar range search problem as considered, e.g., in Willard [W] and Edelsbrunner and Welzl [EW4]. Since there seems to be no solution for the problem (as well as for the one in E^2) that is efficient in both time and space, we consider the following simpler *halfspatial range estimation problem*: S is to be stored such that it is easy to decide for a plane h whether $a(h) < \lceil n/2 \rceil$, or $a(h) \geqq \lceil n/2 \rceil$. The solution to be described below is a generalization of a data structure in Edelsbrunner and Welzl [EW2].

By Observation 2.1, a point p in S is above h if and only if $T(h)$ is below $T(p)$. Let the *K-level of* $A(H)$ (with $H = T(S)$) be the collection of regions (2-faces) r in

$A(H)$, with $a(r) = K - 1$ together with the bounding edges and vertices. Clearly, $a(h) < K$ if and only if $T(h)$ is above or contained in the K-level of $A(H)$. This suggests that the K-level L_K, for $K = \lceil n/2 \rceil$, be used as the basis for our data structure.

Note that L_K intersects each vertical line exactly once and that the projection L'_K of the edges and vertices of L_K onto the x_1x_2-plane gives a planar subdivision defined by straight line edges. Let m denote the number of edges in L'_K. Then there exists a data structure that requires $O(m)$ space and $O(m)$ time for construction from L_K such that $O(\log m)$ time suffices to determine a region of L'_K whose closure includes a query point (see Kirkpatrick [K] or Edelsbrunner, Guibas, and Stolfi [EGS]). Thus, to determine whether or not $T(h)$ is above L_K, we locate projection $T(h)'$ of $T(h)$ in L'_K and then test whether or not $T(h)$ is above the region of L_K that belongs to the located region. This implies:

THEOREM 4.2. *Let S denote a set of n points in E^3 and m the number of edges of L_K, for $K = \lceil n/2 \rceil$. Then there exists a data structure that requires $O(m)$ space such that $O(\log n)$ time suffices to answer a halfspatial range estimation query. $O(n^3)$ time and space is needed to construct the data structure.*

Unfortunately, no upper bound better than $O(n^3)$ is currently known for m. We refer to Erdös, Lovasz, Simmons, and Straus [ELSS] and Edelsbrunner and Welzl [EW1], who derived independently nontrivial bounds for the corresponding quantity in E^2.

Clearly, the notion of a "level" and thus the above method can be generalized beyond three dimensions. Using all K-levels for $1 \leqq K \leqq n$, and binary elimination to determine the one immediately below a query point, yields a solution for the halfspatial range search problem. The complexities are $Q(n^3)$ space and $O(\log^2 n)$ time which matches the best but more general structure by Chazelle [C].

4.3. Order-K Voronoi diagrams. Voronoi diagrams have received a great deal of attention in such diverse areas as geography, archeology, crystallography, physics, mathematics, and computer science. Let S denote a set of n points in E^d, for $d \geqq 2$. Then $V(p) = \{x \text{ in } E^d : d(x, p) < d(x, q), \text{ for } q \text{ in } S - \{p\}\}$ is called the *Voronoi polyhedron of p* in S. The cell complex consisting of the Voronoi polyhedra and the bounding lower dimensional polyhedra is called the *order-1 Voronoi diagram* 1-VOD (S) *of S*. Shamos and Hoey [SH] were the first to describe an optimal algorithm for constructing 1-VOD (S) if S is in E^2. They also introduced "higher-order" Voronoi diagrams: $V(S') = \{x \text{ in } E^d : d(x, p) < d(x, q), \text{ for } p \text{ in } S' \text{ and } q \text{ in } S - S'\}$ is called the *Voronoi polyhedron of S'*. Let K be an integer with $1 \leqq K \leqq n - 1$. Then the cell complex consisting of the Voronoi polyhedra (plus lower dimensional bounding polyhedra) for the subsets S' of S with cardinality K is called the *order-K Voronoi diagram* K-VOS (S) *of S*.

For simplicity, we restrict our attention to E^2; generalizations to three and higher dimensions are straightforward. In a separate paper, [ES], a transformation P is described that relates Voronoi diagrams in E^2 with arrangements of planes in E^3. Each point $p = (p_1, p_2)$ in S is transformed into the plane $P(p)$ that is tangent to the paraboloid $x_3 = x_1^2 + x_2^2$ and touches it in the point $(p_1, p_2, p_1^2 + p_2^2)$. Let L_K denote the K-level of $A(H)$ (with $H = P(S)$) as defined in § 4.2). The vertical projection of the intersection of L_K with L_{K+1} (that is, all 1-faces e with $a(e) = K - 1$ and their endpoints) yield K-VOD (S). The generalization of these considerations implies:

THEOREM 4.3. *Let S denote a set of n points in E^d. Then $O(n^{d+1})$ time suffices to construct all order-K Voronoi diagrams for S, for $1 \leqq K \leqq n - 1$.*

In E^2, K-VOD (S) can be exploited to determine the K closest points to a query point x in $O(\log n + K)$ time. To this end, a region of K-VOD (S) is determined whose

closure includes x. The region r in K-VOD(S) found uniquely determines the K closest points which can, e.g., be stored with r. Thus, the data structures in [K] or [EGS] (this issue, pp. 317-340) yield the result.

Unfortunately, storing the lists of closest points with each region r in each K-VOD(S), for $1 \leqq K \leqq n-1$, increases the space required to $O(n^4)$. If the explicit neighbor lists are required, then $O(n^4)$ is optimal for constructing all higher-order Voronoi diagrams, as Dehne claimed [D]. However, the lists can be encoded into the diagrams as follows:

> Let r denote a region in K-VOD(S), for some $K \geqq 1$. Then r is equipped with a pointer into an arbitrary region r' in $(K-1)$-VOD(S) such that the list of $K-1$ closest points of r' is contained in the list of r. (0-VOD(S) is defined to be E^2 and has an empty list of closest points associated.) The pointer from r to r' is labelled with the one point in the list of r that is missing in the one of r'.

This strategy reduces the space required to $O(n^{d+1})$ and retains the query time of $O(\log n + K)$.

4.4. Degeneracy testing. A set S of $n \geqq d+1$ points in E^d is said to be in *general position* if any subset of $d+1$ points is affinely independent, that is, there is no hyperplane that contains $d+1$ points of S. Recently, van Leeuwen [vL] posed the question whether $O(n^2 \log n)$ time is the best possible time bound for an algorithm that decides whether or not n points in E^2 are in general position. Theorem 3.3 implies that the answer is negative:

THEOREM 4.4. *Let S denote a set of n points in E^d. Then there is an algorithm that decides in $O(n^d)$ time and $O(n^2)$ space whether or not S is in general position.*

Proof. S is in general position if and only if no $d+1$ hyperplanes in H (with $H = T(S)$) intersect in a common point or are normal to a common hyperplane. Construct all two-dimensional subarrangements and determine whether or not any one contains d 1-flats intersecting in a common point or normal to a common 1-flat. There are $O(n^{d-2})$ such subarrangements, and each requires $O(n^2)$ time and space to construct.

4.5. Minimum measure simplices. For simplicity, we confine the discussion in this section to E^2 and thus to minimum area triangles. Generalizations to higher dimensions are straightforward. Let S denote a set of n points in E^2. Any three points p_i, p_j, p_k of S define a triangle TR(i, j, k) with area $m(i, j, k)$. Then MAT(S) = TR(i_0, j_0, k_0) such that $m(i_0, j_0, k_0)$ assumes the minimum is called a *minimum area triangle of S*. We can restrict our attention to S in general position. Otherwise, there are three points on a line that define a triangle with area zero. However, this case can be checked in $O(n^2)$ time by Theorem 4.4.

The problem of finding a minimum area triangle was first considered by Dobkin and Munro [DM] who gave an $O(n^2 \log^2 n)$ time and space algorithm. Later, Edelsbrunner and Welzl [EW2] improved their result to $O(n^2 \log n)$ time and $O(n)$ space. Both approaches are based on:

Observation 4.5. Let MAT(S) = TR(i, j, k). Then p_k is the closest point among $S-\{p_i, p_j\}$ to the line through p_i and p_j.

The line through p_i and p_j corresponds to the intersection of $T(p_i)$ and $T(p_j)$ by Observation 2.1. Furthermore, p_k corresponds to the line $T(p_k)$ immediately above or below (vertically) the intersection. (As two parallel lines have no intersection, S is assumed to contain no two points on a vertical line. Otherwise, S is assumed to contain no two points on a vertical line. Otherwise, S is rotated by an appropriate angle, which

takes $O(n^2)$ time.) The following strategy for computing MAT(S) is suggested by these observations:

Step 1. Construct $A(H)$, with $H = T(S)$.

Step 2. For each region r in $A(H)$ and for each vertex $v = T(p_i) \cap T(p_j)$ on the boundary of r the following actions are taken: Determine each line $T(p_k)$ that contains an edge of r and calculate the area $m(i, j, k)$, provided k is different from i and j. Record the triple (i, j, k) if $m(i, j, k)$ is less than the area of the smallest triangle determined so far.

Obviously, for each vertex $v = T(p_i) \cap T(p_j)$ in $A(H)$, the line $T(p_k)$ immediately above or below v is among the lines tested for v. Not counting the requirements for Step 1, the amount of time required for each region r in $A(H)$ is proportional to the product of $\deg_0(r)$ and $\deg_1(r)$. The sum of these products, over all regions r in $A(H)$, is in $O(n^2)$ by Corollary 2.9. Observing that the algorithm given above as well as all results used in this section generalize to three and higher dimensions, we conclude:

THEOREM 4.6. *Let S denote a set of $n \geq d+1$ points in E^d, $d \geq 2$. Then the minimum measure simplex determined by $d+1$ points in S can be found in $O(n^d)$ time and space.*

This result, and the presented algorithm, were independently discovered for $d = 2$ by Chazelle, Guibas, and Lee [CGL].

5. Discussion. We have presented an optimal method for constructing cell complexes defined by hyperplanes in E^d, basing our algorithm on a new combinatorial result (Theorems 2.7 and 2.8). The result also holds for arrangements of pseudo-hyperplanes. In fact, the algorithm applies to the problem of constructing such more general arrangements, provided that the pseudo-hyperplanes are, in some sense, computationally simple.

Bieri and Nef [BN] described the only existing algorithm known to the authors which computes the faces of an arrangement in E^d. The disadvantage of their algorithm is that it requires more time than ours and does not explicitly establish the incidences between the faces. The significance of the presented optimal method is that there is a host of applications leading to new and faster solutions for problems thought to be unrelated in the past. The five applications shown in § 4 are:

(1) An optimal algorithm for computing λ-matrices for finite sets of points in Euclidean spaces, improving the best result known to date [GP4].

(2) A new data structure and algorithm for halfspatial range estimation for which no sophisticated solution was yet known.

(3) An optimal algorithm for constructing all higher-order Voronoi diagrams. This improves the result of Dehne [D] in E^2 and appears to be the first algorithm known for higher dimensions.

(4) A faster algorithm for testing for degeneracies in a set of points, providing an improvement of existing algorithms and a partial answer to question P20 of [vL].

(5) A faster algorithm for computing minimum measure simplices defined by a set of points. This improves the results of [DM] and [EW2] in E^2 and appears to be the first nontrivial result beyond $d = 2$.

The algorithm also immediately leads to an upper bound on the number of arrangements. For n hyperplanes in E^d, the algorithm takes $O(n^d)$ binary decisions and so can construct only $2^{O(n^d)}$ combinatorially different arrangements. This is also an upper bound for the number of different combinatorial types since the algorithm is not restricted to any subclass of arrangements. This improves the upper bound given in [GP4].

Several open problems are suggested by the results in this paper. Three of the most important ones are as follows:

(1) Let H be a set of n planes in E^3 and let $|L_K(H)|$ denote the number of regions of the K-level $L_K(H)$ of $A(H)$. We define $b_K(n) = \max\{|L_K(H)|: H \text{ a set of } n \text{ planes in } E^3\}$. The authors can show $b_k(n) = \Omega(nk \log k)$, but no nontrivial upper bound is currently available. It is likely that the methods in [ELSS] or [EW1] can be extended in some nontrivial way to obtain an upper bound.

(2) Is $\Omega(n^2)$ a lower bound for deciding whether or not a set of n points in E^2 is in general position? Due to its simple appearance, this computational problem seems to be well suited for a lower bound analysis. Also, there are several problems to which degeneracy testing in E^2 can be reduced. Examples are the minimum area triangle problem of § 4.5, and several geometric problems posed in [LP], [EOW], and [EMPRWW].

(3) There are some geometric problems for which $O(n^2 \log n)$ time solutions are known that might be amenable to applications of Theorem 3.3 to reduce the time to $O(n^2)$. An example is the "shadow problem" of Lee and Preparata [LP]:

> Let S denote a set of n line segments in E^2. Compute a direction (if it exists) such that each line parallel to the direction intersects at most one line segment. If light shines parallel to this direction, none of the shadows overlap.

Acknowledgments. We thank Emmerich Welzl for discussions on Theorem 2.7. We also thank Friedrich Huber for implementing the construction of arrangements in arbitrary dimensions, and Gerd Stoeckl for implementing the algorithms presented in §§ 4.1 and 4.3. The third author wishes to thank Jack Edmonds for the many enlightening discussions.

REFERENCES

[AW] G. L. ALEXANDERSON AND J. E. WETZEL, *Simple partitions of space*, Math. Mag., 51 (1978), pp. 220-225.

[BN] H. BIERI AND W. NEF, *A recursive plane-sweep algorithm, determining all cells of a finite division of R^d*, Computing, 28 (1982), pp. 189-198.

[B] K. Q. BROWN, *Geometric transforms for fast geometric algorithms*, Ph.D. thesis, Rep. CMU-CS-80-101, Dept. Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, 1980.

[C] B. M. CHAZELLE, *How to search in history*, Proc. International Symposium on Fundamental Computer Theory, Springer-Verlag, Berlin, 1983.

[CGL] B. M. CHAZELLE, L. J. GUIBAS AND D. T. LEE, *The power of geometric duality*, Proc. 24th Annual IEEE Symposium of Foundations of Computer Science, 1983, pp. 217-225.

[D] F. DEHNE, *An optimal algorithm to construct all Voronoi diagrams for k nearest neighbor searching in the Euclidean plane*, Proc. 20th Annual Allerton Conference on Communication Control and Computing, 1982.

[DMJ] D. P. DOBKIN AND J. I. MUNRO, private communication.

[EGS] H. EDELSBRUNNER, L. GUIBAS AND J. STOLFI, *Optimal point location in a monotone subdivision*, this issue, pp. 317-340.

[EMPRWW] H. EDELSBRUNNER, H. A. MAURER, F. P. PREPARATA, A. L. ROSENBERG, E. WELZL AND D. WOOD, *Stabbing line segments*, BIT, 22 (1982), pp. 274-281.

[ES] H. EDELSBRUNNER AND R. SEIDEL, *Voronoi diagrams and arrangements*. Rep. 85-669, Dept. Comput. Sci., Cornell Univ., Ithaca, NY, 1985.

[EOW] H. EDELSBRUNNER, M. H. OVERMARS AND D. WOOD, *Graphics in Flatland: a case study*, in Advances in Computing Research, Vol. 1: Computational Geometry, F. Preparata, ed., JAI Press, 1983, pp. 53-59.

[EW1] H. EDELSBRUNNER AND E. WELZL, *On the number of line-separations of a finite set in the plane*, J. Combin. Theory Ser. A., to appear.

[EW2] ———, *Constructing belts in two-dimensional arrangements with applications*, this Journal, to appear.

[EW3] ———, *On the maximal number of edges of many faces in an arrangement*, Rep. F99, Inst. for Information Processing Technical Univ. Graz, Graz, Austria, 1982.

[EW4] ———, *Halfplanar range search in linear space and $O(n^{0.695})$ query time*, Rep. F111, Inst. for Information Processing Technical Univ. Graz, Graz, Austria, 1983.

[ELSS] P. ERDÖS, L. LOVASZ, A. SIMMONS AND E. G. STRAUS, *Dissection graphs of planar point sets*, in A Survey of Combinatorial Theory, J. N. Srivastava et al., eds., North-Holland, Amsterdam, 1973, pp. 139-149.

[GP1] J. E. GOODMAN AND R. POLLACK, *Proof of Grünbaum's conjecture on the stretchability of certain arrangements of pseudolines*, J. Combin. Theory Ser. A, 29 (1980), pp. 385-390.

[GP2] ———, *Three points do not determine a (pseudo-) plane*, J. Combin. Theory Ser. A, 30 (1981), pp. 215-218.

[GP3] ———, *A theory of ordered duality*, Geom. Dedicata, 12 (1982), pp. 63-74.

[GP4] ———, *Multidimensional sorting*, this Journal, 12 (1983), pp. 484-507.

[G1] B. GRÜNBAUM, *Convex Polytopes*, Interscience, London, 1967.

[G2] ———, *Arrangements and spreads*, CBMS Regional Conference Series in Applied Mathematics 10, American Mathematical Society, Providence, RI, 1972.

[G3] ———, *Arrangements and hyperplanes*, Congressum Numerantium III, Louisiana Conference on Combinatrics Graph Theory and Computing, 1971, pp. 41-106.

[K] D. G. KIRKPATRICK, *Optimal search in planar subdivisions*, this Journal, 12 (1983), pp. 28-35.

[LP] D. T. LEE AND F. P. PREPARATA, *Euclidean shortest paths in the presence of rectilinear barriers*, Proc. 7th Conference on Graphtheoretical Concepts in Computer Science, (WG 81), Carl Hanser, 1981, pp. 303-314.

[O] J. O'ROURKE, *On-line algorithms for fitting straight lines between data ranges*, Comm. ACM, 24 (1981), pp. 574-578.

[SH] M. I. SHAMOS AND D. HOEY, *Closest-point problems*, Proc. 16th Annual IEEE Symposium Foundations of Computer Science, 1975, pp. 151-162.

[St] J. STEINER, *Einige Gesetze über die Theilung der Ebene und des Raumes*, J. Reine Angew. Math., 1 (1826), pp. 349-364.

[vL] J. VAN LEEUWEN, *P20*, Bull. EATCS, 19 (1983), p. 150.

[W] D. E. WILLARD, *Polygon retrieval*, this Journal, 11 (1982), pp. 149-165.

[Z] TH. ZASLAVSKY, *Facing up to arrangements: face-count formulas for partitions of space by hyperplanes*, Memoirs Amer. Math. Soc. 154, American Mathematical Society, Providence, RI, 1975.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics

A SIMPLE UNPREDICTABLE PSEUDO-RANDOM NUMBER GENERATOR*

L. BLUM†, M. BLUM‡ AND M. SHUB§

Abstract. Two closely-related pseudo-random sequence generators are presented: The $1/P$ *generator*, with input P a prime, outputs the quotient digits obtained on dividing 1 by P. The $x^2 \bmod N$ *generator* with inputs N, x_0 (where $N = P \cdot Q$ is a product of distinct primes, each congruent to 3 mod 4, and x_0 is a quadratic residue mod N), outputs $b_0 b_1 b_2 \cdots$ where $b_i = \text{parity}(x_i)$ and $x_{i+1} = x_i^2 \bmod N$.

From short seeds each generator efficiently produces long well-distributed sequences. Moreover, both generators have computationally hard problems at their core. The first generator's sequences, however, are *completely predictable* (from any small segment of $2|P|+1$ consecutive digits one can infer the "seed," P, and continue the sequence backwards and forwards), whereas the second, under a certain intractability assumption, is *unpredictable* in a precise sense. The second generator has additional interesting properties: from knowledge of x_0 and N but *not* P or Q, one can generate the sequence forwards, but, under the above-mentioned intractability assumption, one can *not* generate the sequence backwards. From the additional knowledge of P and Q, one *can* generate the sequence backwards; one can even "jump" about from any point in the sequence to any other. Because of these properties, the $x^2 \bmod N$ *generator* promises many interesting applications, e.g., to public-key cryptography. To use these generators in practice, an analysis is needed of various properties of these sequences such as their periods. This analysis is begun here.

Key words. random, pseudo-random, Monte Carlo, computational complexity, secure transactions, public-key encryption, cryptography, one-time pad, Jacobi symbol, quadratic residuacity

What do we want from a pseudo-random sequence generator? Ideally, we would like a pseudo-random sequence generator to quickly produce, from short seeds, long sequences (of bits) that appear in every way to be generated by successive flips of a fair coin.

Certainly, the idea of a (fast) deterministic mechanism producing such non-deterministic behavior seems contradictory: by observing its outcome over time, we could in principle eventually detect the determinism and simulate such a generator.

The resolution [Knuth], usually, is to require of such generators only that the sequences they produce pass certain standard statistical tests (e.g., in the long run, the frequency of 0's and 1's occurring in such a sequence should be nearly the same, and the 0's and 1's should be "well-mixed").

However, the usual statistical tests do not capture enough. An important property of sequences of coin tosses is their unpredictability. Pseudo-random sequences should be unpredictable to computers with feasible resources. We say that a pseudo-random sequence generator is *polynomial-time unpredictable* (unpredictable to the right, unpredictable to the left) [Shamir], [Blum–Micali] if and only if for every finite initial segment of sequence that has been produced by such a generator, but with any element (the rightmost element, the leftmost element) deleted from that segment, a probabilistic

* Received by the editors September 7, 1982, and in final revised form August 15, 1983. A preliminary version of this paper was presented at Crypto 82.

† Department of Mathematics and Computer Science, Mills College, Oakland, California 94613, and Department of Mathematics, University of California at Berkeley, Berkeley, California 94720. This work was supported in part by the Letts-Villard Chair, Mills College.

‡ Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, California 94720. This work was supported in part by the National Science Foundation under grant MCS 82-04506.

§ IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598, and City University of New York, New York, New York 10036. This work was supported in part by the National Science Foundation under grant MCS 82-01267.

Turing machine can, roughly speaking, do no better in guessing in polynomial time (polynomial in the length of the "seed," cf. § 2) what the missing element is than by flipping a fair coin.

1. Two pseudo-random sequence generators. In this paper, two pseudo-random sequence generators are defined and their properties discussed. These are called:

(1) the $1/P$ *generator,*

(2) the $x^2 \bmod N$ *generator.*

The two generators are closely related. For example: *From short seeds, each quickly generates long well-distributed sequences. Both generators contain hard problems at their core* (the discrete logarithm problem and the quadratic residuacity problem, respectively). *But only the second is "unpredictable"—assuming a certain intractibility hypothesis.*

More specifically, THEOREM 2, Problem 4 (§ 6). *Any sequence produced by the* $1/P$ *generator is completely predictable; that is, given a small segment of the sequence, one can quickly infer the "seed" and efficiently extend the given segment backwards and forwards.*

On the other hand, THEOREM 4 (§ 7). *Modulo the quadratic residuacity assumption, the* $x^2 \bmod N$ *generator is polynomial-time unpredictable to the left. We say, for reasons pointed out in the applications* (§ 10), *that the sequences it generates are cryptographically secure.*

The $1/P$ generator has been well studied in the history of number theory [Dickson] and as a pseudo-random number generator [Knuth]. Our results concerning its strong inference properties, we believe, are new and surprising.

The $x^2 \bmod N$ generator is an outgrowth of the coin-flipping protocol of [Blum]. Its strong security properties derive from complexity based number theoretic assumptions and arguments [Blum], [Goldwasser–Micali], [Yao]. Our investigation reveals additional useful properties of this generator: e.g., from knowledge of the (secret) factorization of N, one can generate the sequence backwards; from additional information about N, one can even random access the sequence. Our number-theoretic analyses also provide tools for determining the lengths of periods of the generated sequences.

Both generators have applications. The $1/P$ generator has applications to the generation of generalized de Bruijn (i.e., maximum-length shift-register) sequences. The $x^2 \bmod N$ generator has applications to public-key cryptography.

The two generators are presented together so that each one's properties help to illuminate the other's.

2. Notation and definitions. In this paper, the underlying models of computation are Turing machines [Hopcroft and Ullman]. *Probabilistic procedures* are effective procedures (Turing machines) that can toss a fair coin (at a cost of 1 step per toss) to produce truly random bits during their computation. (*Probabilistic*) *polynomial-time procedures* halt in (worst-case) time $poly(n)$, where *poly* denotes a polynomial, and n is the input *length.*

The *base,* b, will always be an integer >1. For any positive integer, N, let $|N|_b = \lfloor 1+\log_b N \rfloor$ be the *length of* N when N is expanded base b, and let $|N| = |N|_2$. We also let $n = |N|$ so $N = O(2^n)$.

For $\Sigma = \{0, 1, \cdots, b-1\}$, let Σ^* be the set of *finite sequences* of elements of Σ, and let Σ^∞ be the set of (one-sided) *infinite sequences* of elements of Σ.

For $x \in \Sigma^*$, let $|x|$ be the *length of* x, and for integers $k \geqq 0$, let $\Sigma^k = \{x \in \Sigma^* \mid |x| = k\}$. For $x \in \Sigma^\infty$, and for integers $k \geqq 0$, let x^k be the *initial segment of* x of length k, and x_k be the kth coordinate of x where x_0 is the initial coordinate of x.

DEFINITION. Let $\mathbf{N}$ be a set of positive integers, the *parameter values*, and for each $N \in \mathbf{N}$, let $X_N \subset \{0,1\}^n$ be a set of *seeds* (recall $n = |N|$). The set $X = \{(N, x) | N \in \mathbf{N}, x \in X_N\}$ is called a *seed domain.*

We can, and sometimes do, think of X_N as a subset of X by identifying seed $x \in X_N$ with "seed" $(N, x) \in X$. With this identification, X can be thought of as the disjoint union $\cup_{N \in \mathbf{N}} X_N$. The point of view should be clear from context.

DEFINITION. Let $X^n = \{(N, x) | N \in \mathbf{N}, |N| = n, \text{ and } x \in X_N\}$ be the set of *seeds of length n.* Suppose for all sufficiently large integers n, μ_n is a probability distribution on X^n. Then $U = \{\mu_n\}$ is an *accessible probability distribution on* X if there is a polynomial *poly* and a probabilistic *poly*(n)-time procedure that for each sufficiently large input, n, outputs an element of X^n according to μ_n *with negligible error*, i.e., it outputs an element of a set containing X^n according to μ'_n (μ'_n = a probability distribution on the set containing X^n) where, for all t, for all sufficiently large n, $\sum_{(N,x) \in X^n} |\mu_n(N, x) - \mu'_n(N, x)| < 1/n^t$.

A pair (X, U), where X is a seed domain and U is an accessible probability distribution on X, is called a *seed space.* We simply let X denote the seed space when the underlying distribution is clear.

Now, let $\Sigma = \{0, 1, \cdots, b-1\}$.

DEFINITION. A (*base b*) *pseudo-random sequence generator G on seed space X* is an effective map $G: X \to \Sigma^\infty$ such that for each integer $s \geqq 0$, there is an integer $t \geqq 0$ such that for all $(N, x) \in X$ with $\mu_n(N, x) \neq 0$, $[G(N, x)]^{n^s}$, the initial segment of $G(N, x)$ of length n^s, is output in time $O(n^t)$. (Thus, from short "seeds" (i.e., of "length" n), that are produced using at most poly(n) truly random bits, G generates long sequences (i.e., of length n^s), in polynomial time.) $G(N, x)$ is called the *pseudo-random sequence generated by G* with *input* or *seed* (N, x).

Remark. If Σ represents a set of "observable states" for elements of seed space X, then the sequence $G(N, x)$ might represent the observed states through which seed x passes (at times $0, 1, 2, \cdots$) resulting from some underlying transformation of X into itself. This point of view motivates the following more structured (and more restrictive) formulation of a pseudo-random sequence generator.

DEFINITION. A *transformation* T on seed space X is a poly-time effective map $T: X \to X$ such that for all sufficiently large n, $T(X^n) \subset X^n$ and T preserves μ_n (i.e., $\mu_n(A) = \mu_n(T^{-1}(A))$ for each $A \subset X^n$). For each seed $x \in X_N$, the sequence x, Tx, $T^2x, \cdots$ is called the *orbit* of x under T. We sometimes write $x_k = T^k x$, so $x_0 = x$ and $x_{k+1} = T(x_k)$.

DEFINITION. A *partition* B of seed space X into states Σ is a poly-time effective map $B: X \to \Sigma$.

The system $\langle X, T, B \rangle$, with X a seedspace, T a transformation on X, and B a partition naturally defines a base b pseudo-random sequence generator G on X where the kth coordinate, $[G(N, x)]_k = B(T^k x)$. Thus, if $x_0, x_1, x_2, \cdots$ is the orbit of x under T, then $G(N, x) = b_0 b_1 \cdots$ where $b_k = B(x_k)$ is the state of x at time k.

Remark. If T is poly-time invertible on X, i.e., if T^{-1} is defined and poly-time computable, we can, and sometimes do, think of G mapping X into the set of 2-sided infinite sequences on Σ.

In the next two sections we give examples of specific pseudo-random sequence generators. We use x^2 *mod N generator* to denote a particular type of pseudo-random sequence generator, whereas x^2 mod N denotes the remainder upon dividing a specific integer x^2 by N. A similar distinction is made between the $1/P$ *generator* and the string of digits $1/P$.

Throughout this paper $x \bmod N$ denotes the least nonnegative integer remainder upon dividing x by N (rather than denoting the residue class mod N).

Recall that $Z_N^* = \{\text{integers } x | 0 < x < N \text{ and } \gcd(x, N) = 1\}$ is a multiplicative group of order $\varphi(N)$. If P is prime, then $Z_P^* = \{1, 2, \cdots, P-1\}$ is cyclic. For each N, we consider $Z_N^* \subset \{0,1\}^n$ via the natural identification.

3. The $1/P$ generator. Fix an integer $b > 1$ and let $\Sigma = \{0, 1, \cdots, b-1\}$.

DEFINITION ($1/P$ *generator* (*base* b)). To define the *seed space*, let $\mathbf{N} = \{\text{integers } P > 1 \text{ relatively prime to } b\}$ be the *parameter values*, and let the *seed domain* X be the disjoint union $\cup_{P \in \mathbf{N}} Z_P^*$. We can, and sometimes do, identify X with the (dense) subset $\{r/P | P \in \mathbf{N}, r \in Z_P^*\}$ of the unit interval $[0, 1)$. Let μ_n be the distribution on X^n given by $\mu_n(P, r) = u_n(P) \cdot v_P(r)$, where u_n is the uniform probability distribution on $\{P \in \mathbf{N} \,||P|_b = n\}$ and v_P is the uniform distribution on Z_P^*. Then $U = \{\mu_n\}$ is an accessible probability distribution on X.

Let $G: X \to \Sigma^\infty$ be defined by letting $G(r/P) = q_1 q_2 q_3 \cdots$ be the sequence of b-ary quotient digits that immediately follow the decimal point when $r/P \in X$ is expanded base b. (We note that the successive digits of $G(r/P)$ can be computed in $O(|b| \cdot |P|)$-time, and that the sequence $G(r/P)$ is periodic with period dividing $\varphi(P)$.) We call this pseudo-random sequence generator the $1/P$ *generator* (*base* b).

From the state space point of view, the $1/P$ generator (*base* b) is the pseudo-random sequence generator defined by the triple $\langle X, T, B \rangle$ where X is the *seed space* defined above, the *transformation* $T: X \to X$ is defined for x in $[0, 1)$ by $Tx = bx \bmod 1$ (equivalently $T(r) = br \bmod P$ for $r \in Z_P^*$, which is a permutation on Z_P^*), and the *partition* $B: X \to \Sigma = \{0, 1, 2, \cdots, b-1\}$ is defined for x in $[0, 1)$ by $B(x) = \lfloor bx \rfloor$ (equivalently, $B(r) = \lfloor br/P \rfloor$ for $r \in Z_P^*$).

Remark. The $1/P$ *generator* (*base* 2) might be considered to be a discrete realization of the classical arithmetical model of a coin toss defined by the map $2x \bmod 1$ and partition $[0, \frac{1}{2}) \cup [\frac{1}{2}, 1)$ of the unit interval [Billingsley, Kac]. In § 6 we see that while a number of the "ergodic" like properties of the classical model are reflected in this discrete realization, the sequences produced are predictable.

Example. Let the base $b = 10$, and let $P = 7$ and $r = 1$. The pseudo-random sequence generated by the $1/P$ generator (*base* 10) with input $1/7$ is $142857142 \cdots$. Note that 10 is a primitive root mod 7 (i.e., a generator of the cyclic group Z_7^*) and that the period of this sequence is $7 - 1 = 6$ (see Theorem 1). From the state space point of view, the orbit of $1/7$ under T is: $1/7, 3/7, 2/7, 6/7, 4/7, 5/7, 1/7, \cdots$, and so, $b_0 = 1$ (since $1/7 \in [1/10, 2/10)$), $b_1 = 4$ (since $3/7 \in [4/10, 5/10)$), $b_2 = 2$, $b_3 = 8$, $b_4 = 5, \cdots$.

4. The generator $x^2 \bmod N$.

DEFINITION [$x^2 \bmod N$ generator]. Let $\mathbf{N} = \{\text{integers } N | N = P \cdot Q, \text{ such that } P, Q \text{ are equal length } (|P| = |Q|) \text{ distinct primes} \equiv 3 \bmod 4\}$ be the set of *parameter values*. For $N \in \mathbf{N}$, let $X_N = \{x^2 \bmod N | x \in Z_N^*\}$ be the set of quadratic residues mod N. Let $X = \text{disjoint } \cup_{N \in \mathbf{N}} X_N$ be the *seed domain*.

For $(N, x) \in X^n$, let $\mu_n(N, x) = u_n(N) \cdot v_N(x)$, where u_n is the uniform probability distribution on $\{N \in \mathbf{N} \,||N| = n\}$ and v_N is the uniform distribution on X_n. Then $\{\mu_n\}$ is an *accessible probability distribution* on X since

1. asymptotically, $1/(k \ln 2)$ of all k-bit numbers are prime and half the primes of any given lengths are $\equiv 3 \bmod 4$ (by de la Vallee Poussin's extension of the prime number theorem [Shanks]);

2. primality is decidable in polynomial time by (Monte-Carlo) probabilistic procedures [Strassen–Solovay], [Rabin '80] or, assuming the extended Riemann hypothesis, by a deterministic polynomial time procedure [Miller], and

3. gcds are computable in polynomial time, and $|Z_N^*|/|Z_N| \to 1$ as $n \to \infty$.

Let the *transformation* $T: X \to X$ be defined by $T(x) = x^2 \bmod N$ for $x \in X_N$. T is a permutation on X_N (see Lemma 1) and is computable in poly-time. Let the *partition* $B: X \to \{0, 1\}$ be defined by $B(x) =$ parity of x. B is computable in poly-time. Then $\langle X, T, B\rangle$ defines a pseudo-random sequence generator (base 2) called the $x^2 \bmod N$ generator. Thus, with *inputs* (N, x_0) the $x^2 \bmod N$ *generator outputs the pseudo-random sequence* of bits $b_0 b_1 \cdots$ obtained by setting $x_{i+1} = x_i^2 \bmod N$ and extracting the bit $b_i =$ parity (x_i). Such sequences are periodic with period usually equal to $\lambda(\lambda(N))$ (see § 8 for the definition of λ and clarification of "usually"). We also note that the equality $x_i = x_0^{2^i} \bmod N = x_0^{2^i \bmod \lambda(N)} \bmod N$ enables us to efficiently compute the ith sequence element, given x_0, N and $\lambda(N)$, for $i > 0$. For $i < 0$, use $x_i = x_{i \bmod \lambda(\lambda(N))}$.

Example. Let $N = 7 \cdot 19 = 133$ and $x_0 = 4$. Then the sequence $x_0, x_1 = x_0^2 \bmod 133, \cdots$ has period 6, where $x_0, x_1, \cdots, x_5, \cdots = 4, 16, 123, 100, 25, 93, \cdots$. So $b_0 b_1 \ldots b_5 \ldots = 0\ 0\ 1\ 0\ 1\ 1 \cdots$. The latter string of b's is the pseudo-random sequence generated by the $x^2 \bmod N$ generator with input (133, 4). Here, $\lambda(N) = 18$ and $\lambda(\lambda(N)) = 6$.

5. The assumptions. Our main results about unpredictability and cryptographic security follow from assumptions concerning the intractability of certain number-theoretic problems by probabilistic polynomial-time procedures. Stronger results would follow from stronger assumptions concerning the circuit size complexity of the number theoretic problems below. Such results would be desirable, for example, if we wished to assure that sequences produced by our generator appear random to hard-wired circuits.

1. *The discrete logarithm (index finding) problem.* Let P be a prime. Let b be a primitive root mod P (i.e., a generator for Z_P^*). The function $f_{b,P}: Z_P^* \to Z_P^*$ defined by $f_{b,P}(x) = b^x \bmod P$ is a permutation of Z_P^* that is computable in polynomial time. The *discrete logarithm (index finding) problem* with parameters b and P consists in finding for each y in Z_p^* the index x in Z_p^* such that $b^x \bmod P = y$. A (probabilistic) procedure $\mathbf{P}[b, P, y]$ solves the discrete logarithm problem if for all primes P, for all generators b for Z_P^*, and for all y in Z_P^*, $\mathbf{P}[b, P, y] = x$ in Z_P^* such that $b^x \bmod P = y$.

The discrete logarithm assumption. (This asserts that any procedure for solving the discrete logarithm problem will be inefficient for a fraction of the inputs.) Let $\mathbf{P}[b, P, y]$ be a (probabilistic) procedure for solving the discrete logarithm problem. Let $0 < \delta < 1$ be a fixed constant, and let t be a fixed positive integer. Let poly be a fixed polynomial. Then for all sufficiently large n, for all but δ-fraction of n-bit primes P, for all primitive roots b mod P, Probability {(expected) time to compute $\mathbf{P}[b, P, y] \geqq$ poly $(n) | y$ is selected uniformly from $Z_P^*\} > 1/n^t$.

2. *The quadratic residuacity problem* [Gauss]. Let N be a product of two distinct odd primes. Exactly half the elements of Z_N^* have Jacobi symbol $+1$, the other half have Jacobi symbol -1. Denote the former by $Z_N^*(+1)$ and the latter by $Z_N^*(-1)$. None of the elements of $Z_N^*(-1)$ and exactly half the elements of $Z_N^*(+1)$ are quadratic residues. The *quadratic residuacity problem* with parameters N and x consists in deciding, for x in $Z_N^*(+1)$, whether or not x is a quadratic residue.

The quadratic residuacity assumption (QRA). (This asserts that any efficient procedure for guessing quadratic residuacity will be incorrect for a fraction of the inputs.) Let poly $(\cdot)$ be a polynomial. Let $\mathbf{P}[N, x]$ be any (probabilistic) poly-time

procedure which, on inputs N, x, each of length n, outputs 0 or 1. Let $0<\delta<1$ be a fixed constant, and let t be a positive integer. Then for n sufficiently large and for all but δ fraction of numbers N of length n, N a product of two distinct equal length odd primes, the probability that $\mathbf{P}[N, x]$ is incorrect in guessing quadratic residuacity (i.e., $\mathbf{P}[N, x]=0$ if $x \in Z_N^*(+1)$ is a quadratic residue mod N; 1 if not), given that x is chosen uniformly from $Z_N^*(+1)$ and given the sequence of coin flips (in the case the procedure is probabilistic), exceeds $1/n^t$ in the sense that

$$\left(\frac{\sum_{x \in Z_N^*(+1)} \text{Prob}\,(\mathbf{P}[N, x] \text{ is incorrect})}{\varphi(N)/2}\right) > 1/n^t.$$

By Lemma 3, the "$1/n^t$" is replaceable by "$(1/2)-(1/n^t)$."

6. The $1/P$ generator is predictable. Let P and b be relatively prime integers >1 and r_0 an integer in the range $0<r_0<P$. Denote the expansion of r_0/P to base b by

$$r_0/P = .q_1 q_2 q_3 \cdots \tag{1}$$

where $0 \leqq q_i < b$. Since b is prime to P, the expansion is periodic. Then, for $m \geqq 0$,

$$(b^m \cdot r_0)/P = q_1 \cdots q_m . q_{m+1} q_{m+2} \cdots = (q_1 \cdots q_m) + r_m/P \tag{2}$$

where

$$0 < r_m = b^m r_0 \bmod P < P \tag{3}$$

and

$$0 < r_m/P = .q_{m+1} q_{m+2} \cdots = (b^m \cdot r_0/P) \bmod 1 < 1. \tag{4}$$

Here, $q_1, q_2, \cdots$ are (quotient) *digits* base b and $q_1 q_2 \cdots$ denotes their concatenation, whereas r_m, the mth remainder (of r_0/P base b), is an *integer* whose length (base b) is less than or equal to the length of P: $|r_m|_b \leqq |P|$, where in this section $|P|$ denotes $|P|_b$. Recall for $r_0 \in Z_P^*$, (1) defines the pseudo-random sequence generated by the $1/P$ generator with input r_0/P.

There are several reasons one might consider the $1/P$ generator a good pseudo-random sequence generator: if the parameter P is a prime, and b is a primitive root mod P, the sequences produced have long periods and nice distribution properties (Theorem 1 below)[1]. In addition, these sequences possess certain hard-to-infer properties. For example, given a remainder r generated during the expansion of $1/P$ base b, it is hard, in general, to find any index m such that $r_m = r$. This is because $r_m = b^m \bmod P$, so m is the discrete logarithm of r mod P. It follows (Theorem 2, problem 1) that, given a string of quotient digits $q_{m+1} q_{m+2} \cdots q_{m+k}$ ($k \leqq \text{poly}\,(|P|)$), it is hard in general to find its location in the sequence.

---

[1] We remark that it would be natural to restrict the $1/P$ generator (base b) to the seed space $Y = \{(P, r) | P$ is an odd prime, b is a primitive root mod P, $r \in Z_P^*\}$ with the product distribution: for each $(P, r) \in Y^n$, let $\mu_n(P, r) = u_n(P) \cdot v_P(r)$, where u_n is the uniform distribution on the parameters of length n and v_P is uniform on Z_P^*. Then, on reasonable conjecture, $\{\mu_n\}_{n \in Z^+}$ is accessible on Y since: a) E. Artin's conjecture and the prime number theorem imply that if b is not a square, then the cardinality of $\{P | P$ is a prime of length n and b is a primitive root mod $P\}$ is more than $(1/3) \cdot (2^n/n)$, asymptotically as n goes to infinity [Shanks p. 81]. And, there are (Monte-Carlo) probabilistic polynomial-time procedures for b) testing primality [Strassen–Solovay]; c) testing if b is a primitive root mod P, given P and the factorization of $P-1$ [LeVeque, Thm. 4.8]; d) producing, for any k, k bit integers in factored form according to the uniform probability distribution [Bach]; and e) computing greatest common divisors.

On the other hand, Theorem 2 will give a sense, which is correct, that the $1/P$ generator yields a poor pseudo-random sequence: from knowledge of P and any $|P|$-long segment of the expansion of r_0/P base b, one can efficiently extend the segment backwards and forwards (problem 2). More surprisingly (problem 4), from knowledge of any $2|P|+1$ successive elements of the sequence, but *not* P, one can efficiently reconstruct P, and hence efficiently continue the sequence in either direction.

It follows that there is a simple efficient statistical test for deciding whether a $3n$-long string of digits has either been generated by the expansion of $1/P$ base b, for some prime P of length n, or has been generated at random (uniform probability distribution), given that it was produced in one of those two ways. Use $2n+1$ of the given $3n$ digits to recover the suspected P; use this P to generate $3n$ digits; then compare the generated digits with the $3n$ given digits: if they agree, the string has probably (with probability $\geqq 1-1/2^{n-1}$) been generated using the $1/P$ generator.

To lead up to Theorem 1, we consider the following types of sequences (closely related to maximum-length shift register sequences [Golomb]).

DEFINITION. Let P, b denote arbitrary positive integers. A (*generalized*) *de Bruijn sequence of period* $P-1$, *base* b, is a sequence $q_1q_2\cdots$ of b-ary digits (i.e., $0\leqq q_i<b$ for all i) of period $P-1$ such that (1) every b-ary string of length $|P|-1$ occurs at least once in the sequence, and (2) every b-ary string of length $|P|$ occurs at most once in any given period of the sequence.

THEOREM 1. *Let* $P=$ *prime. Let* $b\in\{1,2,\cdots,P-1\}$ *be a primitive root* mod P. *Let* $r_0\in\{1,2,\cdots,P-1\}$. *Then the pseudo-random sequence generated by the* $1/P$ *generator* (*base* b) *with input* r_0/P *is a* (*generalized*) *de Bruijn sequence of period* $P-1$, *base* b.

Proof. Since $r_m=b^m\cdot r_0 \bmod P$ and b is a primitive root mod P, the sequence of remainders r_m (generated during the expansion of $1/P$ base b) is periodic with period $P-1$, the remainders in any period are distinct, and $\{r_m|1\leqq m\leqq P-1\}=\{1,2,\cdots,P-1\}$.

Similarly, the sequence of quotients r_m/P is periodic with period $P-1$, the quotients in any period are distinct, and

$$\{r_m/P|1\leqq m\leqq P-1\}=\{1/P,2/P,\cdots,(P-1)/P\}. \tag{5}$$

Therefore, the sequence of quotient digits q_m is periodic with period at most $P-1$. If the period were less than $P-1$, then there would be integers $0\leqq m_1<m_2<P-1$ such that $.q_{m_1+1}q_{m_1+2}\cdots=.q_{m_2+1}q_{m_2+2}\cdots$. Since $r_m/P=.q_{m+1}q_{m+2}\cdots$, we would have $r_{m_1}/P=r_{m_2}/P$, a contradiction. Therefore the period is $P-1$. [Gauss]

Now, a string $a_1\cdots a_s$ of s b-ary digits appears somewhere in the expansion of r_0/P if and only if it appears as an initial string in the expansion of r_m/P for some $1\leqq m\leqq P-1$ if and only if (by (5)) it appears as an initial string in the expansion of k/P for some $1\leqq k\leqq P-1$. But also, the set of b-ary strings of length s correspond exactly to the subintervals of the unit interval $[0,1)$ of the form $[l/b^s,(l+1)/b^s)$ where l is an integer, $0\leqq l<b^s$. Since $1/P<1/b^{|P|-1}$, there is for each l, at least one k, $1\leqq k\leqq P-1$ such that $k/P\in[l/b^{|P|-1},(l+1)/b^{|P|-1})$ and so we have property 1. Since $1/b^{|P|}<1/P$, there is for each l at most one k, $1\leqq k\leqq P-1$ such that $k/P\in[l/b^{|P|},(l+1)/b^{|P|})$, and so we get property 2. QED

So, if P is prime and b is a primitive root mod P, it follows from Theorem 1 concerning de Bruijn property 1 (and Artin's conjecture—see footnote 2 concerning that conjecture) that neither $|P|-1$ successive digits of quotient, $q_{m+1}\cdots q_{m+|P|-1}$, nor (the approximately $|P|-1$ successive digits of) a remainder, r_m, are enough to construct P, or to extend the sequence, on purely information-theoretic grounds. In contrast, it

will follow from Theorem 2 below that (various combinations of) approximately $2|P|$ digits of information are sufficient to efficiently extend the sequence in either direction.

THEOREM 2. *Let P and b be relatively prime integers >1 (P not necessarily prime), and let r_0 be an integer in the range $0<r_0<P$. The following problems are solvable in polynomial($|P|$)-time.*

Problem 1. *Choose a polynomial,* poly $(\cdot)$, *and hold it fixed.*
INPUT: P, b, *remainder* r_m, *positive integer* $k \leqq \text{poly}(|P|)$.
OUTPUT: r_{m-1}, r_{m+k}; $q_m q_{m+1} \cdots q_{m+k}$.

Problem 2 [Gauss]. *This is a computational version of Theorem* 1 *concerning de Bruijn property* 2. (*A similar algorithm gives the computational version of property* 1.)
INPUT: P, b, $|P|$ *successive digits of quotient* $q_{m+1}q_{m+2}\cdots q_{m+|P|}$.
OUTPUT: r_m (*and hence, by problem* 1, $r_{m+|P|}$ and q_m, $q_{m+|P|+1}$).

Problem 3. *We assume that P is relatively prime to each of* $1, 2, \cdots, b$ (*to ensure that the output is the unique P that generated r_m and r_{m+1}*).
INPUT: b, r_m, r_{m+1} *such that* $r_m \cdot b \neq r_{m+1}$ (i.e. $r_m \geqq P/b$).
OUTPUT: P (*and therefore also, by problem* 1, $q_m q_{m+1}\cdots q_{m+|P|}$).

Problem 4. *We assume that r_0 is relatively prime to P* (e.g., $r_0 = 1$).
INPUT: b; k *quotient digits,* $q_{m+1}q_{m+2}\cdots q_{m+k}$, *where* $k = \lceil \log_b (2P^2) \rceil$ *and m is arbitrary.* (*Note that* $k \leqq 2|P|+1$).
OUTPUT: P; r_m (*and hence by problem* 1, q_m *and* q_{m+k+1}).

Proof. To solve problem 1: $r_{m+k} = b^k r_m \bmod P$ and $r_{m-1} = b^{-1} r_m \bmod P$ where b^{-1} is the inverse of b mod P. We note that

$$(b^k r_m)/P = q_{m+1}\cdots q_{m+k} + r_{m+k}/P. \tag{6}$$

So, $q_m \cdots q_{m+k} = \lfloor (b^{k+1} r_{m-1})/P \rfloor$. (By convention, we do not drop initial digits in a concatenation of quotient digits, e.g., in (6).)

To solve problem 2: By (6), $r_m = (q_{m+1}\cdots q_{m+|P|})\cdot P/b^{|P|} + (r_{m+|P|})/b^{|P|}$. Since $r_{m+|P|} < P < b^{|P|}$, $r_m = \lceil (q_{m+1}\cdots q_{m+|P|})\cdot P/b^{|P|} \rceil$.

In problems 3 and 4, the number P is not available and must be constructed.

To solve problem 3: By (6) with $k=1$, $b\cdot r_m - r_{m+1} = q_{m+1}\cdot P$ where $0 \leqq q_{m+1} < b$. Actually, $0 < q_{m+1}$, since, by assumption, $b\cdot r_m \neq r_{m+1}$. Therefore, P equals some integer in the sequence of real numbers $(b\cdot r_m - r_{m+1})/1$, $(b\cdot r_m - r_{m+1})/2, \cdots, (b\cdot r_m - r_{m+1})/b - 1$. Select any integer P in the sequence such that P is relatively prime to $1, 2, \cdots, b$. Such an integer P is unique; for suppose to the contrary that P, Q are two such integers relatively prime to each of $1, 2, \cdots, b$. Then $P\cdot(i) = Q\cdot(j)$ for some $0 < i, j < b$. Without loss of generality, suppose $P < Q$. Q is relatively prime to each of $1, 2, \cdots, b$, so $\gcd(Q, i) = 1$, so $Q|P$, so $Q \leqq P$, which is a contradiction.

The solution to problem 4, which is very pretty, is by continued fractions: By (6), $r_m/P = q_{m+1}\cdots q_{m+k}/b^k + \varepsilon$ where $0 \leqq \varepsilon < 1/b^k$. By [LeVeque, p. 237, Thm. 9.10], the continued fraction expansion of $q_{m+1}\cdots q_{m+k}/b^k$ has convergent r_m/P if $1/b^k \leqq 1/2P^2$, i.e., $2P^2 \leqq b^k$, i.e., $\log_b (2P^2) \leqq k$, as postulated. So $r_m/P = A_i/B_i$ for one of the convergents $A_1/B_1, A_2/B_2, \cdots$ of the fraction $q_{m+1}\cdots q_{m+k}/b^k$. Since both b and r_0 are relatively prime to P, it follows (from (3)) that $\gcd(r_m, P) = 1$, so $r_m = A_i$ and $P = B_i$.

It remains to show that r_m and P can be obtained by generating the above convergents until for some i the first k digits of A_i/B_i are $q_{m+1}\cdots q_{m+k}$, at which point $r_m = A_i$ and $P = B_i$. To see why, recall that the continued fraction $q_{m+1}\cdots q_{m+k}/b^k = 1/a_1 + 1/a_2 + 1/a_3 + \cdots 1/a_i + \cdots$ has convergents $A_1/B_1 = 1/a_1$, $A_2/B_2 = a_2/(a_1a_2+1), \cdots, A_i/B_i = (a_iA_{i-1} + A_{i-2})/(a_iB_{i-1} + B_{i-2}), \cdots$. Here, the B_i

are strictly increasing with i. Since for some i, $A_i/B_i = r_m/P$, this procedure for obtaining r_m and P will never go beyond $A_i/B_i = r_m/P$. To see that the procedure generates convergents to the point where $A_i/B_i = r_m/P$, note that when $A_j/B_j = q_{m+1} \cdots q_{m+k} \cdots$, the error is sufficiently small to ensure that $A_j/B_j = r_m/P$.

Since A_i and B_i grow exponentially, $P = B_i$ and $r_m = A_i$ can be computed in polynomial($|B_i|$), in particular in O(number of steps to compute the ith Fibonacci number), and therefore in polynomial($|P|$) steps. This solves problem 4. QED

Example. Let $b = 10$ and $P = 503$. Then P is a prime and b is a primitive root mod P, so the $1/P$ *generator* with input 1/503 quickly generates a sequence of base 10 digits with period 502. This sequence is

00198 80715 70576 54075 54671 96819 08548 70775 34791 25248 50894 63220 67594 **43339 9**6023
85685 88469 18489 06560 63618 29025 84493 04174 95029 82107 35586 48111 33200 79522 86282
30616 30218 68787 27634 19483 10139 16500 99403 57852 88270 37773 35984 09542 74353 87673
95626 24254 47316 10337 97216 69980 11928 42942 34592 44532 80318 09145 12922 46520 87475
14910 53677 93240 55666 00397 61431 41153 08151 09343 93638 17097 41550 69582 50497 01789
26441 35188 86679 92047 71371 76938 36978 13121 27236 58051 68986 08349 90059 64214 71172
96222 66401 59045 72564 61232 60437 37574 55268 38966 20278 33001 98807 $\cdots$

Since $|503| = 3$, every string of two decimal digits occurs at least once in the above sequence, and every string of three decimal digits occurs at most once in any period of the sequence.

Since $k = \lceil \log_{10}(2 \cdot 503^2) \rceil = 6$, we can, from any segment of length 6 of the above sequence, efficiently recover P, and then quickly extend the segment in either direction. For example, consider the segment 433399 (shown in bold type above). The continued fraction expansion of .433399 is $433{,}399/1{,}000{,}000 = 1/2+1/3+1/3+1/1+1/16+1/6+1/1+1/1+1/358+\cdots$, and its first five convergents are: $\frac{1}{2} = .5$; $\frac{3}{7} = .48\cdots$; $\frac{10}{23} = .434\cdots$; $\frac{13}{30} = .43333\cdots$; $\frac{218}{503} = .4333996\cdots$. At last, the first 6 digits agree with the segment 433399. So we get $P = 503$ and $r_m = 218$ (and so $r_{m-1} = 10^{-1} \cdot r_m \bmod 503 = 151 \cdot 218 \bmod 503 = 223$). In this way, we can extend the given segment, 433399, forwards and backwards.

7. The x^2 mod N generator is unpredictable. In this section we elaborate on properties of the $x^2 \bmod N$ pseudo-random sequence generator, and prove (modulo the QRA) that it is polynomial-time unpredictable (Theorem 4, this section).

First we recall some number-theoretic facts. Suppose $N = P \cdot Q$ where P and Q are distinct odd primes. Let $Z_N^* = \{\text{integers } x | 0 < x < N \text{ and } \gcd(x, N) = 1\}$. Then QR_N, the set of quadratic residues mod N, form a multiplicative subgroup of Z_N^* of order $\varphi(N)/4$ (where $\varphi(N)$ is the cardinality of Z_N^*). Each quadratic residue $x^2 \bmod N$ has four distinct square roots, $\pm x \bmod N$, $\pm y \bmod N$. If we also assume, as we shall for the rest of this paper, that $P \equiv Q \equiv 3 \bmod 4$, then each quadratic residue mod N has *exactly* one square root which is also a quadratic residue (see Lemma 1, this section). In other words, squaring mod N is a $1-1$ map of QR_N onto QR_N. (Comment: half the primes of length n are congruent to 3 mod 4 asymptotically as $n \to \infty$ [LeVeque], so there are plenty such N.)

We now investigate what properties can be inferred about sequences produced by the $x^2 \bmod N$ generator, given varying amounts of information. In the following, N is of the *prescribed form*, that is to say, $N = P \cdot Q$ where P, Q are distinct primes both congruent to 3 mod 4. Also, x_i is a quadratic residue mod N, $x_{i+1} = x_i^2 \bmod N$ and $b_i = \text{parity}(x_i)$:

1. Clearly, knowledge of N is sufficient to efficiently generate sequences $x_0, x_1, x_2, \cdots$ (and hence sequences $b_0 b_1 b_2 \cdots$) in the forward direction, starting from any given seed x_0. The number of steps per output is $O(|N|^{1+\varepsilon})$ using fast multiplication.

2. Given N, the factors of N are necessary and sufficient to efficiently generate the $x^2 \bmod N$ sequences in the reverse direction, $x_0, x_{-1}, x_{-2}, \cdots$, starting from any given seed x_0. (See proof below).

3. What is more, the factors of N are necessary—assuming they are necessary for deciding quadratic residuacity of an x in $Z_N^*(+1)$—to have even an ε-advantage in guessing in polynomial time the parity of x_{-1}, given N and given x_0 chosen "at random" from QR_N. (Note that to choose a quadratic residue at random with the uniform probability distribution from QR_N, it is sufficient to choose x at random (with the uniform probability distribution) from Z_N^* and square it mod N).[2]

To see Claim 2 above, we first prove the following:

LEMMA 1. *If $N = P \cdot Q$ where P and Q are distinct primes such that $P \equiv Q \equiv 3 \bmod 4$, then each quadratic residue* mod N *has exactly one square root that is a quadratic residue.*

Proof. Whenever N is a product of two distinct odd primes, every quadratic residue mod N has four square roots, $\pm x$ and $\pm y$. Since $N \equiv 1 \bmod 4$, their Jacobi symbols satisfy $(+x/N) = (-x/N)$ and $(+y/N) = (-y/N)$. Since $P \equiv 3 \bmod 4$, $(+x/N) \neq (+y/N)$ (this can easily be proved from the fact that $\gcd(x+y, N) = P$ and $\gcd(x-y, N) = Q$, whence $x+y = kP$ and $x-y = lQ$, whence $(x/P) = (-y/P)$ and $(x/Q) = (y/Q)$). Thus $(+x/N) = (-x/N) \neq (+y/N) = (-y/N)$. Eliminating the two roots, say $\pm y$, with Jacobi symbol -1 with respect to N, we are left with the two roots $\pm x$ having Jacobi symbol $+1$ with respect to N. Exactly one of these roots has Jacobi symbol $+1$ with respect to both P and Q, because $P \equiv 3 \bmod 4$, and this one and this one only is a quadratic residue mod N. QED

The necessity (of knowing the factors of N) now follows: Suppose we can efficiently generate such sequences in the reverse direction. To factor N, select an x in Z_N^* whose Jacobi symbol is $(x/N) = -1$. Let $x_0 = x^2 \bmod N$ and compute x_{-1}. Then efficiently compute $\gcd(x + x_{-1}, N) = P$ or Q. We can sharpen this argument to show [Rabin '79] that the ability to compute x_{-1} for even a fraction of seeds x_0 will enable us to factor N efficiently with high probability.

On the other hand, if we know the factors of N we can use the algorithm described in Theorem 3 (below) to efficiently generate sequences backwards:

THEOREM 3. *There is an efficient deterministic algorithm A which when given N (of the prescribed form), the prime factors of N and any quadratic residue x_0 in Z_N^*, efficiently computes the unique quadratic residue x_{-1}* mod N *such that $(x_{-1})^2 \bmod N = x_0$. Thus,*

$$A(P, Q, x_0) = x_{-1}.$$

Proof. By Lemma 1, the map from the quadratic residues mod N into the quadratic residues mod N, $f: x \to x^2 \bmod N$, is $1-1$ onto. The algorithm A can now be described as follows:

INPUT: P, Q = two distinct primes congruent to 3 mod 4; x_0 = a quadratic residue mod N, where $N = P \cdot Q$.

OUTPUT: A quadratic residue x_{-1} mod N whose square mod N is x_0.

Compute $x_P = \sqrt{x_0} \bmod P$ such that $(x_P/P) = +1$, where $\sqrt{x_0} \bmod P$ denotes an integer in Z_P^* whose square mod P is x_0:

$\sqrt{x_0} \bmod P = \pm x_0^{(P+1)/4} \bmod P$ (for $P \equiv 3 \bmod 4$). Compute $x_Q = \sqrt{x_0} \bmod Q$. Use the Euclidean algorithm to construct integers u, v such that $P \cdot u + Q \cdot v = 1$, and from

[2] A more formal statement of claim 3: Modulo the QRA, given a polynomial poly, a constant $0 < \delta < 1$, and a positive integer t, if $\mathbf{P}[N, x_0]$ is a probabilistic poly-time procedure for guessing the parity of x_{-1} given x_0 in QR_N, then $(\sum_{x_0 \in QR_N} \text{Prob}[\mathbf{P}[N, x_0] = \text{Parity}(x_{-1})])/(\varphi(N)/4) < (1/2) + (1/n^t)$ for sufficiently large n, and all but δ fraction of prescribed integers N of length n.

that obtain the particular number, $x_N = \pm x_P \cdot Q \cdot v \pm x_Q \cdot P \cdot u = \sqrt{x_0} \bmod N$, that is a square root of $x_0 \bmod N$, and that is also a quadratic residue with respect to both P and Q and therefore with respect to N. QED

To see Claim 3 above, we start with the following definition.

DEFINITION. Given a polynomial poly and $0 < \varepsilon \leqq 1/2$, a $0-1$ valued probabilistic poly-time procedure $\mathbf{P}(\cdot, \cdot)$ has an *ε-advantage for N in guessing parity* (of x_{-1} given arbitrary x_0 in QR_N) if and only if $(\sum_{x_0 \in QR_N} \text{Prob}[\mathbf{P}[N, x_0] = \text{Parity}(x_{-1})])/(\varphi(N)/4) \geqq (1/2) + \varepsilon$. In a similar fashion, we can define a procedure having an *ε-advantage for N in guessing quadratic residuacity* (of arbitrary $x \in Z_N^*(+1)$) [Goldwasser–Micali]. The $1/2 + \varepsilon$ makes sense in the second definition since exactly half the elements in $Z_N^*(+1)$ are quadratic residues.

LEMMA 2. *An ε-advantage for guessing parity (of x_{-1} given quadratic residue x_0) can be converted, efficiently and uniformly, to an ε-advantage for guessing quadratic residuacity (of x in $Z_N^*(+1)$).*[3]

Proof. Let $x \in Z_N^*(+1)$ be an element whose quadratic residuacity mod N is to be determined. Set $x_0 = x^2 \bmod N$. Since $P \equiv Q \equiv 3 \bmod 4$, the square roots of $x^2 \bmod N$ that are in $Z_N^*(+1)$ are x and $N - x$ (see proof of Lemma 1), and since N is odd, each of these square roots has opposite parity. Only one of these square roots is a quadratic residue (i.e., equal to x_{-1}), and only one of these has parity equal to parity (x_{-1}). Therefore, x is a quadratic residue mod N if and only if $x = x_{-1}$ if and only if parity (x) = parity (x_{-1}). QED

LEMMA 3 (Goldwasser and Micali). *An ε-advantage for guessing quadratic residuacity can be amplified to a $1/2 - \varepsilon$ advantage, uniformly and efficiently.*[4]

Idea of proof. Suppose $\mathbf{P}[N, x]$ is a probabilistic poly-time procedure that has an ε-advantage for N in guessing quadratic residuacity. Then we can in polynomial time sample (uniformly) the elements x of QR_N and of $Z_N^*(+1) - QR_N$ (by selecting a number at random from Z_N^*, squaring it, then taking its negative mod N), and estimate constants A and B such that

$$\left(\frac{\sum_{x \in QR_N} \text{Prob}[\mathbf{P}[N, x] = 1]}{\varphi(N)/4}\right) \approx A \quad \text{and} \quad \left(\frac{\sum_{x \in Z_N^*(+1) - QR_N} \text{Prob}[\mathbf{P}[N, x] = 1]}{\varphi(N)/4}\right) \approx B,$$

and $A\text{-}B \geqq 2\varepsilon$ (approximately). Now let $x \in Z_N^*(+1)$ be an element whose quadratic residuacity mod N is to be determined. To this end, select r's independently and at random with uniform probability from Z_N^*. Compute $x \cdot r^2 \bmod N$. [Comment: For $x \in QR_N$, $x \cdot r^2 \bmod N$ is uniformly distributed over QR_N; for $x \notin QR_N$, $x \cdot r^2 \bmod N$ is uniformly distributed over $Z_N^*(+1) - QR_N$.] Test each of the resulting numbers, $x \cdot r^2 \bmod N$, for quadratic residuacity by checking if $\mathbf{P}[N, xr^2] = 1$ (using sequences of coin tosses as may be required by $\mathbf{P}$). Compare the resulting fraction of favorable outcomes to the fractions A and B in order to get an amplified advantage in guessing quadratic residuacity of x. QED

---

[3] A more formal statement of Lemma 2: Given poly, $0 < \varepsilon(n) \leqq 1/2$, $\mathbf{N}$ = a set of integers N of the prescribed type. If there is a probabilistic poly-time procedure that has an $\varepsilon(|N|)$-advantage for each $N \in \mathbf{N}$ in guessing parity (of x_{-1} given an arbitrary $x_0 \in QR_N$), then there is a polynomial poly′ and a probabilistic poly′ procedure that has an $\varepsilon(|N|)$-advantage for $N \in \mathbf{N}$ in guessing quadratic residuacity (of arbitrary x in $Z_N^*(+1)$).

[4] A more formal statement of Lemma 3: Given poly, t = a positive integer, and $\mathbf{N}$ = a set of integers N of the prescribed type. If there is a probabilistic poly-time procedure that has a $1/|N|^t$ advantage for $N \in \mathbf{N}$ in guessing quadratic residuacity (of x in $Z_N^*(+1)$, then for any positive integer t' there is a polynomial poly′ and a probabilistic poly′-time procedure that has a $1/2 - 1/|N|^{t'}$ advantage for $N \in \mathbf{N}$ in guessing quadratic residuacity (of $x \in Z_N^*(+1)$).

CLAIM 3 follows: Suppose to the contrary that **P** is a probabilistic poly procedure that has a $1/n^t$ advantage in determining parity (for infinitely many n, and for more than δ of prescribed numbers N of length n). Then convert **P** (Lemma 2) to a probabilistic poly′ procedure **P**′ for determining quadratic residuacity that has an amplified advantage (Lemma 3) of $1/2-1/n^{t'}$ (for these same integers N). This contradicts the quadratic residuacity assumption.

Leading up to Theorem 4 we make the following definition.

DEFINITION.

1. A *predictor* $\mathbf{P}(\cdot,\cdot)$ is a probabilistic poly-time procedure that on inputs N, $b_1 \cdots b_k$, with $b_i \in \{0, 1\}$ and $k \leqq \text{poly}(|N|)$, outputs a 0 or 1 (the output may depend on the sequence of coin-tosses that the probabilistic algorithm makes).

2. **P** has an *ε-advantage* for N in predicting to the left sequences produced by the $x^2 \bmod N$ generator if and only if for some $k \leqq \text{poly}(|N|)$, $\sum_{x \in QR_N} \text{Prob}[P(N, b_1(x), \cdots b_k(x)) = b_0(x)]/(\varphi(N)/4) \geqq (1/2)+\varepsilon$, where $b_i(x) = \text{parity}(x^{2^i} \bmod N)$.

THEOREM 4. *Modulo the* QRA, *the $x^2 \bmod N$ generator is an unpredictable (cryptographically secure) pseudo-random sequence generator. That is to say, for each probabilistic poly-time predictor* **P**, *each constant $0<\delta<1$, and positive integer t,* **P** *has at most a $1/n^t$ advantage for N in predicting sequences to the left (for sufficiently large n and for all but a δ fraction of prescribed numbers N of length n).*

Proof. Suppose we have a predictor for the $x^2 \bmod N$ generator with an ε-advantage for N. This can be converted efficiently and uniformly into a procedure with an ε-advantage in guessing parity (of x_{-1} given arbitrary x_0 in QR_N). To see this, suppose we are given $x_0 \in QR_N$. From seed x_0 generate the sequences $b_0 b_1 b_2 \cdots$. Then parity $(x_{-1}) = b_{-1}$.

Now convert (Lemma 2) to a procedure for guessing quadratic residuacity with an amplified advantage (Lemma 3) to get a contradiction to the quadratic residuacity assumption. QED

It follows from a fundamental theorem of Yao [Yao] that, under the QRA, the sequences produced by the $x^2 \bmod N$ generator pass every probabilistic polynomial-time statistical test (roughly speaking, these sequences cannot be distinguished by any poly (n)-time statistical test—with more than a negligible advantage—from sequences produced by successive flips of a fair coin). More precisely, what does this mean? Yao gives a very general definition of the concept of a probabilistic poly-time statistical test, but the following definition adequately describes such a test for our purpose: formally, a *probabilistic poly-time statistical test*, T, is a probabilistic poly-time algorithm that assigns to every input string in $\{0, 1\}^*$ a real number in the unit interval $[0, 1]$ (the particular value depends in general on the sequence of coin flips made by the algorithm). Let α_m denote the average value that such T assigns to a random m-bit string (chosen with uniform probability from among all m-bit strings). We say that a *pseudo-random sequence generator passes test T* if, for every positive integer t, the average value, over all seeds of length n, that the statistical test assigns to a poly (n)-bit pseudo-random string (produced by the given generator) lies in the interval $\alpha_{\text{poly}(n)} \pm 1/n^t$ for all sufficiently large n. If a generator does not pass test T, then we say that T has an *advantage* in distinguishing between the pseudo-random bits produced by the generator and truly random sequences of bits.

THEOREM 5 (following Yao). *Modulo the* QRA, *the sequences produced by the $x^2 \bmod N$ generator pass every probabilistic polynomial time statistical test.*

Idea of proof. Here and in the sequel, we use poly_1, poly_2, $\cdots$ to denote distinct polynomials. Suppose there were a probabilistic poly_1-time test T that, for infinitely many n, has an advantage in distinguishing between the pseudo-random sequences of

length $\text{poly}_1(n)$ produced (from random seeds of length n) by the $x^2 \bmod N$ generator and truly random sequences of bits of the same $\text{poly}_1(n)$ length. Then for some positive integer t and infinitely many n, the average value that T assigns to the pseudo-random sequences of length $\text{poly}_1(n)$ (generated from seeds of length n) lies outside, say above, $\alpha_{\text{poly}_1(n)} \pm 1/n^t$, whereas the average value it assigns (truly) random sequences lies inside $\alpha_{\text{poly}_1(n)} \pm 1/n^{t+1}$. For each of these n, we can find integers $j, k \geqq 0, j+k = \text{poly}_1(n)$ (in probabilistic $\text{poly}_2(n)$-time) such that "with high probability" the average value that T assigns to sequences in $A = \{r_1 \cdots r_j r_{j+1} b_1 \cdots b_k\}$ is closer to $\alpha_{\text{poly}_1(n)}$ by at least $1/(n^{t+1} \cdot \text{poly}_1(n))$ than the average higher value it assigns to sequences in $B = \{r_1 \cdots r_j b_0 b_1 \cdots b_k\}$—where the $b_0 \cdots b_k$ are sequences produced by the generator, the seed x_0 having been chosen uniformly at random, and the $r_1 \cdots r_{j+1}$ are sequences of independent random bits. Integers j, k are found by trying different values of j, k, in each case sampling elements of the associated sets A, B, and applying T to these samples. The Weak Law of Large Numbers assures that this can be done in probabilistic $\text{poly}_2(n)$-time.

We can convert T into a predictor for the generator: Given a sequence $b_1 \cdots b_k$ produced by the generator, we submit a sample of $\text{poly}_3(n)$-many sequences of the form $r_1 \cdots r_j 0 b_1 \cdots b_k$ (where the $r_1, \cdots, r_j$ are independently chosen random bits) to test T and estimate the average value, call it α^0, assigned by T to the entire set of these sequences. Then submit the corresponding sequences $r_1 \cdots r_j 1 b_1 \cdots b_k$ to T and estimate the associated α^1. T's "advantage" in distinguishing between pseudo-random and random sequences can now be converted into an advantage in predicting b_0 correctly: Use a biased coin to predict b_0. Set the bias so that the coin has probability $\frac{1}{2} + \frac{1}{2}(\alpha^0 - \alpha^1)$ of coming up heads. Toss the coin and if it comes up heads, predict $b_0 = 0$, else $b_0 = 1$. (It is tempting but incorrect to suggest that we predict $b_0 = 0$ if α^0 is greater than α^1, else $b_0 = 1$. Problems arise if for a *few* choices of $b_1 \cdots$, the random strings $r_1 \cdots r_j b_1 \cdots b_k$ give a correct *strong* bias toward $b_0 = 0$, whereas for *most* choices of $b_1' \cdots$, the strings $r_1 \cdots r_j b_1' \cdots b_k'$ give a *weak* bias toward $b_0' = 1$. One would end up giving wrong answers for the majority of strings $b_1' \cdots b_k'$. In this case, the expectation of predicting b_0 correctly would be less than $\frac{1}{2}$. The biased coin makes the expectation greater than $\frac{1}{2}$.) QED

Remark. We can construct another unpredictable generator as follows: recall that since $N \equiv 1 \bmod 4$, both x and $-x$ (in Z_N^*) have the same Jacobi symbol, and since N is odd, x and $-x$ have opposite parity. Therefore, the parity property partitions $Z_N^*(+1)$ in half. In a similar fashion, the location property, where location $(x) = 0$ if $x < (N-1)/2$, 1 if $x \geqq (N-1)/2$, partitions $Z_N^*(+1)$ in half. Exactly one of x and $-x$ is a quadratic residue; but which of the two is the quadratic residue is hard to decide. Thus we get the following.

THEOREM. *The modified $x^2 \bmod N$ generator, gotten by extracting the location bit at each stage (instead of parity) is cryptographically secure (modulo the quadratic residuacity assumption).*

Conjecture. The modified $x^2 \bmod N$ generator, gotten by extracting two bits at each stage, parity (x) and location (x), is cryptographically secure.

Question. Parity (x) is the least significant bit of x; we can think of location (x), in a sense, as the most significant bit. How many bits (and which ones) can we extract at each stage and still maintain cryptographic security?

8. Lengths of periods (of the sequences produced by the $x^2 \bmod N$ generator). What exactly is the period of the sequence generated by the $x^2 \bmod N$ generator? For quadratic residues $x_0 \bmod N$, let $\pi(x_0)$ be the period of the sequence $x_0, x_1, x_2, \cdots$, where $x_i = x_0^{2^i} \bmod N$. Since the $x^2 \bmod N$ generator is an unpredictable

pseudo-random sequence generator (modulo QRA), it follows that on the average, $\pi(x_0)$ will be long. In this section we investigate the precise lengths of these periods *without* relying on unproven assumptions (such as quadratic residuacity). To start, we show that the period is a divisor of $\lambda(\lambda(N))$.

DEFINITION. Let $M = 2^e * P_1^{e_1} * \cdots * P_k^{e_k}$, where $P_1, \cdots, P_k$ are distinct odd primes. Carmichael's λ-function is defined by

$$\lambda(2^e) = \begin{cases} 2^{e-1} & \text{if } e = 1 \text{ or } 2, \\ 2^{e-2} & \text{if } e > 2, \end{cases}$$

and $\lambda(M) = \text{lcm}\,[\lambda(2^e), (P_1-1) * P_1^{e_1-1}, \cdots, (P_k-1) * P_k^{e_k-1}]$. Carmichael [LeVeque], [Knuth] proves that $\lambda(M)$ is both the least common multiple and the supremum of the orders of the elements in Z_M^*. As corollary, Carmichael's extension of Euler's theorem asserts that $\alpha^{\lambda(M)} \equiv 1 \bmod M$ if $\gcd(\alpha, M) = 1$ [Knuth, vol. 2, p. 19].

The following theorem asserts that the period, $\pi(x_0)$, divides $\lambda(\lambda(N))$.

THEOREM 6. *Let N be a number of the prescribed form (that is to say, $N = P * Q$ where P, Q are distinct primes both congruent to* 3 mod 4*). Let x_0 be a quadratic residue* mod N. *Let $\pi = \pi(x_0) =$ period of the sequence $x_0, x_1, x_2, \cdots$. Then $\pi | \lambda(\lambda(N))$.*

Proof. Let $\text{ord}_N x$ denote the order of x mod N. Then for $x \in Z_N^*(+1)$, $\text{ord}_N x$ is odd, because:

(1) $\text{ord}_N x_i = \text{ord}_N x_{i+1}$. This is because
  (i) $\text{ord}_N x_{i+1} | \text{ord}_N x_i$, and
  (ii) $x_0, x_1, \cdots$ cycles.

(2) for all positive integers u, $2^u \| \text{ord}_N x_i \Rightarrow 2^{u-1} \| \text{ord}_N x_{i+1}$. (Here, $2^u \| \text{ord}_N x$ means $2^u | \text{ord}_N x$ and 2^{u+1} does not divide $\text{ord}_N x$.)

Hence, by Carmichael's extension of Euler's theorem,

$$2^{\lambda(\text{ord}_N x_0)} \equiv 1 \bmod (\text{ord}_N x_0).$$

But π is the least positive integer such that $2^\pi \equiv 1 \bmod (\text{ord}_N x_0)$, since π is the least positive integer such that $x_0 = x_0^{2^\pi} \bmod N$.

Therefore, $\pi | \lambda(\text{ord}_N x_0)$. (This is a stronger result than the statement of the theorem!)

But $\lambda(\text{ord}_N x_0) | \lambda(\lambda(N))$ since $\text{ord}_N(x_0) | \lambda(N)$ for x_0 in Z_N^*.

Therefore, $\pi | \lambda(\lambda(N))$. QED

The following theorem provides conditions under which $\lambda(\lambda(N)) | \pi(x_0)$—and therefore $\lambda(\lambda(N)) = \pi(x_0)$.

THEOREM 7. *Let N be a number of the prescribed form, x_0 a quadratic residue* mod N, *$\pi(x_0) =$ period of the sequence $x_0, x_1, \cdots$.*

1. *Choose N so that* $\text{ord}_{\lambda(N)/2}(2) = \lambda(\lambda(N))$. (*Note: this equality frequently holds for prescribed N. See Theorem* 8.)

2. *Choose quadratic residue x_0 so that* $\text{ord}_N(x_0) = \lambda(N)/2$. (*Note: one can always choose a quadratic residue x_0 this way. See the paragraph below immediately following the proof of this theorem.*)

Then $\lambda(\lambda(N)) | \pi(x_0)$ (*and therefore* $\lambda(\lambda(N)) = \pi(x_0)$).

Proof. Recall that $x_i = (x_0)^{2^i} \bmod N$, and so $\pi =$ least positive integer such that $x_\pi = (x_0)^{2^\pi} \bmod N = x_0$.

Next note that $2^\pi \bmod (\lambda(N)/2) = 1$: By $2, \cdot \lambda(N)/2 =$ least positive integer such that $x_0^{\lambda(N)/2} \bmod N = 1$. But $x_0^{2^\pi} \bmod N = x_0$, so $x_0^{2^\pi - 1} \bmod N = 1$. Therefore, $\lambda(N)/2 | 2^\pi - 1$.

Finally, we show that $\lambda(\lambda(N))|\pi$: By 1, $\lambda(\lambda(N))$ = least positive integer such that $2^{\lambda(\lambda(N))} \bmod(\lambda(N)/2) = 1$, but (we just saw), $2^{\pi} \bmod(\lambda(N)/2) = 1$. Therefore $\lambda(\lambda(N))|\pi$. QED

Condition 2 of the above theorem holds for a substantial fraction of quadratic residues, x_0 in Z_N^*. Specifically, the number of quadratic residues in Z_N^* that are of order $\lambda(N)/2 \bmod N$ is $\Omega(N/(\ln\ln N)^2)$ (where $f(n) = \Omega(g(n))$ means there exists a constant c such that $f(n) > c \cdot g(n)$ for all sufficiently large n). To derive this lower bound, let $N = P \cdot Q$. Let g_P, g_Q be generators mod P, Q respectively. Let $a \equiv g_P \bmod P$, $\equiv g_Q \bmod Q$. It is easy to see that $\operatorname{ord}_N a = \operatorname{lcm}[P-1, Q-1] = \lambda(N)$. Now there are $\varphi(\varphi(P))$ generators mod P and $\varphi(\varphi(Q))$ generators mod Q. By the Chinese remainder theorem, $Z_N^* = Z_P^* \times Z_Q^*$, so there are at least $\varphi(\varphi(P)) \cdot \varphi(\varphi(Q))$ elements in Z_N^* of order $\lambda(N)$. But $\varphi(x) > x/(6 \ln\ln x)$ for all integers $x > 2$. Hence

$$\varphi(\varphi(P)) \cdot \varphi(\varphi(Q)) = \varphi(P-1) \cdot \varphi(Q-1) \geqq \frac{P-1}{6 \ln\ln(P-1)} \frac{Q-1}{6 \ln\ln(Q-1)}$$

$$\geqq \frac{N-P-Q+1}{[6 \ln\ln(N-1)]^2} = \Omega\left(\frac{N}{(\ln\ln N)^2}\right).$$

The map $x \to x^2 \bmod N$ is 4:1. Therefore, there are at least $\Omega(N/4(\ln\ln N)^2)$ quadratic residues in Z_N^* of order $\lambda(N)/2$.

Condition 1 of the above theorem is harder to ensure in general. The following definition and theorem give conditions of special interest for our applications, under which condition 1 will hold.

DEFINITION. A prime P is *special* if $P = 2P_1 + 1$ and $P_1 = 2P_2 + 1$ where P_1, P_2 are odd primes. A number $N = P * Q$ is a *special* number of the prescribed form if and only if P, Q are distinct odd primes both congruent to 3 mod 4, and P, Q are both special (note: distinctness of P and Q implies that $P_2 \neq Q_2$).

Example. The primes 2879, 1439, 719, 359, 179, 89, are special. The number $N = 23 * 47$ is a special number of the prescribed form.

Remark. It is reasonable to expect [Shanks] that the fraction of n-bit numbers that are special primes is asymptotically $1/((\ln P)(\ln P_1)(\ln P_2))$, which is asymptotically $1/(n^3 \ln^3 2)$ since $2^n < P < 2^{n+1}$, $2^{n-1} < P_1 < 2^n$, and $2^{n-2} < P_2 < 2^{n-1}$. It follows that there is an efficient, i.e., polynomial (n), probabilistic algorithm to find special n-bit primes: simply generate n bit numbers at random and use a probabilistic primality test [Strassen–Solovay], [Miller], [Rabin '80] to select the ones that are special.

THEOREM 8. *Suppose N is a special number of the prescribed form, and that 2 is a quadratic residue with respect to at most one of P_1, Q_1.*[5] *Then* $\operatorname{ord}_{\lambda(N)/2}(2) = \lambda(\lambda(N))$ *(and therefore $\lambda(\lambda(N)) = \pi(x_0)$ for some x_0).*

Proof. For N of the prescribed form, $\lambda(N) = \operatorname{lcm}[2P_1, 2Q_1] = 2P_1Q_1$, and $\lambda(\lambda(N)) = \operatorname{lcm}[2P_2, 2Q_2] = 2P_2Q_2$. It is easy to see that $\lambda(\lambda(N)/2) = \lambda(\lambda(N))$, so $\operatorname{ord}_{\lambda(N)/2}(2) | \lambda(\lambda(N))$. Therefore, $\operatorname{ord}_{\lambda(N)/2}(2) | 2P_2Q_2$.

Assume to the contrary that $\operatorname{ord}_{\lambda(N)/2} 2 \neq 2P_2Q_2$. Then either $\operatorname{ord}_{\lambda(N)/2}(2) = P_2Q_2$ or $\operatorname{ord}_{\lambda(N)/2}(2) | 2P_2$ or $\operatorname{ord}_{\lambda(N)/2}(2) | 2Q_2$. Without loss of generality, we may assume that $\operatorname{ord}_{\lambda(N)/2} 2 | 2P_2$ or $\operatorname{ord}_{\lambda(N)/2}(2) = P_2Q_2$.

[5] Roughly three fourths of all special numbers of the prescribed form satisfy this additional condition (that 2 is a quadratic residue with respect to at most one of P_1 and Q_1). The condition is needed: for example, the special number in prescribed form, $N = 719 * 47$, fails this condition (for this N, $\operatorname{ord}_{\lambda(N)/2}(2) = \lambda(\lambda(N))/2$).

Case 1. $\text{ord}_{\lambda(N)/2}\, 2 | 2P_2$.

Then $2^{2P_2} \equiv 1 \bmod (\lambda(N)/2) \equiv 1 \bmod P_1Q_1$.

Therefore $2^{2P_2} \equiv 1 \bmod Q_1$.

But $2^{2Q_2} \equiv 1 \bmod Q_1$ since $Q_1 = 2Q_2 + 1$, by Fermat's Little Theorem.

Therefore $2^{\gcd(2P_2, 2Q_2)} \equiv 1 \bmod Q_1$.

Therefore $2^2 \equiv 1 \bmod Q_1$. This is a contradiction (since $Q_2 \geqq 3$ and therefore $Q_1 \geqq 7$).

Case 2. $\text{ord}_{\lambda(N)/2}(2) = P_2Q_2$. Then $2^{P_2Q_2} \equiv 1 \bmod P_1Q_1$, which implies $2^{P_2Q_2} \equiv 1 \bmod Q_1$, which implies $2^{Q_2} \not\equiv -1 \bmod Q_1$ since P_2 is odd. Therefore, $2^{(Q_1-1)/2} \not\equiv -1 \bmod Q_1$. Therefore, 2 is a quadratic residue with respect to Q_1. Similarly, 2 is a quadratic residue with respect to P_1. Contradiction. QED

Open question. Let $\pi_b(x_0)$ be the period of the sequence $b_0 b_1 \cdots$ produced by the $x^2 \bmod N$ generator with input (N, x_0). Then $\pi_b(x_0) | \pi(x_0)$. What is the exact relation between $\pi_b(x_0)$ and $\pi(x_0)$? Are they generally equal?

9. Algorithms for determining length of period and random accessing. The following two theorems provide algorithms for determining

(1) $\pi(x_0)$, the period of the $x^2 \bmod N$ sequence that begins with x_0, and

(2) the ith element x_i.

These will be useful in the cryptographic applications.

THEOREM 9. *There exists an efficient algorithm A which, when given any N of the prescribed form,*[6] $\lambda(N)$, $\lambda(\lambda(N))$ *AND the factorization of* $\lambda(\lambda(N))$, *efficiently determines the period* $\pi(x_0)$ *of any quadratic residue* x_0 *in* Z_N^*, *i.e.,* $A[N, \lambda(N), \lambda(\lambda(N)),$ *factorization of* $\lambda(\lambda(N)), x_0] = \pi(x_0)$.

Proof. Let $\pi = \pi(x_0)$.

Recall that

(1) $x_i = (x_0)^{2^i} \bmod N$ and $x_\pi = (x_0)^{2^\pi} \bmod N = x_0$.

(2) $\pi | \lambda(\lambda(N))$ (by Theorem 6).

Therefore, $(x_0)^{2^{\lambda(\lambda(N))}} \bmod N = x_0$.

It follows that $(x_0)^{2^{\lambda(\lambda(N)) \bmod \lambda(N)}} \bmod N = x_0$ (by Carmichael's extension of Euler's theorem, $\alpha^{\lambda(N)} \equiv 1 \bmod N$ if $\gcd(\alpha, N) = 1$; therefore $x_0^{\lambda(N)} \equiv 1 \bmod N$; therefore $x_0^{2^{\lambda(\lambda(N))} + k\lambda(N)} \bmod N = x_0$).

Therefore, from knowledge of $\lambda(N)$, $\lambda(\lambda(N))$, and the factorization of $\lambda(\lambda(N))$, one can efficiently determine π: look for the largest $d | \lambda(\lambda(N))$ such that $(x_0)^{2^{\lambda(\lambda(N))/d \bmod \lambda(N)}} \bmod N = x_0$. Then $\pi = \lambda(\lambda(N))/d$. QED

THEOREM 10a. *There exists an efficient deterministic algorithm A such that given* $N, \lambda(N)$, *any quadratic residue* x_0 *in* Z_N^*, *and any positive integer i, A efficiently computes* x_i, *i.e.,*

$$A[N, \lambda(N), x_0, i] = x_i.$$

Proof. $x_i = x_0^{2^i \bmod \lambda(N)} \bmod N$.

The number of steps to compute x_i in this fashion, given N, $\lambda(N)$, x_0 and i, is $O(|N|^{2+\varepsilon})$ using fast multiplication. QED

THEOREM 10b. *There exists an efficient deterministic algorithm A such that given any N of the prescribed form,* $\lambda(N)$, *any quadratic residue* x_0 *in* Z_N^*, *and any positive integer i, A efficiently computes* x_{-i} (*note the negative subscript*), *i.e.,* $A[N, \lambda(N), x_0, i] = x_{-i}$.

Proof. [Miller] has shown how to efficiently factor $N = P \cdot Q$ given $\lambda(N)$. The proof of Theorem 3 shows that $\sqrt{x_0} \bmod P = \pm x_0^{(P+1)/4}$. Exactly one of these two

[6] $N = P * Q$, where P, Q are primes congruent to 3 mod 4.

roots is a quadratic residue since -1 is *not* a quadratic residue mod P for $P \equiv 3 \bmod 4$. Therefore, $x_{-1} \bmod P = x_0^{P+1/4}$ or $-x_0^{P+1/4}$. Similarly, $x_{-2} \bmod P = \pm x_0^{(P+1/4)^2}$ (since $(-1)^{P+1/4} = \pm 1$). Continuing, $x_{-i} \bmod P = \pm x_0^{(P+1/4)^i} \bmod P = \pm x_0^{(P+1/4)^i \bmod (P-1)}$ mod P, which can be computed efficiently. From $x_{-i} \bmod P$ and $x_{-i} \bmod Q$, The Chinese Remainder Theorem enables one to efficiently compute x_{-i}. QED

Conversely, the following theorem asserts that an algorithm that knows the period, π, and for any i can obtain the ith element x_i of the sequence $x_0, x_1, \cdots$ obtained by squaring mod N can factor N.

THEOREM 11. *Let O denote an oracle such that $O(N, x_0, i) = \langle \pi, x_i \rangle$, where $\pi = \pi(x_0)$. There is an efficient probabilistic algorithm A^O such that $A^O(N) = P$ or Q, for $N = P * Q$.*

Proof. The algorithm A^O works as follows:

Search at random for $y \in Z_N^*$ such that $(y/N) = -1$ (half the elements of Z_N^* have Jacobi symbol -1 with respect to N). Set $x_0 = y^2 \bmod N$. Ask the Oracle for π, then for $x_{\pi-1}$ (recall: $x_\pi = x_0$). Then $y^2 = (x_{\pi-1})^2 = x_\pi = x_0 \bmod N$. But $y \neq \pm x_{\pi-1}$ since $(y/N) = -1$ and $(\pm x_{\pi-1}/N) = +1$. Therefore, $\gcd(y + x_{\pi-1}, N) = P$ or Q (by elementary number theory). QED

Open question. Can an algorithm use an oracle that outputs just x_i—namely, $O(N, x_0, i) = x_i$—to factor N?

Open question. Can an algorithm use an oracle that outputs just π—namely, $O(N, x_0) = \pi$—to factor N?

10. Applications. (1.1) The $1/P$ generator (base 2) is useful for constructing (generalized) de Bruijn sequences. These have applications, for example, in the design of radar for environments with extreme background noise [Golomb]. We believe there may be additional interesting applications making use of properties identified in this paper, particularly the property that from $2|P|+1$ but *not* $|P|-1$ quotient digits, one can infer the sequence backwards and forwards. For example, one could split a key, P, between two parties—by giving $|P|$ successive quotient digits to each so that together they have $2|P|$ successive digits. Neither party alone would have the slightest information *which* prime, P, was key, but cooperatively they could determine P efficiently.

(1.2) Maximum-length shift-register sequences (which are closely related to the $1/P$ *generator*) are used for encryption of messages [Golomb]. We view the inference procedure given here as yet another step toward breaking such crypto-systems.

(2.1) The $x^2 \bmod N$ sequence can be used for *public-key cryptography*: Alice can enable Bob to send messages to her (over public channels) that only she can read. Alice constructs and publicizes a number N_A, her *public key*, with the prescribed properties: $N_A = P_A * Q_A$ where P_A and Q_A are distinct equal length randomly chosen primes both congruent to 3 mod 4. She keeps private the primes P_A and Q_A, her *private key*.

Bob encrypts: Suppose Bob wants to send a k-bit message $\vec{m} = (m_1, \cdots, m_k)$, where $k = \text{poly}(|N_A|)$, to Alice. Using Alice's public key, Bob constructs a *one-time pad*: he selects an integer x_0 from $Z_{N_A}^*$ at random, squares it mod N_A to get a quadratic residue x_1, and uses the $x^2 \bmod N$-generator with input (N_A, x_1) to generate the one-time pad $\vec{b} = (b_1, \cdots, b_k)$. Bob then sends BOTH the encrypted message, $\vec{m} \oplus \vec{b} = (m_1 \oplus b_1, \cdots, m_k \oplus b_k)$, AND x_{k+1} to Alice over public channels, where $\oplus$ is the exclusive-or.

Alice decrypts: From her knowledge of P_A and Q_A, her private key, Alice has enough information to efficiently compute $x_k, x_{k-1}, \cdots, x_1$ from x_{k+1} by backwards

jump (Theorem 3). From that, she reconstructs the one-time pad $\vec{b}$ and, by $\oplus$-oring $\vec{b}$ with the encrypted message, decrypts the message, $\vec{m}$.

Anyone who can reconstruct (i.e., guess with some advantage) even one bit of $\vec{m}$ from knowledge of k and x_{k+1} can thereby obtain (guess with some advantage) a bit of the one-time pad $\vec{b}$. This is impossible (by the quadratic residuacity assumption and the following theorem) if $\vec{m}$ is a randomly selected message.

THEOREM 12 (stronger version of claim 3). *Let* poly *be a polynomial. Let* $0 < \delta < 1$. *Let t be a positive integer. Then for all but a* δ*-fraction of numbers N of the prescribed type*[8]*, the factors of N are necessary—assuming they are necessary for deciding quadratic residuacity of x in* $Z_N^*(+1)$*—to have even an* ε*-advantage,*[7] $\varepsilon = 1/|N|^t$, *in guessing in* poly*-time any pair* (l, b_l) *(i.e., any bit* b_l *and its location l in the sequence* $b_1, \cdots, b_k$*),* $1 \leqq l \leqq k = \text{poly}(|N|)$ *given N and* x_{k+1}*, where* $b_l = \text{parity}(x_l)$.

Proof. Assume to the contrary that the probabilistic poly-time procedure **P** has an ε-advantage in guessing a pair. This **P** can be used to obtain a probabilistic poly-time procedure that has an $\varepsilon/\text{poly}(|N|)$-advantage in deciding quadratic residuosity of a *randomly-chosen* $x \in Z_N^*(+1)$: Select l, $1 \leqq l \leqq \text{poly}(|N|)$, *at random* with the uniform probability distribution, set $x_{l+1} = x^2 \bmod N$, and generate x_{k+1}. Compute $\mathbf{P}[N, x_{k+1}]$. The chances are $1/\text{poly}(|N|)$ that $\mathbf{P}[N, x_{k+1}] = (l, b)$ for the above-chosen l and *some* b. If so, then guess that x is a quadratic residue if and only if parity $(x) = b$. If not, toss a fair coin to decide quadratic residuacity of x. The advantage (in guessing quadratic residuacity of x) will be $\varepsilon/\text{poly}(|N|)$. QED

(2.2) Having constructed a number $N_A = P_A \cdot Q_A$ with the prescribed properties, Alice can compute $\lambda(N)$ and use it, by Theorem 10a, to quickly compute $x_i = x_0^{2^i \bmod \lambda(N)} \bmod N$ (for any $x_0 \in QR_N$). This means she can use word i as address to retrieve word x_i or bit b_i efficiently—as if the $x^2 \bmod N$ generator were a random access memory that is storing a pseudo-random sequence. [Brassard] has suggested applications, e.g., to the construction of unforgeable subway tokens, where this jumping ahead is desirable.

(2.3) Cryptographically secure pseudo-random sequence generators (such as the $x^2 \bmod N$ generator) may be viewed as *amplifiers* of randomness (short random strings are amplified to make long pseudo-random strings).

(2.4) One often uses pseudo-random sequences (rather than random sequences) because they are reproducible [von Neumann]. For the pseudo-random sequences produced by the $x^2 \bmod N$ generator, one has only to store a short seed in order to reproduce a long sequence; one does not have to store the entire random sequence.

11. Brief history relevant to this paper. W. Diffie and M. Hellman [Diffie–Hellman] first introduce the notion of a trapdoor function and public-key cryptography.

R. Rivest, A. Shamir and L. Adleman [Rivest–Shamir–Adleman] propose the first concrete example (and implementation to public-key cryptography) of a trapdoor function relying on (but not yet proved equivalent to) a number theoretic conjecture (which they propose) that factoring is hard. The RSA trapdoor function is $x^s \bmod N$ (where $N = P \cdot Q$, P, Q are distinct odd primes and $\gcd(s, \varphi(N)) = 1$). Later [Shamir–Rivest–Adleman] utilize a private-key commutative function in their solution to the problem of mental poker posed by R. Floyd.

---

[7] DEFINITION. A probabilistic poly-time procedure $\mathbf{P}[N, x_{k+1}]$ has an ε*-advantage for N in guessing a pair* (l, b_l), $1 \leqq l \leqq k = \text{poly}(|N|)$ (given arbitrary x_{k+1} selected uniformly from QR_N) if and only if

$$\left(\frac{\sum_{x_{k+1} \in QR_N} \text{Prob}\left(V_{l=1}^{k} \{\mathbf{P}[N, x_{k+1}] = (l, \text{parity}(x_l))\}\right)}{(\varphi(N))/4} \right) \geqq (1/2) + \varepsilon.$$

M. O. Rabin [Rabin '79] introduced for his signature scheme the many-one trapdoor $x^2 \bmod N$ (where $N = P \cdot Q$ for distinct odd primes P, Q), which he proves is as hard to invert as factoring.

M. Blum [Blum] for his coin-flipping algorithm first chose $P \equiv Q \equiv 3 \bmod 4$ to construct a trapdoor (the 3 mod 4 trapdoor) $x^2 \bmod N$ (as hard to invert as factoring) which is $1-1$ on the quadratic residues mod N.

S. Goldwasser and S. Micali [Goldwasser–Micali] use these properties (of the $x^2 \bmod N$ trapdoor and the 3 mod 4 trapdoor) and the quadratic residuacity assumption which they first propose, to construct a protocol for mental poker and an encryption scheme that hides partial information. This directly addresses the problem pointed out by R. Lipton [Lipton] that partial information can be preserved and transmitted by trapdoor functions (e.g. the set of quadratic residues is invariant under the RSA function) giving rise to an advantage, and enabling trapdoors to be inverted on certain message spaces.

A. Shamir [Shamir] proposed the first example (based on RSA) of a cryptographically strong (i.e. polynomial-time unpredictable) pseudo-random sequence generator.

M. Blum and S. Micali [Blum–Micali] present general conditions on predicates that will ensure a cryptographically strong generator. Using these conditions and the Discrete Logarithm Conjecture they construct cryptographically strong sequences of pseudo-random bits.

A. Yao [Yao], in his foundational paper on complexity based information theory, constructs a "perfect" pseudo-random sequence generator on the very general assumption that there exists a so-called "stable" one-way function.

Our $x^2 \bmod N$ generator is based directly on a 3 mod 4 trapdoor and the *QRA*. We believe that the 3 mod 4 scenario, because of its nice mathematical properties (e.g. Lemma 1) will continue to lead to fruitful applications. We also believe that an in-depth analysis of sequences produced by unpredictable pseudo-random sequence generators, as begun in this paper, will provide useful information concerning the nature of these generators, and lead to insights about the number theoretic assumptions that have been made.

Acknowledgments. We thank Silvio Micali for pointing us to the literature on de Bruijn sequences, and for his numerous helpful and encouraging suggestions. We are grateful to a number of people for valuable discussions on this work, including G. Brassard, S. Even, A. Lempel, L Levin, J. Plumstead, M. O. Rabin, D. Rich, S. Smale, R. Solovay and A. Yao. Umesh Vazirani provided a necessary ingredient for our proof of Theorem 5 [Yao]; Rene Peralta did the same for Theorem 10b.

Note added in proof. The assertion "modulo the *QRA*" and/or its equivalent in Theorems 4, 5 and 12 can be replaced by "modulo the assumption that factoring is hard". This is a consequence of the main theorem in W. ALEXI, B. CHOR, O. GOLDREICH AND C. P. SCHNORR, *RSA/Rabin bits are* $(\frac{1}{2})+(1/\text{poly}(\log N))$ *secure*, IEEE 25th Symposium on Foundations of Computer Science, 1984, pp. 449–457.

REFERENCES

[1] L. ADLEMAN, *On distinguishing prime numbers from composite numbers*, Proc. 21st IEEE Symposium on Foundations of Computer Science, 1980, pp. 387–408.

[2] E. BACH, *How to generate random integers with known factorization*, submitted for publication.

[3] P. BILLINGSLEY, *Ergodic Theory and Information*, John Wiley, New York, 1965.

[4] M. BLUM, *Coin flipping by telephone*, in Proc. IEEE Spring COMPCON, 1982, pp. 133–137.

[5] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo random bits*, IEEE 23rd Symposium on the Foundations of Computer Science (1982), pp. 112–117.

[6] G. BRASSARD, *On computationally secure authentication tags requiring short secret shared keys*, in Advances in Cryptology, Proc. of Crypto 82, ed. D. Chaum, R. L. Rivest and A. T. Sherman, Plenum Press, New York, 1983, pp. 79–86.
[7] L. DICKSON, *History of the Theory of Numbers*, Chelsea Pub. Co., 1919 (republished 1971).
[8] W. DIFFIE AND M. HELLMAN, *New directions in cryptography*, IEEE Trans. Inform. Theory, IT-22 (Nov. 1976), pp. 644–654.
[9] S. EVEN, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.
[10] C. G. GAUSS, *Disquisitiones Arithmeticae*, 1801; reprinted in English transl. by Yale Univ. Press, New Haven, CT, 1966.
[11] S. GOLDWASSER AND S. MICALI, *Probabilistic encryption and how to play mental poker keeping secret all partial information*, 14th STOC, 1982, pp. 365–377.
[12] S. GOLOMB, *Shift Register Sequences*, Aegean Park Press, 1982.
[13] J. HOPCROFT AND J. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, MA, 1969.
[14] M. KAC, *What is randomness?*, American Scientist, 71 (August 1983), pp. 405–406.
[15] D. KNUTH, *The Art of Computer Programming: Vol. 2, Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1981.
[16] W. LEVEQUE, *Fundamentals of Number Theory*, Addison-Wesley, Reading, MA, 1977.
[17] R. LIPTON, *How to cheat at mental poker*, Univ. California, Berkeley, preliminary report, August 1979.
[18] G. MILLER, *Riemann's hypothesis and tests for primality*, Ph.D. thesis, Univ. California, Berkeley, 1975.
[19] J. PLUMSTEAD, *Inferring a sequence generated by a linear congruence*, IEEE 23rd Symposium on Foundations of Computer Science, 1982, pp. 153–159.
[20] S. POHLIG AND M. HELLMAN, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Trans. Inform. Theory, IT-24 (1978), pp. 106–110.
[21] M. O. RABIN, *Digital signatures and public-key functions as intractable as factorization*, MIT/LCS/TR-212 Tech. memo, Massachusetts Institute of Technology, 1979.
[22] ———, *Probabilistic algorithm for testing primality*, J. Number Theory, 12 (1980), pp. 128–138.
[23] R. RIVEST, A. SHAMIR AND L. ADLEMAN, *A method for obtaining digital signatures and public key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.
[24] A. SHAMIR, R. RIVEST AND L. ADLEMAN, *Mental poker*, in The Mathematical Gardner, D. Klarner, ed., Wadsworth, New York, 1981, pp. 37–43.
[25] A. SHAMIR, *On the generation of cryptographically strong pseudo-random sequences*, ICALP, 1981.
[26] D. SHANKS, *Solved and Unsolved Problems in Number Theory*, Chelsea, New York, 1976.
[27] R. SOLOVAY AND V. STRASSEN, *A fast Monte-Carlo test for primality*, this Journal, 6 (1977), pp. 84–85.
[28] J. VON NEUMANN, *Various techniques used in connection with random digits*, Collected Works, vol. 5, Macmillan, New York, 1963, pp. 768–770.
[29] A. YAO, *Theory and applications of trapdoor functions*, IEEE 23rd Symposium on Foundations of Computer Science, 1982, pp. 80–91.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
004

EVALUATING RATIONAL FUNCTIONS: INFINITE PRECISION IS FINITE COST AND TRACTABLE ON AVERAGE*

LENORE BLUM† AND MICHAEL SHUB‡

Abstract. If one is interested in the computational complexity of problems whose natural domain of discourse is the reals, then one is led to ask: what is the "cost" of obtaining solutions to within a prescribed absolute *accuracy* $\varepsilon = 1/2^s$ (or *precision* $s = -\log_2 \varepsilon$)? The *loss of precision* intrinsic to solving a problem, independent of method of solution, gives a lower bound on the cost. It also indicates how realistic it is to assume that basic (arithmetic) operations are exact and/or take one step for complexity analyses. For the relative case, the analogous notion is the *loss of significance.*

Let $P(X)/Q(X)$ be a rational function of degree d, dimension n and real coefficients of absolute value bounded by ρ. The loss of precision in evaluating P/Q will depend on the input x, and, in general, can be arbitrarily large. We show that, w.r.t. normalized Lebesgue measure on B_r, the ball of radius r about the origin in R^n, the average loss is small: loglinear in d, n, ρ, r, and K_Q, a simple constant.

To get this, we use techniques of integral geometry and geometric measure theory to estimate the volume of the set of points causing the denominator values to be small. Suppose $\varepsilon > 0$ and $d \geqq 1$. Then:

THEOREM. *Normalized volume* $\{x \in B_r \mid |Q(x)| < \varepsilon\} < \varepsilon^{1/d} K_Q^{1/d} nd(d+1)/2r$.

An immediate application is a loglinear upper bound on the average loss of significance for solving systems of linear equations.

Key words. loss of precision/significance, condition of a problem, rational functions, average case, integral geometry, models of real computation

1. Introduction.

One approach to the analysis of the computational complexity of algebraic problems, such as the cost of evaluating rational functions over the reals, presupposes a model of computation with all arithmetic operations exact and of unit cost, for example, the real number model (Borodin-Munro [1], Knuth [6]). *How do results concerning such an idealized infinite continuous model apply to the finite discrete process of computing on actual machines*? For example, in the real number model there is no problem to decide given real x if $x \neq 0$, and if so then to compute $1/x$. However, on any real computer, when x is ultimately presented digit by digit, the "marking time" to observe the first nonzero digit of x, as well as the "input precision" for x needed to compute $1/x$ to within absolute accuracy $1/2^s$, can be arbitrarily large. If x is presented in floating point notation, the computation of $1/(1-x)$ in the unit interval to relative accuracy ε exhibits analogous problems.

How do these factors influence the actual cost and reliability of real computation, particularly as we move away from these simple examples to more interesting ones? For instance, given an invertible $n \times n$ real matrix A, what is the "input precision" needed to specify the entries of A in order to compute $1/\det A$ to within accuracy $1/2^s$ (Moler [9])? Such questions, dealing with tolerable round-off error and achievable accuracy are often avoided in approaches to algebraic complexity theory that assume infinite precision. As Knuth says in [6, p. 486], they are "beyond the scope of this book".

In this paper we address some of these questions with regard to the problem of evaluating rational functions. One approach might be to analyze achievable accuracy

* Received by the editors December 28, 1983, and in revised form January 22, 1985.

† Mills College, Oakland, California 94613, Department of Mathematics, University of California, Berkeley, California 94720. The work of this author was supported in part by NSF-VPW grant RII-8310570 and the Letts-Villard Chair, Mills College, Oakland, California. This work was done in part while the author was a National Science Foundation Visiting Professor at The Graduate Center, City University of New York.

‡ IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598 and City University of New York, New York, New York 10036. The work of this author was supported in part by National Science Foundation grant MCS 82-01267.

and related costs assuming given input size or machine precision. In an attempt to perhaps more fully interpret and apply results concerning the continuous model, we take a somewhat opposite approach. That is, we ask, given a rational function $P(X)/Q(X)$ of n variables over the reals, and desired accuracy $1/2^s$, what "input precision" is needed to achieve this desired accuracy? Clearly, the answer will depend on s, on P and Q (i.e., the number of variables, the degrees and coefficients), and on the input $x=(x_1, \cdots, x_n)$. It is unbounded in general. Surprisingly, however, we show (Theorem 4 in § 5) that the average input precision sufficient for x in B_r, the ball of radius r about the origin in $\mathbf{R}^n$, is finite and exceedingly "tractable". Here average means with respect to normalized Lebesgue measure on B_r. This tractability result, as well as others in this paper, is explicit rather than asymptotic.

Our main tool is a formula which enables us to estimate the volume of the set of inputs causing the denominator values to be small. In its normalized version, we get the following elegant estimate (in § 4).

MAIN THEOREM.

$$\frac{\text{Vol}\{x \in B_r \mid |Q(x)| < \varepsilon\}}{\text{Vol}\{B_r\}} < C_Q \frac{\varepsilon^{1/d}}{r}$$

where $Q: \mathbf{R}^n \to \mathbf{R}$ *is a polynomial of degree* $d \geqq 1$, r, ε *are positive real numbers,* $C_Q = K_Q^{1/d} nd(d+1)/2$ *and* K_Q *is a simple constant depending on the coefficients of* Q (*see* § 4).

This result allows us to estimate, for example, the average "marking time" to determine that $Q(x)$ is not zero for $x \in B_r$ (Theorem 3 in § 5).

We prove the Main Theorem using techniques from geometric measure theory and integral geometry (Federer [2] and Santalo [11]), thus using methods that are somewhat new to computational complexity theory.

We address questions both of absolute and relative accuracy. However, unless otherwise stated, "accuracy" will mean "absolute accuracy". The results on "relative accuracy" are included in § 6.

Remark 1. *Computational complexity and the condition/loss of precision of a problem.* If one is interested in the computational complexity of problems whose natural domain of discourse is the reals (or complex numbers), then one is led, both naturally and necessarily, to ask: What is the "cost" of obtaining solutions to within a *prescribed accuracy* $\varepsilon = 1/2^s \leqq 1$ (or equivalently, to within a *prescribed precision* $s = -\log_2 \varepsilon$)?

Clearly, a satisfactory answer must resolve a number of issues relating to tolerable error and the number of bit and/or basic operations required. In particular, we at least must answer the following two questions:

(1) What is the necessary and sufficient *input accuracy* δ (or equivalently, *input precision* $-\log \delta$) required for the data in order to obtain a solution to within *output accuracy* ε? Here δ will in general depend on both ε and the data, and we assume that both ε and δ are positive real numbers less than or equal to 1. (If $\delta > 1$, then we will define the *associated precision* to be 0.) The ratio ε/δ is a measure of the *condition* of the problem (Henrici [4], Wilkinson [17]) and will be "very large" for *ill-conditioned problems*, i.e., for problems where the necessary *input precision* $|\log \delta|$ is "much larger" than the prescribed *output precision* $|\log \varepsilon|$ or where the *loss of precision*

$$\log^+ \frac{\varepsilon}{\delta} = \begin{cases} |\log \delta| - |\log \varepsilon| & \text{if } \varepsilon/\delta > 1, \\ 0 & \text{otherwise} \end{cases}$$

is "great". *Thus, ill-conditioned problems are generally not computationally tractable* (nor

are they computationally stable—small perturbations of the input can result in large changes in the output).[1]

(2) Given input data of precision $-\log \delta$, what is the cost, (e.g. in terms of the number of basic arithmetic operations needed) of evaluating the solution of within output precision $-\log \varepsilon$?

The condition of a problem (1) is often investigated in numerical analysis; the basic number of steps required (2) is generally investigated in complexity theory. Clearly, a deeper understanding of the complexity issues arising in the real (or complex) case requires an understanding of *both* issues.

The condition and loss of precision are measures intrinsic to a problem and independent of method of solution. However, we remark that from a computational point of view, the *loss of precision* appears more natural than the *condition.* For one, the loss of precision gives a lower bound on the complexity of a problem. But also, the mathematics helps affirm its naturalness even more. For example, a simple computation shows that, while the average (with respect to the uniform distribution) condition of the problem of evaluating $1/x$ in the unit interval is infinite, the average loss of precision (i.e., average $\log^+$ of the condition) is very small, consistent with our intuition. (See § 2 and also § 6 for an example in the relative case.) Thus, we propose that one focus on the loss of precision, rather than the condition of a problem, for purposes of computational complexity.

In this paper we investigate the condition, more precsiely, the loss of precision in evaluating rational functions with respect to an average case analysis. This point of view is new. Our results show that, although the problem of evaluating rational functions can be arbitrarily badly ill-conditioned, the average loss of precision (and standard deviation) is small.

An anlysis of average cost now just incorporates known results about the number of arithmetic operations to evaluate rational functions, as well as the cost of multiplying m-bit numbers (Borodin-Munro [1], Knuth [6]). Thus even on the level of bit operations required, our results imply that the problem of evaluating, to accuracy $1/2^s$, rational functions of n variables, of degree d, whose coefficients lie in a ball of radius ρ, on a point x in a ball of radius r about the origin in R^n, is on the average tractable in s, n, d, and T, the maximum number of nonzero terms.

Remark 2. *Interpretation and application.* We can interpret our results as saying that computation of rational functions of infinite precision real numbers is tractable on the average, as long as one only uses as much precision as one needs. It suggests a methodology for the design of algorithms to compute such functions: start with the input precision sufficient on the average to achieve the desired accuracy and increase as necessary, e.g. by the standard deviation.

We give a general formula for input precision sufficient on the average that holds for all rational functions. Hence the result implies the average tractability of many problems such as the computation of $1/\det A$ for an invertible $n \times n$ matrix A, the inversion of the matrix by Cramer's rule, the estimation of the average logarithm of the condition number of such a matrix, and other problems expressable by a polynomial number of rational functions. Of course, in any specific problem one expects to do better. For example, Norman Weiss [16] has shown us that the exponent of ε in the

[1] For a problem whose underlying function P is differentiable, an infinitesimal version of the *condition* of the problem at input x is the size $|P'(x)|$ of the linear operator $P'(x)$ [Henrici]. The analogous measure of *loss of precision* would be $\log^+ |P'(x)|$. Results similar to those in this paper can be achieved using these notions.

computation of the volume estimate for the determinant in any dimension can be taken equal to 1 instead of $1/d$ and for the discriminant of polynomials of one variable of degree d, the exponent is trivially seen to be no worse than $\frac{1}{2}$.

In general, sharper volume estimates for polynomials should be possible depending on such criteria as irreducibility, the lower coefficients, etc. and these would be interesting. However, the example $P(x) = x^d$ demonstrates the sharpness of our results for the general case with regard to the exponent of ε.

Remark 3. *Abstract models of computation: recursive analysis.* Our results have implications for recursive analysis, the abstract theory of computation of functions on the reals (or complex numbers). A model for such a theory of computation could be based on the notion of function-oracle Turing machines. (See Ko-Friedman [7] for the development of a formal theory). *Informally*, we imagine a Turing machine M_f for computing a real function f being fed, by oracle, a real input x digit by digit as necessary. Computations are performed by M_f in the usual oracle Turing machine manner. Outputs are produced digit by digit converging to a sequence representing the real value $f(x)$. Computable real functions are then those functions that can be realized in this model, and are necessarily continuous (on their domains). The complexity of functions is measured by the cost (in terms of time and space) of producing outputs that are accurate to within ε. Hence, in this model, polynomial-time computable functions must have polynomially bounded moduli of continuity (i.e., input precision $-\log \delta$ which is polynomial in the desired output precision $-\log \varepsilon$). Thus, rational functions are not in general polynomial-time computable.

Our results show that if in this model we extend the notion of "tractable" to mean "polynomial time computable on the average", then we significantly extend the class of tractable functions in a way that is both natural and useful. In so doing, we also show how methods from integral geometry and probability theory can be used to obtain results in recursive analysis.

Remark 4. *Relation to other work.* Much of our work and techniques are motivated by Smale [13] and Shub-Smale [12]. Myong Hi Kim [5] is doing an analysis of the finite precision analogue of the real number model results on average tractability of Newton-Euler iteration schemes for finding approximate zeros of polynomials. Smale [14] is also investigating general notions for the condition (or loss of precision) of a problem.

The first draft of this paper, including the main results on sufficient input precision, was finished in the fall of 1983. In Steve Smale's seminar in the spring of 1984, Blum suggested that the log of the condition was the appropriate concept for study. Smale then introduced the expression "loss of precision" and asked what the average loss was for linear systems. This motivated the facile application of our main results in § 7. Using special techniques Eric Kostlan [8] and Adrian Ocneanu [10] get sharper results for linear systems.

Remark 5. *Further directions and results.* The results can be easily extended to the complex case, the exponent of ε changes to $2/d$, and generalized to semi-analytic functions. R. Hardt [3] has proven a volume estimate for these functions which should be sufficient to conclude the average tractability of the loss of precision function for a single f. Also, averages could be computed with respect to other probability measures, reflecting nonuniform distributions inherent in classes of naturally arising problems.

Remark 6. *Organization of paper.* We begin with three simple examples (in § 2) to illustrate key points. In § 3 we outline our procedure for the analysis of input precision sufficient to evaluate a rational function to within accuracy ε. In § 4 we estimate the volume of the set of points on which a polynomial has values near zero.

Using this, we estimate, in § 5, the average marking time to determine that a polynomial is not zero. This enables us to estimate the input precision sufficient on average. Section 6 deals with floating point notation and the relative case. In § 7 we give an immediate application of the main results of the paper to the problem of solving linear systems.

2. Three simple examples. We start with three elementary examples in order to illustrate key points.

(1) Let $x \in [0, 1]$ and let $l(x) = |\log_2(x)|$. Then $m(x) = \lfloor l(x) \rfloor + 1$ is sufficient "marking time" to observe the first nonzero entry in a binary expansion of x. Although $m(x)$ can be arbitrarily large, its average or expected value Av m is less than 2.5 since

$$\text{Av } m < \int_0^1 (l(x)+1)\, dx = \log_2 e + 1.$$

(2) Suppose that an integer $s \geqq 1$ is given and that $x \in [0, 1]$. We wish to compute $1/x$ to within absolute accuracy $1/2^s$. If $|x - x^*| < 1/2^{s+1+2l(x)+\varepsilon}$ where $\varepsilon = 1/2^{s+1}$, it is not hard to see that $|1/x - 1/x^*| < 1/2^{s+1}$ (and if $|1/x^* - y^*| < 1/2^{s+1}$ then $|1/x - y^*| < 1/2^s$). Thus, an "input precision" $\Pi_s(x)$ for x sufficient to compute $1/x$ to within absolute accuracy $1/2^s$ (allowing truncation in the answer) is $s+1+\lceil 2l(x)+\varepsilon \rceil$, which again can be arbitrarily large depending on x. In this case, the average is

$$\text{Av } \Pi_s = \int_0^1 \Pi_s(x)\, dx \leqq \int_0^1 (s+1+\lceil 2l(x)+\varepsilon \rceil)\, dx < \int_0^1 (s+1+2l(x)+1)\, dx$$
$$= s + 2\log_2 e + 2$$

which is approximately $s+4.885$, and anyway, less than $s+5$, again exceedingly tractable.

Thus, the average "loss of precision" in evaluating $1/x$ in $[0, 1]$ is finite and does not exceed $2\log_2 e + 2$, consistent with our intuition. On the other hand, using the fact that given $\varepsilon > 0$, $\delta(\varepsilon, x) = \varepsilon x^2/(1+\varepsilon x)$, a simple calculation shows that the average "condition" ε/δ for the problem is infinite.

In these examples, we can also easily estimate the variance. Here it is useful to note that for square integrable functions f and f^* on $[0, 1]$ if $f^* \leqq f \leqq f^* + 1$ then $|f - \int f| < |f^* - \int f^*| + 1$. So, a crude estimate by Cauchy-Schwarz gives $\text{Var}(f) \leqq \text{Var}(f^*) + 2\sigma(f^*) + 1$ where

$$\text{Var}(f) = \int_0^1 \left(f(x) - \int_0^1 f(x)\, dx \right)^2 dx$$

is the variance of f, and $\sigma(f) = \sqrt{\text{Var}(f)}$ is the standard deviation.

Thus, in example 2, letting $\Pi_s^*(x) = s + 2l(x) + 1$, a straightforward calculation shows that

$$\text{Var}(\Pi_s) = \int_0^1 (\Pi_s(x) - \text{Av } \Pi_s)^2\, dx \leqq 4(\log_2 e)^2 + 4\log_2 e + 1.$$

Hence, $\sigma(\Pi_s) < 4$.

(3) Suppose that a positive integer s is given, and real numbers P and Q are chosen at random in the unit interval $[0, 1]$. We wish to compute P/Q to within relative accuracy $1/2^s$. What precision $\Pi_s(P, Q)$ is sufficient for P and Q? If $|P| > 1/2^l$ and

$|Q|>1/2^l$ then

$$\left|\frac{P^*/Q^*}{P/Q}-1\right|<\frac{1}{2^{s+1}}$$

as long as $|P-P^*|<1/2^A$ and $|Q-Q^*|<1/2^A$, where $A>s+l+3$.

Thus, $\Pi_s(P, Q)\leqq s+\max(m(P), m(Q))+3$. The average value of Π_s in the unit square is

$$\begin{aligned}\text{Av}\,\Pi_s &\leqq s+\left[\Sigma m\left(\left(1-\frac{1}{2^m}\right)^2-\left(1-\frac{1}{2^{m-1}}\right)\right)\right]+3\\ &\leqq \Sigma m\left(-\frac{1}{2^{m-1}}+\frac{1}{2^{m-2}}\right)+s+3\\ &=\Sigma m\left(\frac{1}{2^{m-1}}\right)+s+3=\left(\frac{1}{1-1/2}\right)^2+s+3=s+7.\end{aligned}$$

In the above examples we have basically used the following, which we also use later on.

LEMMA 1. *Let s, m, v, θ and A be positive integers and P, Q, P^*, Q^* be real numbers none of which is zero. Then*:

1) *If $|Q|\geqq 1/2^m$ and $|Q-Q^*|\leqq 1/2^\theta$ then $|1/Q-1/Q^*|<1/2^s$ as long as $\theta>s+2m$.*

2) *If $|Q|\geqq 1/2^m$, $|P|\leqq 2^V$, $|Q-Q^*|\leqq 1/2^\theta$ and $|P-P^*|\leqq 1/2^A$ then $|P/Q-P^*/Q^*|<1/2^s$ as long as $\theta>s+2m+v+1$ and $A>s+m+1$.*

3) *If $|P-P^*|<(1/2^{s+2})|P|$ and $|Q-Q^*|<(1/2^{s+2})|Q|$, then*

$$\left|\frac{P^*/Q^*}{P/Q}-1\right|<\frac{1}{2^s}.$$

Proof.

1)

$$\left|\frac{1}{Q}-\frac{1}{Q^*}\right|=\left|\frac{Q^*-Q}{Q\cdot Q^*}\right|\leqq\frac{1/2^t}{(1/2^m)(1/2^m-1/2^\theta)}=\frac{2^m}{2^{\theta-m}-1}$$

$$<\frac{1}{2^s}\quad\text{as long as } \theta>s+2m.$$

2)

$$\left|\frac{P}{Q}-\frac{P^*}{Q^*}\right|\leqq|P|\left|\frac{1}{Q}-\frac{1}{Q^*}\right|+\left|\frac{1}{Q^*}\right|\cdot\frac{1}{2^A}.$$

Now by 1)

$$\left|\frac{1}{Q}-\frac{1}{Q^*}\right|<\frac{1}{2^{s+v+1}}$$

and thus

$$\left|\frac{1}{Q^*}\right|<2^m+\frac{1}{2^{s+v+1}}.$$

Hence

$$\left|\frac{P}{Q}-\frac{P^*}{Q^*}\right|<\frac{1}{2^{s+1}}+\left(2^m+\frac{1}{2^{s+v+1}}\right)\frac{1}{2^A}<\frac{1}{2^s}.$$

3) $|1-P^*/P|<1/2^{s+2}$ and $|1-Q^*/Q|<1/2^{s+2}$. So,

$$1-\frac{1}{2^{s+2}}<\frac{P^*}{P}<1+\frac{1}{2^{s+2}}$$

and

$$1-\frac{1}{2^{s+2}}<\frac{Q^*}{Q}<1+\frac{1}{2^{s+2}}.$$

So

$$1-\frac{1}{2^{s+2}+1}<\frac{Q}{Q^*}<1+\frac{1}{2^{s+2}}.$$

Now multiply and subtract 1 to see that $|P^*Q/Q^*P-1|<1/2^s$.

3. Procedure for the analysis. We define the *precision associated with accuracy* $\delta>0$ to be $|\log_2 \delta|$ if $\delta \leqq 1$ and to be 0 otherwise.

Our procedure for the analysis of sufficient "input precision" (and hence a lower bound on the "cost" involved) on average, is as follows.

First, using Lemma 1.2 in § 2, we estimate the (output) precision $O_s(P, Q, x)$ for $P(x)$ and $Q(x)$ (depending on the values of $P(x)$ and $Q(x)$) necessary and sufficient for the value $f(x)=P(x)/Q(x)$ to be within accuracy $1/2^s \leqq 1$:

$$O_s(P, Q, x) \leqq s+2m_Q(x)+\log_2|P(x)|+1$$

where

$$m_Q(x)=\begin{cases}|\log_2|Q(x)|| & \text{if } |Q(x)| \leqq 1,\\ 0 & \text{otherwise}\end{cases}$$

is the *marking time* to determine that the denominator of f is not zero.

Next, we let $I_t(P, Q, x)$ be the (input) precision for x *and the coefficients of* P *and* Q, necessary and sufficient for the values $P(x)$ and $Q(x)$ to be within accuracy $1/2^t \leqq 1$. Even the most naive method of evaluating P and Q by multiplying out the monomials and adding them up will show that, for every positive integer t, $I_t(P, Q, x) \leqq t+H(d, T_f, \rho, r)$ where

$$H(d, T_f, \rho, r) \leqq \log_2 d+\log_2 T_f+d\log_2^+ r+\log_2^+ \rho+2.$$

Here $\log^+=\max\{\log, 0\}$, $d=\max\{\deg P, \deg Q\}$, T_f is the maximum of the number of nonzero terms of P and Q and is bounded by $\binom{n+1}{d}$, where n is the maximum number of variables of P and Q, ρ is an upper bound on the absolute value of the coefficients of P and Q, r is a positive real and $|x|<r$. Here $|x|$ is the Euclidean norm.

So, letting $\Pi_s(P, Q, x)$ be the *input precision* for x *and the coefficients of* P *and* Q necessary and sufficient to evaluate $f(x)$ to within accuracy $1/2^s$, we have $\Pi_s(P, Q, x) \leqq I_t(P, Q, x)$ where $t=O_s(P, Q, x)$. And so,

$$\begin{aligned}\Pi_s(P, Q, x) &\leqq O_s(P, Q, x)+H(d, T_f, \rho, r)\\ &\leqq s+2m_Q(x)+\log_2|P(x)|+1+H(d, T_f, \rho, r)\\ &\leqq s+2m_Q(x)+[\log_2 T_f+d\log_2^+ r+\log_2 \rho]+1+H(d, T_f, \rho, r)\\ &\leqq s+2m_Q(x)+2[\log_2 T_f+d\log_2^+ r+\log^+ \rho]+\log_2 d+3.\end{aligned}$$

Now, we also note that computing $P(x)/Q(x)$ to within accuracy $1/2^s$ using the naive method suggested above and fast multiplication requires no more than $2(d+1)T_f$ arithmetic operations times $O(\Pi_s(P, Q, x) \log \Pi_s(P, Q, x) \log\log \Pi_s(P, Q, x))$ bit operations.

Thus, in order to show that tractability of the average input precision necessary and sufficient to evaluate f to within accuracy $1/2^s$ and hence the tractability of evaluating f on average, our main problem, and the focus of this paper, is to estimate and show the tractability of the average marking time to decide that a polynomial is not zero. To do this we first show that the volume of the set of points on which a polynomial has values near zero is "small".

4. The volume estimate. Given a polynomial $Q: \mathbf{R}^n \to \mathbf{R}$ of degree $d \geqq 1$, express Q as

$$Q(X) = \sum_I a_I X_1^{i_1} \cdots X_n^{i_n} + a_0$$

where

$$I = \{i_1, \cdots, i_n\}, \qquad 0 \leqq i_1, \cdots, i_n \leqq d$$

and

$$|I| = \sum_{j=1}^{n} i_j \leqq d.$$

Let $b_I = i_1! \cdots i_n! a_I$ (where $0! = 1$) and $K_Q = 1/\max_{|I|=d} |b_I|$.

Let r be a positive real number and let $B_r = \{x \in R^n \,|\, |x| \leqq r\}$ be the ball of radius r about the origin in R^n. Let A_{n-1} be the $(n-1)$-dimensional surface area of ∂B_1, the boundary of B_1. Let V_n be the volume of B_1. So, the surface area of ∂B_r is $A_{n-1} r^{n-1}$ and the volume of B_r is $V_n r^n$.

MAIN THEOREM 1. *Let $Q: R^n \to R$ be a polynomial of degree $d \geqq 1$. For any real numbers $r > 0$ and $\varepsilon > 0$,*

$$\text{Vol}\{x \in B_r \,|\, |Q(x)| < \varepsilon\} < 2\varepsilon^{1/d} K_Q^{1/d} \left[V_{n-1} + \left(\frac{d(d+1)}{2} - 1 \right) \frac{A_{n-1}}{2} \right] r^{n-1}.$$

Proof outline. The proof is by induction on the degree d. The linear case ($d = 1$) is straightforward. The inductive step uses the co-area formula (Federer [2]) to show that the Vol $\{x \in B_r \,|\, |Q(x)| < \varepsilon\}$ can be computed by first computing the volumes of the fibers $Q^{-1}(w) \cap B_r$ for ($|w| < \varepsilon$), and then integrating over the fibers. Then the methods of integral geometry (Santalo [11]) are used to compute the volumes of the fibers. The "capsule" $\{x \in B_r \,|\, |Q(x)| < \varepsilon\}$ is partitioned in such a way as to make use of the inductive hypothesis.

Proof. Since $\{x \in B_r \,|\, |Q(x)| < \varepsilon\} = \{x \in B_r \,|\, |K_Q Q(x)| < \varepsilon K_Q\}$, and noting that $K_{K_Q Q} = 1$, it suffices to prove the theorem when $K_Q = 1$. We prove this theorem by induction on the degree d. If the degree of Q is 1, $Q(X) = \sum_{i=1}^n a_i X_i + a_0$ and $1 = 1/\max_{i>0} |a_i|$. The gradient $\nabla Q = (a_1, \cdots, a_n)$ and $1/|\nabla Q| \leqq 1$. Vol $\{x \in B_r \,|\, |Q(x)| < \varepsilon\}$ is therefore less than the volume of the ball of radius r in the $n-1$ plane orthogonal to $(a_1, \cdots, a_n)$ with $(-a_0/|\nabla Q|^2)$ $(a_1, \cdots, a_n)$ as center, times $2\varepsilon/|\nabla Q|$. So, Vol $\{x \in B_r \,|\, |Q(x)| < \varepsilon\} < 2\varepsilon V_{n-1} r^{n-1}$. Now we proceed by induction. Assuming the inequality proven for $d \geqq 1$ we attempt to prove it for Q of degree $d+1$. Choose x_i such that $K_{\partial Q/\partial x_i} = K_Q = 1$.

Let $S_1 = \{x \in B_r \mid |Q(x)| < \varepsilon\}$ and $S_2 = \{x \in B_r \mid |\partial Q/\partial x_i| < \varepsilon^{d/(d+1)}\}$. $\text{Vol}(S_1) \leqq \text{Vol}(S_1 - S_2) + \text{Vol}(S_2)$. By the inductive hypothesis

$$\text{Vol}(S_2) \leqq 2(\varepsilon^{d/(d+1)})^{1/d}\left[V_{n-1} + \left(\frac{d(d+1)}{2} - 1\right)\frac{A_{n-1}}{2}\right] r^{n-1}.$$

On $S_1 - S_2$, $1/|\nabla Q| \leqq 1/|\partial Q/\partial x_i| < 1/\varepsilon^{d/(d+1)}$. So, by the co-area formula (see Federer [2, Thm. 3.2.12, p. 249]),

$$\begin{aligned}\text{Vol}(S_1 - S_2) &= \int_{-\varepsilon}^{\varepsilon}\left[\int_{Q^{-1}(w)\cap(S_1-S_2)} \frac{1}{|\nabla Q|}\right] dw \\ &\leqq 2\varepsilon \frac{1}{\varepsilon^{d/(d+1)}} \max\left(\text{vol}\left(Q^{-1}(w) \cap (S_1 - S_2)\right)\right) \\ &= 2\varepsilon^{1/(d+1)}(\max \text{vol}\left(Q^{-1}(w) \cap (S_1 - S_2)\right)\end{aligned}$$

where vol $Q^{-1}(w) \cap (S_1 - S_2)$ and the first integral are with respect to the induced $(n-1)$-dimensional volume on the hypersurface $Q^{-1}(w)$. Thus we will be done as soon as we show that $\text{Vol}(Q^{-1}(w)) \cap (S_1 - S_2) \leqq (d+1)(A_{n-1}/2) r^{n-1}$. This follows from the following proposition which is essentially contained in Smale [13]; the general case was shown to us in a personal communication from Smale.

PROPOSITION 1 (Smale). *Let $Q: \mathbf{R}^n \to \mathbf{R}$ be a polynomial of degree $d > 1$. For any real number $r > 0$*

$$\text{Vol}\left(Q^{-1}(w) \cap B_r\right) \leqq d\frac{A_{n-1}}{2} r^{n-1}.$$

For the sake of completeness, we include a sketch of this proposition. The typical line L_1 in R^n can meet $Q^{-1}(w) \cap B_r$ in at most d points. Of the ones that do, the typical one intersects $\partial B_r = S_r^{n-1}$ in two points. Using results from Santalo [11, 14.70, p. 245] we have

$$\begin{aligned}\frac{A_n}{A_1}\text{Vol}\left(Q^{-1}(w) \cap B_r\right) &= \int_{Q^{-1}(w)\cap B_r \cap L_1 \neq \varnothing} \#\left(Q^{-1}(w) \cap B_r \cap L_1\right) dL_1 \\ &\leqq \frac{d}{2}\int_{Q^{-1}(w)\cap B_r \cap L_1 \neq \varnothing} 2\, dL \\ &\leqq \frac{d}{2}\int_{S_r^{n-1}\cap L_1 \neq \varnothing} 2\, dL_1 \\ &= \frac{d}{2}\frac{A_n}{A_1}\text{Vol}\left(S_r^{n-1}\right) \\ &= \frac{d}{2}\frac{A_n}{A_1} A_{n-1} r^{n-1}.\end{aligned}$$

Q.E.D

MAIN THEOREM 2. *Let $Q: \mathbf{R}^n \to \mathbf{R}$ be a polynomial of degree $d \geqq 0$. For any real numbers $r > 0$ and $\varepsilon > 0$*

$$\frac{\text{Vol}\{x \in B_r \mid |Q(x)| < \varepsilon\}}{\text{Vol}\{B_r\}} < \frac{C_Q}{r}\varepsilon^{1/d}$$

where $C_Q = K_Q^{1/d} n((d(d+1))/2)$.

Proof. Recall (e.g. Santalo [11, p. 976] $V_0 = 1$, $V_2 = 2$, $V_2 = \pi$ and generally $V_n = 2\pi^{n/2}/n\Gamma(n/2)$ where Γ is the gamma function. Also, $A_0 = 2$, $A_1 = 2\pi$, $A_2 = 4\pi$ and generally $A_n = 2\pi^{(n+1)/2}/\Gamma((n+1)/2)$. Thus for $n = 1$,

$$\text{Vol}\,(x \in B_r \mid |Q(x)| < 2\,\varepsilon^{1/d} K_Q^{1/d} \frac{d(d+1)}{2}.$$

So, if we normalize, we have for $n = 1$,

$$\frac{\text{Vol}\,\{x \in B_r \mid |Q(x)| < \varepsilon\}}{\text{Vol}\,\{B_r\}} < \frac{\varepsilon^{1/d}}{r} K_Q^{1/d} \frac{d(d+1)}{2},$$

and for $n > 1$,

$$\frac{\text{Vol}\,\{x \in B_r \mid |Q(x)| < \varepsilon\}}{\text{Vol}\,\{B_r\}} < \frac{2\varepsilon^{1/d}}{r} K_Q^{1/d} \left[\frac{V_{n-1}}{V_n} + \left(\frac{d(d+1)}{2} - 1 \right) \frac{A_{n-1}}{2V_n} \right].$$

Now $A_{n-1}/V_n = n$ and

$$\frac{V_{n-1}}{V_n} = \left(\frac{n}{n-1} \right) \frac{\Gamma(n/2)}{\Gamma((n-1)/2)\pi^{1/2}} < \frac{n}{2}.$$ Q.E.D.

5. Average marking time and average input precision. We continue with the notation and setting introduced in §§ 3 and 4. We now wish to estimate the average value of $m_Q(x)$, the "marking time" to determine that $Q(x)$ is not zero for $x \in B_r$.

Recall,

$$m_Q(x) = \begin{cases} |\log_2 |Q(x)|| & \text{if } |Q(x)| \leqq 1, \\ 0 & \text{otherwise.} \end{cases}$$

From the normalized volume estimate, and assuming normalized Lebesgue measure on B_r, we see that for $0 \leqq \varepsilon \leqq 1$, $m_Q(x) \leqq |\log_2 \varepsilon|$ with probability at least $1 - C_Q \varepsilon^{1/d}/r$. This enables us to estimate the average value of $m_Q(x)$.

THEOREM 3. *Let $Q: \mathbf{R}^n \to \mathbf{R}$ be a polynomial of degree $d \geqq 1$ and let r be a positive real number. Then* Av m_Q, *the average of $m_Q(x)$ in B_r with respect to normalized Lebesgue measure, satisfies*

$$\underset{|x|<r}{\text{Av}}\, m_Q \leqq d \left(\log_2^+ \left(\frac{C_Q}{r} \right) + \log_2 e \right)$$

where $C_Q = K_Q^{1/d} nd(d+1)/2$ and $\log^+ = \max(\log, 0)$.

Also, $\sigma(m_Q)$ the standard deviation of $m_Q(x)$ in B_r, satisfies

$$\sigma(m_Q) \leqq d \log_2 e \left(\left(\ln^+ \frac{C_Q}{r} \right)^2 + 2 \ln^+ \frac{C_Q}{r} + 2 \right)^{1/2}.$$

Proof. We first recall the following (from Shub-Smale [12, Part II]):

DEFINITION. Let (X, μ) be a probability space with no atoms. Let $m: X \to \mathbf{R}_+$ be a real valued nonnegative measurable function and let $f: (0, 1) \to \mathbf{R}$ be decreasing and Riemann integrable. We say that $m(x) \leqq f(\mu)$ with probability $1 - \mu$ if $\mu\{x \mid m(x) \leqq f(y)\} \geqq 1 - y$ for all $0 < y < 1$.

PROPOSITION 2. *Suppose as above that $m(x) < f(\mu)$ with probability $1 - \mu$. Then*

(1)

$$E(m) = \int_X m(x)\mu(dx) \leqq \int_0^1 f(\mu)\, d\mu;$$

(2)

$$\text{Var}(m) = \int_X (m(x) - E(m))^2 \mu(dx) \leqq \int_0^1 f^2(\mu)\, d\mu - (E(m))^2.$$

Returning to our proof, we let $e(\mu) = (\min(1, r\mu/C_Q))^d$ for $0 \leqq \mu \leqq 1$. Thus, letting $f(\mu) = |\log_2 e(\mu)|$, we get $m_Q(x) \leqq f(\mu)$ with probability $1-\mu$. And so, by above $\int_{B_r} m_Q(x)\, dx \leqq \int_0^1 f(\mu)\, d\mu$.

Now, $\int_0^1 f(\mu)\, d\mu = -d \int_0^\gamma \log_2(\gamma\mu/C_Q)\, d\mu = \log_2 e(d\gamma + d\gamma \ln(C_Q/\gamma r))$, where $\gamma = \min(C_Q/r, 1)$. Thus, if $C_Q/r \geqq 1$, then $\gamma = 1$ and we have

$$\int f(\mu)\, d\mu = d \log_2 e\left(\ln \frac{C_Q}{r} + 1\right).$$

And, if $C_Q/r < 1$ then $\gamma = C_Q/r$ and $\int f(\mu)\, d\mu < d \log_2 e$. So, $\int m_Q(x)\, dx \leqq \int f(\mu)\, d\mu \leqq d \log_2 e(\ln^+(C_Q/r) + 1) = d(\log_2^+(C_Q/r) + \log_2 e)$.

The second estimate follows in a similar fashion, now using part 2 of Proposition 2 and the fact that $\sigma(m_Q) = \sqrt{\text{Var}(m_Q)}$. Q.E.D.

And thus we have also shown:

COROLLARY. *Let $Q: \mathbf{R}^n \to \mathbf{R}$ be a polynomial of degree d. Then (with respect to normalized Lebesgue measure)*

$$\int_{B_1 \cap Q^{-1}[-1,1]} |\ln|Q(x)|| \leqq \begin{cases} 2d \ln(d) + d \ln(n) + d + \ln(K_Q) & \text{if } C_Q \geqq 1, \\ C_Q d & \text{if } C_Q < 1. \end{cases}$$

By § 3 and Theorem 3 we get:

THEOREM 4. *Let $f(X) = P(X)/Q(X)$ be a rational function $f: \mathbf{R}^n \to \mathbf{R}$ of degree $d \geqq 1$. Suppose that the maximum absolute value of the coefficients of P and Q is ρ and that a real number $r \geqq 0$ and an integer $s > 0$ are given. Then $\text{Av}_{|x|<r} \Pi_S(P, Q, x)$, the average input precision for x and the coefficients of P and Q necessary and sufficient to evaluate $f(x)$ to within accuracy $1/2^s$ on B_r with normalized Lebesgue measure, is finite and satisfies*

$$\operatorname*{Av}_{|x|<r} \Pi_s(P, Q, x) \leqq s + 2 \operatorname*{Av}_{|x|<r} m_Q + \log_2 \max_{|x|<r} |P(x)| + 1 + H(d, T_f, \rho, r)$$

$$\leqq s + 2d\left(\log_2^+ \frac{C_Q}{r} + \log_2 e\right) + 2[\log_2 T_f + d \log^+ r + \log_2^+ \rho] + \log_2 d + 3$$

where $C_Q = K_Q^{1/d} nd(d+1)/2$ and T_f, the maximum number of nonzero terms of P and of Q, is bounded by $\binom{n+1}{d}$.

So, the average loss of precision in evaluating $P(x)/Q(x)$ in B_r is loglinear and crudely bounded by

$$2d(2 \log_2 d + 2 \log_2(n+1) + \log_2^+ r + \log_2 e) + \log_2 d + 2(\log_2^+ \rho + \log_2 K_Q^+) + 3.$$

We may average as well over the polynomials themselves. Let $P(n, d)$ denote the vector space of real polynomials $Q: \mathbf{R}^n \to \mathbf{R}$ of degree d, and F a vector subspace determined by allowing a fixed subset of the coefficients to be nonzero, with at least one of these the coefficient of a term of degree d. Let k be the number of nonzero coefficients of degree d and m the dimension of f. Let $C(\rho)$ be the cube of side 2ρ in F i.e., $C(\rho) = \{Q \in F |$ the maximum absolute value of a coefficient of Q is less than or equal to $\rho\}$. Thus the volume of $C(\rho) = 2^m \rho^m$. We normalize this volume to one, fix $r > 0$ and average $\log_2^+(C_Q/r)$ over $C(\rho)$.

LEMMA 2. $\log_2^+ (C_Q/r) < (1/d) \log_2^+ (1/r\rho\mu^{1/k}) + \log_2 n + \log_2 (d(d+1)/2)$ *with probability* $1-\mu$.

Proof. $K_Q \leqq 1/\max_{I=d} |a_I| \leqq 1/\rho\mu^{1/k}$ with probability at least $1-\mu$ since the normalized volume of

$$\{Q \in C(\rho) | \max_{|I|=d} |a_I| < \rho\mu^{1/k}\}$$

is equal to

$$\frac{2^m \rho^{m-k} (\rho\mu^{1/k})^k}{2^m \rho^m} = \mu.$$

So,

$$\log_2^+ \left(\frac{C_Q}{r}\right) = \log_2^+ \left[\frac{K_Q^{1/d} nd(d+1)/2}{r}\right] \leqq \frac{1}{d} \log_2^+ \left(\frac{K_Q}{r}\right) + \log_2 n + \log_2 \frac{d(d+1)}{2}$$

$$\leqq \frac{1}{d} \log_2^+ \frac{1}{r\rho\mu^{1/k}} + \log_2 n + \log_2 \frac{d(d+1)}{2}$$

with probability $1-\mu$. Q.E.D.

Now let

$$\psi_k(\rho, r) = \begin{cases} \log_2 (e)\left(\frac{1}{k} - \ln \rho r\right) & \text{if } \rho r \leqq 1, \\ \log_2 (e) \frac{1}{k} \left(\frac{1}{\rho r}\right)^k & \text{if } \rho r \geqq 1. \end{cases}$$

Now Proposition 2 gives

LEMMA 3.

$$\int_{C(\rho)} \log_2^+ \frac{C_Q}{r} \leqq \log_2 n + \log_2 \frac{d(d+1)}{2} + \frac{1}{d} \psi_k(\rho, r).$$

Proof.

$$\int \log_2^+ \left(\frac{1}{r\rho\mu^{1/k}}\right) = \log_2 (e) \int_0^\gamma \ln \left(\frac{1}{r\rho\mu^{1/k}}\right) d\mu$$

where $\gamma = \min (1, (1/\rho r)^k)$ and the two cases give the two integrals.

THEOREM 5. *Let F be as above and r, ρ real numbers bigger than zero. Then the average marking time for a point* (Q, x) *in* $C(\rho) x B_r$ *is less than or equal to* $d \log_2 n + d \log_2 (d(d+1)/2) + \psi_k(\rho, r) + d \log_2 e$.

Proof. The double integral $\iint_{C(\rho) x B_r} M$ is just the iterated single intervals by Fubini's theorem and thus Theorem 3 and Lemma 3 finish the argument.

6. The relative case. We now consider the relative case. The *relative condition* of evaluating a function f at x is the ratio ε/δ where $|\Delta x|/|x| < \delta$ implies $|\Delta f|/|f(x)| < \varepsilon$. Its logarithm$^+$ can be considered a measure of the *loss of significance* for x in evaluating $f(x)$ since it represents the loss of significant digits when input x and output $f(x)$ are given in scientific or floating point notation. Rewriting the ratio $|\Delta f|/|f|/|\Delta x|/|x|$ we get $|\Delta f|/|\Delta x| \cdot |x|/|f(x)|$ which is approximated by $|Df| \cdot |x|/|f(x)|$, the *infinitesimal condition* of the problem. (Note that by the mean value theorem, we have $|\Delta f|/|x| \leqq \sup |Df(x^*)|$ over all x^* on the line segment between x and $x+\Delta x$.) We call the logarithm$^+$ of the infinitesimal condition, the *infinitesimal loss of significance.*

As an example, we consider the problem of evaluating $f(x) = 1/(1-x)$ in the unit interval. Here the infinitesimal condition is $x/(1-x)$ and the infinitesimal loss of significance is $\log^+(x/(1-x))$. So, analogous to the absolute case, we have a problem were the average loss of significance is small ($\int_0^1 \log_2^+(x/(1-x))\,dx = 1$), although the average condition is infinite.

We now wish to estimate the average loss of significance for x in evaluating a general rational function $f = P/Q$. Suppose we know that $|\Delta x| < \delta'$ implies $|\Delta f|/|f| < \varepsilon$. Then we have, $|\Delta x|/|x| < \delta$ implies $|\Delta f|/|f| < \varepsilon$ where $\delta = \delta'/|x|$. So,

$$\log^+\left(\frac{\varepsilon}{\delta}\right) = \log^+\left(\frac{\varepsilon}{\delta'/|x|}\right) \leqq \log^+\left(\frac{\varepsilon}{\delta'}\right) + \log^+|x|.$$

By Lemma 1 and § 3 we have,

$$\log^+\left(\frac{\varepsilon}{\delta'}\right) \leqq \max(m_P(x), m_Q(x)) + 2 + H(d, T_f, \rho, r).$$

So, Theorem 6 follows.

THEOREM 6. *Let $f = P/Q\colon \mathbf{R}^n \to \mathbf{R}$ be a rational function of degree $d \geqq 1$ and let $r > 0$ be a real number. Then, the average loss of significance for x in evaluating $f(x)$ in B_r is bounded by*

$$\operatorname*{Av}_{|x|<r} \log^+|x| + \operatorname*{Av}_{|x|<r} \max(m_P(x), m_Q(x)) + H(d, T_f, \rho, r) + 2$$

$$\leqq \log_2 r - \frac{1}{n} + d\log_2^+\left[\frac{C_P + C_Q}{r}\right] + d\log_2 e + H(d, T_f, \rho, r) + 2.$$

Proof. To integrate $\int_{B_r} \log^+|x|/\operatorname{vol} B_r$ use polar coordinates and integration by parts.

Also, by Theorem 2 we have,

$$\frac{\operatorname{Vol}\{x \in B_r | \min(|P(x)|, |Q(x)|) < \varepsilon\}}{\operatorname{Vol} B_r} \leqq \frac{(C_P + C_Q)\varepsilon^{1/d}}{r}.$$

Now proceed as in Theorem 3.

7. Loss of significance in solving linear equations. An immediate consequence of Theorem 3 is a tractable upper bound for the average loss of significance in solving systems of linear equations. This was a question of great interest to von Neumann [15]. Here the problem is: *Given* $A \in \mathbf{R}^{n^2}$, a real $n \times n$ invertible matrix and a vector $b \in \mathbf{R}^n$, *solve* $Ax = b$ for x. What is the loss of significance?

Suppose an error Δb in input causes an error Δx in solution, i.e., $A(x + \Delta x) = b + \Delta b$. By linearity, we have $A(\Delta x) = \Delta b$ and by invertibility, we have $A^{-1}(\Delta b) = \Delta x$. So, the *relative condition* of the problem is

$$\frac{\dfrac{|\Delta x|}{|x|}}{\dfrac{|\Delta b|}{|b|}} = \frac{|\Delta x||b|}{|\Delta b||x|} = \frac{|A^{-1}(\Delta b)|}{|(\Delta b)|}\frac{|Ax|}{|x|}$$

where $||$ is some standard norm. We note that the right-hand side is bounded above by $K_A = \|A^{-1}\|\,\|A\|$, where $\|A\| = \max|Ax|/|x| = \max_{|x|=1}|Ax|$ is the *operator norm* of A. Numerical analysts, see e.g. Moler [9], call K_A the *condition number* of the matrix A, and it can be considered the worst case measure of the relative condition for the

problem of solving $Ax = b$ (with fixed A, varying b). Analogously, we consider $\log K_A$ which is a measure of the worst case *loss of significance* of the system.

THEOREM 7. *With respect to the Euclidean norm* $||$ *in* $\mathbf{R}^n$ *and* $\mathbf{R}^{n^2}$ *and normalized Lebesgue measure,*

$$\operatorname*{Av}_{A\in B_1\subseteq \mathbf{R}^{n^2}} \ln K_A \leqq \ln n - \left(\frac{n-1}{2}\right)\ln(n-1) + n + 4n\ln n.$$

Proof. Let $A = (a_{ij})$ and $A^{-1} = (b_{ij}/\operatorname{Det} A)$ where b_{ij} is the ijth cofactor of A. Suppose $A \in B_1$. Then, $\|A\| \leqq 1$: For suppose $|x| = 1$. Let a_i be the vector $(a_{i1}, \cdots, a_{in})$. Then,

$$|Ax| = \left(\sum_{i=1}^{n} (a_i \cdot x)^2\right)^{1/2} \leqq \left(\sum_{i=1}^{n} |a_i|^2|x|^2\right)^{1/2} = \left(\sum_{i=1}^{n} |a_i|^2\right)^{1/2} = \left(\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2\right)^{1/2} \leqq 1$$

since $A \in B_1 \subset \mathbf{R}^{n^2}$.

To estimate $\|A^{-1}\|$ we estimate $|b_{ij}|$: Suppose $b_{ij} = \operatorname{Det}(c_{lk})$ where c_{lk} are entries from A. Let $C_k = (\sum_{l=1}^{n-1} c_{lk}^2)^{1/2}$ be the length of the kth column vector. Then $|b_{ij}| \leqq \prod_{k=1}^{n-1} C_k$. Since $A \in B_1$, $\sum_{k=1}^{n-1} C_k^2 \leqq 1$. And so, $\prod_{k=1}^{n-1} C_k \leqq (1/\sqrt{n-1})^{n-1}$. Thus, $|b_{ij}| \leqq (1/\sqrt{n-1})^{n-1}$. So we have,

$$K_A = \|A^{-1}\|\,\|A\| \leqq \|A^{-1}\| \leqq \frac{n}{\operatorname{Det} A}\max|b_{ij}|$$

$$\leqq \frac{n}{\operatorname{Det} A}\frac{1}{(n-1)^{(n-1)/2}}.$$

Therefore,

$$\ln K_A \leqq \ln n - \left(\frac{n-1}{2}\right)\ln(n-1) + |\ln|\operatorname{Det} A||.$$

Now, Det A is a polynomial in n^2 variables of degree n and $K_{\operatorname{Det} A} = 1$. By the corollary to Theorem 3,

$$\operatorname*{Av}_{A\in B_1} \ln K_A \leqq \ln n - \left(\frac{n-1}{2}\right)\ln(n-1) + 2n\ln n + n\ln n^2 + n$$

$$= \ln n - \left(\frac{n-1}{2}\right)\ln(n-1) + n + 4n\ln n.$$

This estimate follows from very general considerations, and so one would expect better results for this particular problem. Indeed, using methods different from ours, Eric Kostlan [8] has shown that although the average condition number (with respect to the standard Gaussian measure or the space of $n \times n$ complex matrices) is infinite, the average loss of significance is less than $\frac{5}{2}\ln n$. Adrian Ocneanu [10] has calculated sharp upper and lower bounds, $(3+\varepsilon)\ln n$ and $(\frac{2}{3}-\varepsilon)\ln n$ respectively, where ε can be made as small as desired by setting a lower bound on n.

Acknowledgments. We would like to thank Steve Smale and Henryk Wozniakowski for useful conversations.

REFERENCES

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

[2] H. FEDERER, *Geometric Measure Theory*, Springer, Berlin, 1969.
[3] R. HARDT, *Some analytic bounds for subanalytic sets*, in Differential Geometric Control Theory, 1981, pp. 259-267.
[4] P. HENRICI, *Essentials of Numerical Analysis*, John Wiley, New York, 1982.
[5] M. H. KIM, Thesis, in preparation.
[6] D. KNUTH, *The Art of Computer Programming, Vol.* 2, Seminumerical Algorithms, 2nd ed., Addison-Wesley, Reading, MA, 1981.
[7] K. KO AND H. FRIEDMAN, *Computational complexity of real functions*, Theoret. Comput. Sci., 20 (1982), pp. 323-352.
[8] E. KOSTLAN, *Estimates in numerical linear algebra*, preprint.
[9] C. B. MOLER, *Three research problems in numerical linear algebra*, in Numerical Analysis, Proc. Symposia in Applied Mathematics, Vol. XXII, G. H. Golub and J. Oliger, eds., American Mathematical Society, Providence, RI, 1978, pp. 1-18.
[10] A. OCNEANU, *On the loss of precision in solving large linear systems*, preprint.
[11] L. SANTALO, *Integral Geometry and Geometric Probability*, Addison-Wesley, Reading, MA.
[12] M. SHUB AND S. SMALE, *Computational complexity: on the geometry of polynomials and a theory of cost*, Part I, Ann. Scientifique de L'Ecole Normale Superieure, 4, 18 (1985); Part II, this Journal, 15 (1986), pp. 145-161.
[13] S. SMALE, *The fundamental theorem of algebra and complexity theory*, Bull. Amer. Math. Soc. (New series) 4 (1981), pp. 1-36.
[14] ———, *On the efficiency of algorithms of analysis*, preprint.
[15] J. VON NEUMANN, *Collected Works, Vol.* 5: *Design of Computers, Theory of Automata and Numerical Analysis*, A. H. Taub, ed., Pergamon, New York, 1963.
[16] N. WEISS, personal communication.
[17] J. H. WILKINSON, *Rounding Errors in Algebraic Processes*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
005

OPTIMAL APPROXIMATIONS AND POLYNOMIALLY LEVELABLE SETS*

PEKKA ORPONEN†, DAVID A. RUSSO‡ AND UWE SCHÖNING§

Abstract. A set A not in P is *polynomially levelable* if any algorithm for A has speedup to a polynomial infinitely often in A: precisely, if given any algorithm M for A and polynomial p, it is possible to find another algorithm M' for A and polynomial p', such that M' runs in time $p'(|x|)$ on infinitely many inputs x in A, on which the running time of M exceeds $p(|x|)$. Intuitively, this condition states that among polynomial time computable approximations to A there is no optimal one, or one giving correct answers on a maximally large subset of A. It appears that most naturally occurring intractable sets are polynomially levelable. We prove this for sets not in P that are either "paddable," "self-reducible," or complete for a deterministic time class. We also discuss levelability preserving reductions, and give a simple reducibility characterization of nonlevelable sets.

Key words. computational complexity, approximations, paddability, self-reducibility, completeness, polynomial reductions

1. Introduction.

We define a set of strings A to be *polynomially levelable*, or P-*levelable* for short, if for any algorithm M recognizing A and polynomial p there exist another algorithm M' for A and polynomial p', such that M' runs in time $p'(|x|)$ on infinitely many strings $x \in A$ on which the running time of M exceeds $p(|x|)$. (Use of the term "levelable" was suggested by K. Ko [9] in analogy with the notions discussed in [4], [19].) P-levelability can be thought of as a kind of pseudo-speedup property, but it is more naturally viewed in the context of approximations.

Consider the following simple case of approximation: an *approximation algorithm* for a set A is a polynomial time algorithm M that on each input x outputs either "1" (accept), "0" (reject) or "?" (do not know), in such a manner that $M(x)=1$ implies $x \in A$ and $M(x)=0$ implies $x \notin A$. (This notion of approximation was essentially introduced by Meyer and Paterson in [14]; see also [11].) An approximation algorithm is *optimal* if no other polynomial time algorithm correctly decides infinitely many more inputs, i.e., outputs infinitely many more correct 1's or 0's. It is fairly easy to show that a recursive set A has an optimal approximation algorithm in this sense if and only if neither A nor $\bar{A}$ is P-levelable.

Meyer and Paterson [14] show that many natural intractable sets do not have "good" approximation algorithms. We prove that often they do not have even optimal ones. In §§ 3 and 4 of this paper we prove the P-levelability of all sets not in P that are either "honestly paddable" [3], [21], "honestly k-creative" [21], or in a restricted sense "self-reducible" [10], [14], [18]. These results cover all known NP- and PSPACE-complete sets [3], [21], and for the common notion of "invertible" paddability we prove even much stronger results. Further, we derive as a corollary to a theorem by L. Berman [2] the P-levelability of intractable sets that are complete for deterministic time classes, for example, EXPTIME.

In § 5 we consider the problem of proving the P-levelability of *all* NP- and PSPACE-complete sets and introduce a levelability preserving subclass of the poly-

* Received by the editors July 10, 1984, and in final form December 26, 1984. This work was supported in part by the Emil Aaltonen Foundation, the Academy of Finland, the Deutsche Forschungsgemeinschaft, and the National Science Foundation under grant DCR83-12472.

† Department of Computer Science, University of Helsinki, SF-00250 Helsinki 25, Finland. This work was carried out while this author was visiting the Department of Mathematics, University of California, Santa Barbara, California 93106.

‡ Department of Mathematics, University of California, Santa Barbara, California 93106.

§ Institut für Informatik, Universität Stuttgart, D-7000 Stuttgart 1, West Germany.

nomial time many-one reductions. Another related class of reductions gives us a natural reducibility characterization of the non-P-levelable sets, very similar to that given for the "bi-immune" sets in [1].

The notion of P-levelability originally evolved from our studies of complexity cores of intractable sets [5], [15], [16], [17]. A *complexity core* for a set A is a set C such that given any algorithm M for A and polynomial p, the run time of M exceeds $p(|x|)$ on almost all x in C. In a sense, the inputs in C are "uniformly hard" for any algorithm deciding A. This notion was introduced by Lynch [12], who proved that any recursive set A not in P has an infinite complexity core. Moreover, it can be shown that any recursive set A not in P contains an infinite complexity core [16].

If A were to contain a core that is *maximal* up to finite variations, we could interpret the core as consisting of all the hard (yes-)instances of the decision problem for A. In [5] it was pointed out that this happens if and only if A is *not* P-levelable. In [16] it was further shown that for a recursive set A, non-P-levelability is equivalent to the condition that A be the disjoint union of a set in P and a P-immune set. (A set is P-immune if it has no infinite P-subsets [6].) The non-P-levelable sets were in this context called *almost* P-*immune*. By the results reported in this paper, we now know that many natural intractable sets are not almost P-immune, and do not contain maximal complexity cores. Instead their complexity core structure is very complicated. (Surprisingly, all recursive P-levelable sets have the same complexity core structure, as lattices ordered by inclusion modulo the finite sets [15]. Moreover, it has been shown that their P-subset structures are also isomorphic [17].)

2. Preliminaries. We consider sets of strings over the alphabet $\Sigma = \{0, 1\}$. The length of a string $x \in \Sigma^*$ is denoted $|x|$; similarly, the cardinality of a set $A \subseteq \Sigma^*$ is denoted $|A|$. For a set $A \subseteq \Sigma^*$ and integer $n \geqq 0$, $A^{(n)}$ denotes the finite set $\{x \in A \mid |x| \leqq n\}$. A set A is (polynomially) *sparse* if for some polynomial p, $|A^{(n)}| \leqq p(n)$ for every $n \geqq 0$. Set A is *eventually contained* in set B if the difference $A - B$ is finite.

As a model of computation we use deterministic Turing machines; occasional reference is also made to nondeterministic and oracle machines. These machines and their time complexity measures are defined as in, for example, [7]. The set of strings accepted by machine M with oracle A is denoted $L(M, A)$, and the set $L(M, \varnothing)$ is denoted briefly $L(M)$. The complexity classes P and NP have their standard definitions, and the class EXPTIME is defined as:

$$\text{EXPTIME} = \{L(M) \mid M \text{ runs in time } 2^{cn}, \text{ for some constant } c\}.$$

A set that has no infinite subsets in P is called P-*immune*.

Set A is (polynomial time many-one) *reducible* to set B, denoted $A \leqq_m^{\mathrm{P}} B$, if there is a polynomial time computable function $f: \Sigma^* \to \Sigma^*$ such that for all $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$. Set A is $\mathscr{C}$-*hard*, for a class of sets $\mathscr{C}$, if every $B \in \mathscr{C}$ is reducible to A, and $\mathscr{C}$-*complete*, if in addition $A \in \mathscr{C}$. Sets A and B are *polynomially isomorphic* if A is polynomial time reducible to B via a one-one, onto function f whose inverse f^{-1} can also be computed in polynomial time. A function $f: \Sigma^* \to \Sigma^*$ is (polynomially) *honest* if there is a polynomial p such that $p(|f(x)|) \geqq |x|$, for all $x \in \Sigma^*$. Polynomial isomorphisms are necessarily honest.

3. *P*-levelability and paddability. In this section we introduce the notion of P-levelability, and prove our basic result: that "honestly paddable" sets not in P are P-levelable. We also present a stronger version of this theorem for "invertibly" paddable sets, and apply the proof technique to establish the P-levelability of Young's [21] "honestly k-creative" sets.

DEFINITION 3.1. Given a deterministic Turing machine M and a function f on the natural numbers, denote by $E(M,f)$ the set $\{x \in A | M \text{ accepts } x \text{ in time } f(|x|)\}$. A set $A \subseteq \Sigma^*$ is P-*levelable* if, given any recognizer M for A and polynomial p, it is possible to find another recognizer M, for A and polynomial p', such that the difference $E(M', p') - E(M, p)$ is infinite.

A simple, but useful, characterization of the P-levelable sets is obtained by observing that for a recursive set A, the sets $E(M, p)$ for machines M recognizing A and polynomials p correspond exactly to the polynomial time recognizable subsets of A. Hence it can be seen that a recursive set A is *not* P-levelable if and only if it contains a P-subset E that is *maximal*, in the sense that each $E' \subseteq A$, $E' \in P$ is eventually contained in E.

A set $A \subseteq \Sigma^*$ is *paddable* [3], [21] if there exists a polynomial time computable function pad: $\Sigma^* \times \Sigma^* \to \Sigma^*$ that is one-one in its second argument and satisfies: $x \in A$ if and only if pad $(x, y) \in A$, for all $x, y \in \Sigma^*$. If there exists a polynomial p such that $p(|\text{pad}\,(x, y)|) \geqq |x| + |y|$, for all $x, y \in \Sigma^*$, then A is *honestly paddable*. If there exists a polynomial time computable function decode: $\Sigma^* \to \Sigma^*$, such that decode (pad (x, y)) = y for all $x, y \in \Sigma^*$, then A is *invertibly paddable*. It was observed in [13] that if a set is invertibly paddable, then it has a padding function that can be inverted in both arguments in polynomial time. From this it easily follows that invertibly paddable sets are also honestly paddable.

THEOREM 3.2. *Let A be a recursive set not in* P. *If A is honestly paddable, then A is* P-*levelable.*

Proof. Let pad (x, y) be an honest padding function for A, and let p be a nondecreasing polynomial such that $p(|\text{pad}\,(x, y)|) \geqq |x| + |y|$ for all $x, y \in \Sigma^*$. Define

$$f(x) = \text{pad}\,(x, 0^{p(|x|)+1}).$$

Then $|f(x)| > |x|$, and $f(x) \in A$ if and only if $x \in A$, for all $x \in \Sigma^*$. Moreover, f can be computed in polynomial time.

Assume that A is not P-levelable, with a maximal P-subset E. Consider the following "boundary" set of E:

$$B = \{x | x \notin E, f(x) \in E\}.$$

Clearly, $B \in \text{P}$ and $B \cap E = \varnothing$. Further, $B \subseteq A$ because $E \subseteq A$ and f is membership-preserving. We prove that B is infinite, contradicting the maximality of E.

For any $x \in A$, the set $E_x = \{x, f(x), f(f(x)), \cdots\}$ is a subset of A. Because f is length-increasing E_x is infinite and in P. By the maximality of E, E_x must be eventually contained in E. In particular, for each $x \in A - E$ there is a smallest $n \geqq 1$ such that $f^{(n)}(x) \in E$. (Here $f^{(n)}$ denotes the n-fold composition of f.) For this $n, f^{(n-1)}(x) \in B$. Hence, from each $x \in A - E$ begins a length-increasing sequence $x, f(x), f(f(x)), \cdots$ of strings in $A - E$, ending in a string $y \in B$. Because $A \notin \text{P}$, $A - E$ is infinite. But because of the increase in length, only finitely many sequences can end in each $y \in B$. Thus, B is infinite. □

If we assume the padding to be invertible, we can prove a much stronger result. We say that a set A is *nonsparse* P-*levelable* if for any P-subset E of A there exists another P-subset E' of A such that the difference $E' - E$ is nonsparse.

THEOREM 3.3. *Let A be a recursive set not in* P. *If A is invertibly paddable, then A is nonsparse* P-*levelable.*

Proof. Assume to the contrary that A has a P-subset E that is maximal up to sparse sets. Consider the following boundary set of E:

$$B = \{z | z \notin E, \text{ but } z = \text{pad}\,(x, y) \text{ and pad } (x, yb_1b_2) \in E \text{ for some } b_1, b_2 \in \{0, 1\}\}.$$

Clearly $B \in \mathrm{P}$ and $B \subseteq A - E$, so B must be sparse. Let q be a polynomial such that $|B^{(n)}| \leq q(n)$ for all n.

Let M be any algorithm for A. We may assume, without loss of generality, that E contains the P-set

$$E_0 = \{\text{pad}(x, y) | M \text{ accepts } x \text{ in } |y| \text{ steps}\}.$$

(Otherwise replace E by $E \cup E_0$ in the proof.) Given any string $x \in A$, the set $\{\text{pad}(x, y) | y \in \Sigma^*\}$ is eventually contained in E_0. In fact, given any finite set $F \subseteq A$, the set

$$F^* = \{\text{pad}(x, y) | x \in F, y \in \Sigma^*\}$$

is eventually contained in E_0. Our proof is now based on the fact that if $F \subseteq A - E$, the number of strings in F^* increases at an exponential rate with the length of the padding y, whereas the boundary B admits only a polynomial "flow" of strings from $A - E$ to E. By choosing the basis set F so that it has a certain "critical mass," we can make the difference $F^* - E$ infinite, leading to a contradiction.

Let d be the density function of the set $A - (E \cup B)$, $d(n) = |(A - (E \cup B))^{(n)}|$, and let p be a polynomial such that $|\text{pad}(x, y)| \leq p(|x| + |y|)$ for all $x, y \in \Sigma^*$. It can be shown ([16, Cor. 3.4]) that because A is paddable and not in P, the difference $A - E'$ is nonsparse for any P-subset E' of A. Hence d is not polynomially bounded, and we can find an n such that

$$q(p(n+2k)) \leq 2^{k-1} d(n) \qquad \text{for all } k \geq 1.$$

(Let m be such that $q(p(t)) \leq mt^m$ for all $t \geq 1$, and M so large that $M \cdot 2^{k-1} \geq 2^m m k^m$ for all $k \geq 1$. Choose n so that $d(n) \geq Mn^m$. It follows that

$$q(p(n+2k)) \leq m(n+2k)^m \leq m \cdot 2^m k^m n^m \leq M \cdot 2^{k-1} n^m \leq 2^{k-1} d(n),$$

for all $k \geq 1$.)

Let $F = (A - (E \cup B))^{(n)}$ and define F^* as above. To prove that $F^* - E$ is infinite, divide F^* into levels;

$$F_k = \{\text{pad}(x, y) | x \in F, y \in \Sigma^*, |y| \leq 2k\}.$$

We prove by induction that for each $k \geq 1$, F_k contains at least $2^k d(n)$ strings not in $E \cup B$.

Case $k = 1$. The set F_1 contains $4|F| = 4d(n)$ strings, of which at most $q(p(n+2))$ may be in B, and none in E. Hence

$$|F_1 - (E \cup B)| \geq 4d(n) - q(p(n+2)) \geq 4d(n) - 2d(n) = 2d(n).$$

Case $k \to k+1$. Assume that $|F_k - (E \cup B)| \geq 2^k d(n)$. Then, as above,

$$\begin{aligned} |F_{k+1} - (E \cup B)| &\geq 4 \cdot 2^k d(n) - q(p(n+2(k+1))) \\ &\geq 2 \cdot 2^{k+1} d(n) - 2^{k+1} d(n) \\ &= 2^{k+1} d(n). \end{aligned}$$

□

The preceding result applies to very many natural intractable sets.

COROLLARY 3.4. *The following sets are nonsparse* P*-levelable, unless they are already in* P:

(i) SAT, TAUT *and all other "natural"* NP*-complete and co-*NP*-complete sets*;

(ii) ISO (*the set representing the graph isomorphism problem, which is not known to be* NP*-complete* [14]);

(iii) QBF.

Berman and Hartmanis [3] conjectured that all NP-complete sets are polynomially isomorphic. Their conjecture was motivated by the observation that all NP-complete sets known at the time were easily seen to be invertibly paddable, and invertibly paddable NP-complete sets can be shown to be isomorphic [3], [21]. Recently, however, Young [21] has introduced a class of structurally constructed NP-complete sets (the k-creative sets) which do not appear to have padding functions. We show that the proof technique of Theorem 3.2 can be used also to establish the P-levelability of all k-creative sets actually constructed in [21].

For the remainder of § 3 we will assume a standard enumeration of nondeterministic Turing machines $\{M_i\}$, where i is a "nice" encoding (see, for example, [7]) of machine M_i. (Define $L(M_i)$ to be empty if i does not encode a syntactically correct program.) We say that M_i *witnesses* $L(M_i) \in \mathrm{NP}^{(k)}$ if the run time of M_i on input x is bounded by $|i| \cdot |x|^k + |i|$, for every $x \in \Sigma^*$.

DEFINITION 3.5. A set C is *k-creative* if C is a member of NP and there exists a polynomially computable function f such that for each M_i which witnesses $L(M_i) \in \mathrm{NP}^{(k)}$, $f(i) \in L(M_i)$ if and only if $f(i) \in C$. The function f is called a *productive* function for the set $\bar{C} = \Sigma^* - C$.

It is not difficult to show that all k-creative sets are NP-complete. On the other hand, it is not obvious that k-creative sets exist. However, it was shown in [21] that every honest, one-one, polynomially computable function f is the productive function for the complement of some k-creative set. While it is not required for the definition, all of the k-creative sets C constructed in [21] have honest productive functions for $\bar{C}$.

THEOREM 3.6. *All k-creative sets C with honest productive functions for $\bar{C}$ are* P*-levelable, unless* $\mathrm{P} = \mathrm{NP}$.

Proof. Let C be k-creative and f an honest productive function for $\bar{C}$. Let M be an NTM which witnesses that C is in NP and let q_0 be a polynomial time bound for M. From the proof of Theorem 3.2 we see that it is sufficient to show that there is a polynomially computable length increasing membership preserving function, h, for C. We construct such a function by utilizing the membership properties of the productive function f, i.e., $h(x) = f(g(x))$, for the appropriately defined g. First we define $g(x)$ to be an encoding of a Turing machine so that $|g(x)| \geqq p(|x|)$, where p is a polynomial as yet unspecified, which behaves as follows: "input y; simulate M on input x; accept y if M accepts x." To compute g on input x one need only make minor modifications to the encoding of M and append $p(|x|)$ "instructions" which can never be executed. Thus, g can be computed in polynomial time and has the property that for all x, $|g(x)| \geqq p(|x|)$.

Notice that the set accepted by the program encoded by $g(x)$ is either Σ^* or the empty set. Moreover, it is not difficult to see that for all x, $M_{g(x)}$ witnesses that $L(M_{g(x)})$ is a member of $\mathrm{NP}^{(k)}$ whenever p is chosen so that $p(n) \geqq q_0(n)$ for all $n \geqq 0$. Hence, for such a polynomial p, it is possible to define $h(x) = f(g(x))$ and be certain that h is defined for all x in Σ^*.

From the definition of $g(x)$ we notice that if x is a member of C then $L(M_{g(x)}) = \Sigma^*$, while if x is not a member of C then $L(M_{g(x)}) = \varnothing$. Hence, $f(g(x)) \in L(M_{g(x)})$ if and only if $x \in C$. Because f is a productive function for $\bar{C}$, $f(g(x)) \in L(M_{g(x)})$ if and only if $f(g(x)) \in C$. Therefore, $x \in C$ if and only if $f(g(x)) \in C$, i.e., $h(x) = f(g(x))$ is membership preserving.

We have seen that for the appropriate polynomial p, in the definition of g, $h(x)$ is polynomially computable and membership preserving. It only remains to show that $h(x)$ is length increasing. Since f was assumed to be honest, there is a polynomial q_1 for which $q_1(|f(x)|) > |x|$ for all x in Σ^*. Let p be any polynomial which is everywhere

greater than q_0 and q_1 (for example, $p(n) = q_0(n) + q_1(n)$). It is easy to see that this is sufficient to insure that h is the desired polynomially computable, length increasing, membership preserving function. □

4. Self-reducibility and completeness. In this section we prove that P-levelability is a consequence of both a restricted form of self-reducibility and completeness for a deterministic time class. We begin with self-reducibility.

A set $A \subseteq \Sigma^*$ is (polynomial time) *self-reducible* [8], [10], [14], [18] if there exist a well-founded partial order $\leqq$ on Σ^*, and a polynomial time deterministic oracle Turing machine M, such that $L(M, A) = A$, and M on any input x queries only strings that strictly precede x in the order $\leqq$. (One simple choice of order is to define $x < y$ if and only if $|x| < |y|$.) It can be shown that any such self-reducing machine M determines a unique "fixed point" set, i.e., a set A such that $L(M, A) = A$. We prove this below in a special case.

A self-reducing machine M is *monotone* if for all $X, Y \subseteq \Sigma^*$, $X \subseteq Y$ implies $L(M, X) \subseteq L(M, Y)$. (Monotone self-reducibility is a generalization of "positive truth table self-reducibility" [10], [20], which in turn is a generalization of the common "disjunctive" and "conjunctive" reducibilities.) The question of the existence of a unique fixed point set is slightly simpler for monotone machines than in general.

LEMMA 4.1. *Let M be a monotone self-reducing machine. Then there is a unique set A such that $L(M, A) = A$.*

Proof. Define

$$A_0 = \varnothing;$$

$$A_{i+1} = L(M, A_i) \qquad \text{for } i \geqq 0.$$

Because M is monotone, $A_0 \subseteq A_1 \subseteq A_2 \subseteq \cdots$. We claim that the set

$$A = \bigcup_{i \geqq 0} A_i$$

is the unique fixed point. First, because M can query only finitely many strings on each input,

$$x \in L(M, A) \quad \text{iff}$$

$$x \in L(M, A_i) = A_{i+1} \quad \text{for some } i \geqq 0 \quad \text{iff}$$

$$x \in A.$$

Hence $L(M, A) = A$. To show uniqueness, assume that $B \neq A$ is another set for which $L(M, B) = B$. Let x be a minimal element with respect to the reduction order in the symmetric difference $A \triangle B$. Then for each string y queried by M on input x, $y \in A$ if and only if $y \in B$. Hence $x \in A$ if and only if, $x \in B$, contradicting our choice of x. □

We need to impose an additional restriction on our self-reducing machines. Let M be a self-reducing machine for a set A. M determines a "query" ordering $\leqq_M$ on Σ^*, given by the reflexive and transitive closure of the following relation $<^1_M$: $y <^1_M x$ if M queries y on input x and oracle A. Let M be a monotone machine, and let the sequence $A_0, A_1, A_2, \cdots$ be defined as in Lemma 4.1. We require M to have the following property:

(*) For each $y \in \Sigma^*$ there is some $i \geqq 0$ such that $x \in A$ and $y \leqq_M x$ implies $x \in A_i$.

In other words, for each string y there is a bound i such that no string in A "higher" than A_i depends on y. This condition is satisfied, for instance, by the length preserving

self-reductions of Karp and Lipton in [8], but not necessarily by the more general ones considered by Meyer and Paterson in [14]. However, all known natural examples of self-reducible sets have monotone self-reducing machines satisfying condition (*).

THEOREM 4.2. *Let A be a recursive set not in* P. *If A is monotone self-reducible via a machine satisfying* (*), *then A is* P-*levelable.*

Proof. Let M be a polynomial time monotone self-reducing machine for A, satisfying condition (*), and let E be any P-subset of A. Consider the set $L(M, E)$. Clearly $L(M, E) \in \mathrm{P}$, and because M is monotone, $L(M, E) \subseteq L(M, A) = A$. We prove that the difference $L(M, E) - E$ is infinite, thus showing that E is not maximal.

Let $\leqq_M$ be the query ordering determined by M, and $A_0, A_1, \cdots$ the sequence of sets defined in Lemma 4.1. We establish first that for each $x \in A - E$ there exists a $y \in L(M, E) - E$ such that $y \leqq_M x$. This is certainly true of $x \in A_1 - E$, since $A_1 = L(M, \varnothing) \subseteq L(M, E)$; simply choose $y = x$. So let $x \in A_i - E$ for some $i \geqq 2$. Then either $x \in L(M, E)$, and we are done, or $x \in A_i - L(M, E)$. In the latter case there must exist an $x' \in A_{i-1} - E$ such that $x' \leqq_M x$, because otherwise $x \in L(M, E \cap A_{i-1}) \subseteq L(M, E)$, contrary to our assumption. Repeating the argument leads eventually to an element $y \in L(M, E) - E$ with $y \leqq_M x$.

Now, assume that $L(M, E) - E$ is finite. Condition (*) then gives a uniform index k such that if $x \in A$ and if $y \leqq_M x$ for some $y \in L(M, E) - E$, then $x \in A_k$. By the above argument, this means that $A - E \subseteq A_k$. But it is easy to see that each of the sets A_k, $k \geqq 0$, is in P, so we get $A = E \cup (A - E) = E \cup A_k \in \mathrm{P}$, contrary to our initial assumption. □

As a corollary, we can prove the P-levelability of some interesting sets which do not seem to have padding functions.

COROLLARY 4.3. *The following sets are* P-*levelable, unless they are in* P:

(i) FACTOR $= \{\langle m, a, b\rangle | m, a, b$ *natural numbers,* $1 < a \leqq b < m$, m *has a factor between* a *and* $b\}$ [8], [14];

(ii) *for any nondeterministic Turing machine* M:

$$L_M = \{\langle x, c\rangle | c \text{ codes an initial segment of an accepting computation of } M \text{ on input } x\}.$$ □

Our final P-levelability result applies to, for example, all EXPTIME-complete sets.

THEOREM 4.4. *Let A be a recursive set not in* P. *If A is complete for some deterministic time class, then A is* P-*levelable.*

Proof. Assume that A is not P-levelable. It was shown in [15] that A then is the disjoint union of an infinite P-immune set C and a set in P. It is easy to see that if A is complete for some class of sets, then so is C. However, L. Berman proved in [2] that infinite P-immune sets cannot be complete for deterministic time classes. □

5. Reductions preserving P-levelability. There seems to be no simple way of extending our results to prove the P-levelability of all NP-complete sets. In general, very little is known about the structure of arbitrary NP-complete sets. The straightforward approach to proving such results would be to first establish that some known NP-set has the structural property in question, and then show that the property is preserved under $\leqq_m^{\mathrm{P}}$-reductions. Unfortunately, $\leqq_m^{\mathrm{P}}$-reductions can behave very badly on polynomial time recognizable subsets, which are our main concern. (In fact, a $\leqq_m^{\mathrm{P}}$-reduction can map a set in P onto any r.e. set.) In this section we show that this is the essence of our problem; if a reduction behaves well "locally" on P-sets, then it preserves the "global" property of P-levelability. A subclass of the well behaved reductions also gives us a natural reducibility characterization of sets that are not P-levelable.

DEFINITION 5.1. A polynomial time many-one reduction f from A to B is P-*to*-P if for each $E \subseteq A$, $E \in \mathrm{P}$, the image set $f[E]$ is in P. The reduction is P-*to-finite* if $f[E]$ is finite for each $E \subseteq A$, $E \in \mathrm{P}$.

A well-known class of P-to-P reductions is formed by the polynomial time isomorphisms of Berman and Hartmanis [3]. Their conjecture that all NP-complete sets are polynomially isomorphic implies, of course, very strongly that these sets are related by P-to-P reductions.

We begin with the characterization theorem.

THEOREM 5.2. *A recursive set A is not* P-*levelable if and only if there is a* P-*to-finite reduction from A to some recursive set B.*

Proof. Assume first that $A \neq \emptyset$ is not P-levelable, with a maximal P-subset E. A P-to-finite reduction from A can be obtained by fixing any element $a \in A$ and defining

$$f(x) = \begin{cases} a & \text{if } x \in E, \\ x & \text{otherwise.} \end{cases}$$

Clearly if A is recursive, then so is $f[A] = (A - E) \cup \{a\}$, and f is a reduction from A to $f[A]$.

To show the converse, let f be a P-to-finite polynomial time reduction from A to B, and let M_B be an algorithm for B. Define

$$E = \{x \mid M_B \text{ accepts } f(x) \text{ in } |x| \text{ steps}\}.$$

It is easy to see that $E \subseteq A$ and $E \in \mathrm{P}$. We claim that E is in fact maximal among the P-subsets of A. To show this, let F be another P-subset of A. Because f is P-to-finite, the image $f[F] \subseteq B$ is finite, and so there is a uniform bound n such that M_B accepts each $y \in f[F]$ in n steps. But then each x in F with $|x| \geqq n$ is also in E. Hence F is eventually contained in E. □

We can now use this characterization to establish the preservation result.

THEOREM 5.3. *Let A and B be recursive sets, and let A be* P-*to*-P *reducible to B. Then A is* P-*levelable implies that B is* P-*levelable.*

Proof. Assume that B is not P-levelable, with a maximal P-subset F. Let f be a P-to-P reduction from A to B. Let E be a P-set in $A - f^{-1}[F]$. Because f is P-to-P, the image $f[E] \subseteq B - F$ is in P. In fact, by the maximality of F, $f[E]$ is finite. But because $F \in \mathrm{P}$, also $f^{-1}[F] \in \mathrm{P}$, and we can modify f to map all of $f^{-1}[F]$ to a single point. This new reduction will be P-to-finite, so by the preceding theorem, A cannot be P-levelable. □

We conjecture that all NP-complete sets are related by P-to-P reductions, and so are P-levelable if $\mathrm{P} \neq \mathrm{NP}$. This is a very weak consequence of the Berman–Hartmanis conjecture: it is even vacuously true if $\mathrm{P} = \mathrm{NP}$, in which case the latter conjecture fails badly [13].

6. Concluding remarks. We have established that P-levelability is a common property among the natural intractable sets, by showing that it follows from a number of other frequently encountered properties. We have proved the P-levelability of intractable sets that are, for example, honestly paddable, or monotone self-reducible in a certain restricted way, or complete for deterministic time classes.

Several improvements and extensions naturally suggest themselves. First, one would like to establish the P-levelability of *all* NP-complete sets, instead of just the "known" ones. Unfortunately, this is likely to be a difficult task, because the result would be equivalent to the nonexistence of P-immune NP-complete sets [5]. This was

conjectured by L. Berman in [2], but the conjecture has remained unproven. One possible approach would be to try to prove that all NP-complete sets are related by P-to-P reductions, as suggested in § 5. It would also be interesting to investigate other common reductions, for example, is it the case that all sets complete for NP (or PSPACE) with respect to log-space reducibility are P-levelable?

Another shortcoming one would like to see corrected is the need for the extra condition on self-reducibilities in Theorem 4.2. Proving the result without the condition would establish the nonexistence of monotone self-reducible P-immune sets, and conversely. A more general question is: what are the relationships between P-immunity and the different notions of self-reducibility?

In a different vein, it would be interesting to extend these results to other notions of polynomial approximation. Recall that in § 1 we required the approximation algorithms to be "safe:" if they cannot reach a correct decision, they output a "?." Yesha [20] has introduced a more general "unsafe" approximation notion, in which the algorithms may make errors on a small fraction of the inputs. Our techniques do not seem to work in this generalization.

Acknowledgments. The authors would like to thank Professor Ronald V. Book for his unfailing interest and support during the course of this work, and Mrs. Leslie Wilson for her expert typing of the numerous drafts of this paper.

REFERENCES

[1] J. L. Balcázar and U. Schöning, *Bi-immune sets for complexity classes*, Math. Systems Theory, to appear.

[2] L. Berman, *On the structure of complete sets: almost everywhere complexity and infinitely often speedup*, Proc. 17th IEEE Symposium on Foundations of Computer Science, 1976, pp. 76–80.

[3] L. Berman and J. Hartmanis, *On isomorphism and density of* NP *and other complete sets*, this Journal, 6 (1977), pp. 305–322.

[4] M. Blum and I. Marques, *On complexity properties of recursively enumerable sets*, J. Symbolic Logic, 38 (1973), pp. 579–593.

[5] D.-Z. Du, T. Isakowitz and D. Russo, *Structural properties of complexity cores*, submitted for publication.

[6] P. Flajolet and J. M. Steyaert, *On sets having only hard subsets*, Proc. 2nd International Colloquium on Automata, Languages, and Programming, 1974, Lecture Notes in Computer Science 14, Springer-Verlag, Berlin, pp. 446–457.

[7] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[8] R. Karp and R. Lipton, *Some connections between nonuniform and uniform complexity classes*, Proc. 12th ACM Symposium on Theory of Computing, 1980, pp. 302–309.

[9] K. Ko, *Non-levelable sets and immune sets in the accepting density hierarchy in* NP, submitted for publication.

[10] ——— *On self-reducibility and weak* P*-selectivity*, J. Comput. System Sci., 26 (1983), pp. 209–221.

[11] K. Ko and D. Moore, *Completeness, approximation and density*, this Journal, 10 (1981), pp. 787–796.

[12] N. Lynch, *On reducibility to complex or sparse sets*, J. Assoc. Comput. Mach., 22 (1975), pp. 341–345.

[13] S. R. Mahaney and P. Young, *Orderings of polynomial isomorphism types*, Theoret. Comput. Sci., to appear.

[14] A. R. Meyer and M. S. Paterson, *With what frequency are apparently intractable problems difficult?* Tech. Rep. TM-126, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, 1979.

[15] P. Orponen, *A classification of complexity core lattices*, submitted for publication.

[16] P. Orponen and U. Schöning, *The structure of polynomial complexity cores*, Mathematical Foundations of Computer Science 1984, Lecture Notes in Computer Science 176, Springer-Verlag, Berlin, pp. 452–458.

[17] D. A. Russo and P. Orponen, *A duality between recursive complexity cores and polynomial time computable subsets*, submitted for publication.

[18] C. P. SCHNORR, *Optimal algorithms for self-reducible problems*, Proc. 3rd International Colloquium on Automata, Languages, and Programming, 1976, Edinburgh Univ. Press, pp. 322–337.

[19] R. SOARE, *Computational complexity, speedable and levelable sets*, J. Symbolic Logic, 42 (1977), pp. 545–563.

[20] Y. YESHA, *On certain polynomial-time truth-table reducibilities of complete sets to sparse sets*, this Journal, 12 (1983), pp. 411–425.

[21] P. YOUNG, *Some structural properties of polynomial reducibilities*, Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 392–401.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
006

PROBABILISTIC BOUNDS ON THE PERFORMANCE OF LIST SCHEDULING*

JOHN L. BRUNO† AND PETER J. DOWNEY‡

Abstract. The problem of scheduling tasks on m processors to minimize the schedule length (makespan) is NP-complete. Here we study the behavior of list schedules under the assumptions that there are no task precedence constraints and that task times are chosen from a *uniform* distribution.

We show that, given a desired degree of confidence $1-\varepsilon$, we can find a minimum sample size N such that if $n \geqq N$ and the n task times $\bar{X} = (X_1, \cdots, X_n)$ are chosen from any uniform distribution, then

$$\mathbf{P}\left[\frac{L(\bar{X})}{OPT(\bar{X})} < 1 + \frac{4(m-1)}{n}\right] > 1-\varepsilon$$

where $L(\bar{X})$ is the length of any list schedule and $\mathrm{OPT}(\bar{X})$ is the length of the optimal schedule. Thus for n sufficiently large, the performance of any list schedule can be made arbitrarily close to that of the optimal policy with any desired degree of confidence. For example, for $m=2$ and $\varepsilon = 0.01$, the ratio is bounded by 1.11 when $n=36$ and bounded by 1.03 when $n=100$.

Key words. list scheduling, approximation algorithms, Kolmogorov-Smirnov statistics, makespan scheduling, probabilistic algorithm analysis, NP-complete problems

AMS (MOS) subject classifications. 68C, 05B, 68Q
CR categories. F.2.2, G.2.1

1. Introduction.

The *makespan scheduling problem* is: given n tasks with processing times $X_1, X_2, \cdots, X_n$, find a way to sequence these tasks on m identical processors so as to minimize the makespan (the finishing time of the last task). There are no precedence or resource constraints affecting the ordering of the tasks.

It is well known that this problem is NP-complete for $m \geqq 2$ (and NP-complete in the strong sense for $m \geqq 3$ [Gar79]). We can interpret this result as saying that it is unlikely there will ever be found a simple scheduling rule which produces an optimal schedule for any given set of task times.

In view of this complexity result, work has focused on finding "approximation algorithms": simple, nonoptimal but very good scheduling policies. The standard measure of "goodness" is the relative performance—the ratio of the policy's finishing time to that of the optimal. The policies considered are restricted to the class of *list schedules*, described as follows. Based on the task times, a list $L = (i_1, \cdots, i_n)$ of task indices is computed. The tasks are assigned to processors in the order induced by L; an assignment is made from the list whenever a processor becomes free during scheduling. Since complete information is known on task times when the list L is computed, and since no idle time is inserted during scheduling, list schedules coincide with work-conserving schedules in this model.

One widely studied example of a list schedule is the *LPT* (*L*argest *P*rocessing *T*ime first) schedule, which lists the tasks in nonincreasing order by time.

* Received by the editors February 18, 1983, and in final form January 14, 1985.

† Department of Computer Science and the Computer Systems Laboratory, University of California, Santa Barbara, California 93106. The work of this author was partially supported by the National Science Foundation under grant MCS80-04257.

‡ Department of Computer Science, The University of Arizona, Tucson, Arizona 85721. The work of this author was partially supported by the National Science Foundation under grant MCS80-04679.

Suppose the task times are $X = (X_1, \cdots, X_n)$. For a given list schedule determined by list L, let $L(\bar{X})$ be the finishing time of the schedule. Let $LPT(\bar{X})$ be the finishing time of the *LPT* schedule, and let $OPT(\bar{X})$ be that of the optimal schedule.

Intuitively, *LPT* should be among the better list schedules. Graham [Gra69] has studied the worst-case behavior of *LPT*, with the result

$$\frac{LPT(\bar{X})}{OPT(\bar{X})} \leqq \frac{4}{3} - \frac{1}{3m}. \tag{1.1}$$

Thus *LPT* finishes not more than $\frac{1}{3}$ later than the optimal, for any set of tasks.

Graham's result is the best possible bound for *LPT* in the worst case [Gra69]. In this paper we address the probabilitistic question: how well do *LPT* and other list schedules perform "most of the time"? In other words, if task times are not chosen by a malevolent adversary, but only by a randomizing (indifferent) one, can the bound in (1.1) be improved upon? Coffman, Frederickson and Lueker [Coff84] have examined the expectation of the makespan for the *LPT* schedule, and compared it to the expectation of the optimum. Here we study the distribution of the relative performance for any list schedule.

In § 3 we show that, given a desired degree of confidence $1-\varepsilon$, we can find a minimum sample size N such that if $n \geqq N$ and n task times $\bar{X}$ are chosen from any uniform distribution, then

$$\mathbf{P}\left[\frac{L(\bar{X})}{OPT(\bar{X})} < 1 + \frac{4(m-1)}{n}\right] > 1-\varepsilon \tag{1.2}$$

for any list schedule L. The minimum sample size N depends only on ε and not on the distribution of times or on the structure of the schedule L. Actually, Theorem 3.2 gives a sharper result[1] than (1.2), in which it is possible to quantify the size of N.

The results in this paper are not about "probabilistic scheduling", since here scheduling is done with complete a priori information concerning task times. The probabilistic aspect comes in sampling the task processing times from a distribution before making scheduling decisions. The results are more properly regarded as probabilistic analyses of deterministic scheduling policies. While Graham's result (1.1) gives information about the worst that can happen in using *LPT*, (1.2) provides information about the shape of the distribution of finishing times of *any* list schedule when task times are uniformly distributed.

A result which holds in general for any list schedule is of interest since it reflects the realities of scheduling "on-line", without any special ordering of the tasks beforehand.

The result (1.2) holds for all *sufficiently large n*, where the choice of n depends only upon the desired degree of confidence $1-\varepsilon$, and not upon the particular uniform distribution from which the X_i are chosen. To see the force of this result, it will be instructive to compare (1.2) with a deterministic result. Suppose we agree ahead of time that all task processing times will be restricted to integers in the interval $[A, B]$. Then by a theorem due to Graham (Theorem 3.1 below), for every n task times $\bar{X}$ and any list schedule L

$$\frac{L(\bar{X})}{OPT(\bar{X})} \leqq 1 + \frac{(m-1)}{n}\frac{B}{A}. \tag{1.3}$$

[1] Since the submission of this paper, the bounds given by Theorem 3.2 have been improved still further using results on sums of i.i.d. uniform variates. This work is reported in [Cof85], where an analysis for exponential task times also appears.

TABLE 1

Quantiles of the one-sided Kolmogorov-Smirnov test statistic [Mil56]. *Values of* $d_{n,\varepsilon}$ *displayed are defined by* $\mathbf{P}[\mathbf{D}_n^+ \leqq d_{n,\varepsilon}] = 1 - \varepsilon = P_n(d_{n,\varepsilon})$.

$1-\varepsilon =$		Values of $d_{n,\varepsilon}$ .95	.99
$n =$	5	.509	.627
	10	.369	.457
	15	.304	.377
	20	.265	.329
	25	.238	.295
	30	.218	.270
	35	.202	.251
	40	.189	.235
$n > 40$		$\frac{1.22}{\sqrt{n}}$	$\frac{1.52}{\sqrt{n}}$

As n, the number of tasks, is allowed to grow in (1.3), L comes as close to the optimal as desired.

Is (1.3) stronger than (1.2), in being deterministic instead of probabilistic? No, because in (1.3) the bound results from choosing a very large number of tasks from a bounded interval, so that none of the execution times is large compared with the sum of all the execution times. Thus *given* $[A, B]$, for all sufficiently large sets of tasks, L is as close to OPT as desired. By contrast, (1.2) allows the interval $[A, B]$ to be chosen *after* n is bound. Indeed, the size of the interval plays no role in the result, so that task times could be allowed to range very much larger than n. The price paid for this freedom is that (1.2) now holds, not for every choice of $\bar{X}$ (i.e., worst case), but only with high probability *if* the data are distributed uniformly.

In § 2 the main probabilistic tools used in this paper are presented. Section 3 contains the main result (Theorem 3.2). In § 4 the result is illustrated by numerical examples.

2. Preliminaries. Let $X_1, \cdots, X_n$ be mutually independent random variables with a common continuous distribution F. The vector $\bar{X} = (X_1, \cdots, X_n)$ is referred to as a *random sample of size n from F.* If

$$X_{(1)} \leqq X_{(2)} \leqq \cdots \leqq X_{(n)}$$

are the n values of $\bar{X}$ arranged in nondecreasing order, then the random variable $X_{(i)}$ is the *ith order statistic* of the sample.

For an event $[E]$, let $I[E]$ take the value 1 if $[E]$ occurs and 0 otherwise. The *sample distribution function* of $\bar{X} = (X_1, \cdots, X_n)$ is defined as

$$\mathbf{F}_n(x) = \frac{1}{n} \sum_{i=1}^{n} I[X_i \leqq x]. \tag{2.1}$$

That is, $\mathbf{F}_n(x)$ is the proportion of sample values lying in the interval $(-\infty, x]$.

Given a sample $\bar{X}$ of size n define the random variable

$$\mathbf{D}_n^+ = \sup_{-\infty \leqq x \leqq \infty} (\mathbf{F}_n(x) - F(x)). \tag{2.2}$$

$\mathbf{D}_n^+$ is referred to as the *Smirnov statistic* or the *one-sided Kolmogorov-Smirnov statistic* of the sample.

Intuitively we would expect that as the sample size n gets large, $\mathbf{D}_n^+$ would approach zero with high probability. Theorem 2.2 below shows this is true. More remarkably, the distribution of $\mathbf{D}_n^+$ can be completely described; it depends *only* on n and not at all upon the distribution F.[2] To see this, we show that F can be replaced in (2.2) by U, the uniform distribution on $[0, 1]$. The following result is well known in different forms [Dur73].

THEOREM 2.1. *Let $\bar{X}$ be a sample of size n from a continuous distribution F. If $\mathbf{D}_n^+$ is defined by* (2.2),

$$\mathbf{D}_n^+ = \max_{1 \leqq i \leqq n} \left(\frac{i}{n} - Y_{(i)} \right), \tag{2.3}$$

where $\bar{Y}$ is a sample of size n from U, the uniform distribution on $[0, 1]$.

Proof. Define the inverse of F by

$$F^{-1}(t) = \sup_{-\infty \leqq x \leqq \infty} \{x : F(x) \leqq t\}, \qquad t \in [0, 1].$$

Then F^{-1} is nondecreasing, right-continuous and satisfies

$$F(x) \leqq t \quad \text{if and only if} \quad x \leqq F^{-1}(t), \tag{2.4}$$

$$F(F^{-1}(t)) = t. \tag{2.5}$$

Given the sample $\bar{X}$ from F, the n random variables $Y_i = F(X_i)$ are independent and distributed uniformly on $[0, 1]$. For by (2.4), the event $[Y_i \leqq t] = [F(X_i) \leqq t]$ is identical to the event $[X_i \leqq F^{-1}(t)]$, and the latter has probability $F(F^{-1}(t)) = t$. Thus $\bar{Y}$ is a sample from U.

Let $\mathbf{U}_n$ be the sample distribution function of sample $\bar{Y}$. Let $Y_{(i)}$ be the ith order statistic of the sample.

Now

$$\mathbf{D}_n^+ = \sup_{-\infty \leqq x \leqq \infty} [\mathbf{F}_n(x) - F(x)] = \sup_{\substack{x = F^{-1}(t) \\ 0 \leqq t \leqq 1}} [\mathbf{F}_n(x) - F(x)] \tag{2.6}$$

since, because F is continuous, F^{-1} takes on all values of x except in those intervals of x on which F is constant. But in any interval in which F is constant, we also have $\mathbf{F}_n$ constant, as there is zero probability of any X_i occurring there.

From (2.6), using (2.5) and then (2.4),

$$\begin{aligned}
\mathbf{D}_n^+ &= \sup_{0 \leqq t \leqq 1} (\mathbf{F}_n F^{-1}(t) - F F^{-1}(t)) \\
&= \sup_{0 \leqq t \leqq 1} \left(\frac{1}{n} \sum_{i=1}^{n} I[X_i \leqq F^{-1}(t)] - t \right) \\
&= \sup_{0 \leqq t \leqq 1} \left(\frac{1}{n} \sum_{i=1}^{n} I[F(X_i) \leqq t] - t \right) \\
&= \sup_{0 \leqq t \leqq 1} \left(\frac{1}{n} \sum_{i=1}^{n} I[Y_i \leqq t] - t \right) \\
&= \sup_{0 \leqq t \leqq 1} (\mathbf{U}_n(t) - t).
\end{aligned}$$

[2] This *distribution-free* property makes $\mathbf{D}_n^+$ important in testing for the goodness of fit of data to an hypothesized distribution. The *one-sided Kolmogorov–Smirnov test* is described in [Con80].

Since $\mathbf{U}_n$ is a step function with jumps at $Y_{(i)}$, this last difference has local maxima at the jumps. So the supremum can be replaced with

$$\mathbf{D}_n^+ = \max_{1 \leqq i \leqq n} (\mathbf{U}_n(Y_{(i)}) - Y_{(i)}) = \max_{1 \leqq i \leqq n} \left(\frac{i}{n} - Y_{(i)} \right).$$

The last equality follows since exactly i of the sample values are $\leqq Y_{(i)}$ by definition. This last equality establishes the result. □

By the above theorem, for each n the cdf

$$P_n(x) = \mathbf{P}[\mathbf{D}_n^+ \leqq x] \tag{2.7}$$

is well defined. The following closed form representation of this cdf is derived in [Bir51]:

THEOREM 2.2. (Birnbaum and Tingey).

$$P_n(x) = 1 - x \sum_{i=0}^{\lfloor n - nx \rfloor} \binom{n}{i} \left(\frac{i}{n} + x \right)^{i-1} \left(1 - \frac{i}{n} - x \right)^{n-i}. \tag{2.8}$$ □

Because of the importance of $\mathbf{D}_n^+$ in hypothesis testing, extensive tables have been computed based on (2.8) [Bir51], [Mil56], [Owe62]. For testing purposes, the function (2.8) needs to be inverted; the tables contain the solution $d_{n,\varepsilon}$ of the equation

$$P_n(d_{n,\varepsilon}) = 1 - \varepsilon \tag{2.9}$$

for selected n and small values of ε. Table 1 gives a short table of values of $d_{n,\varepsilon}$ compiled from [Bir51] and [Mil56]. These data will be used to work examples in § 4 below.

The large sample behavior of $P_n(x)$ is described by an asymptotic result [Dur73].

THEOREM 2.3. (Smirnov). For $n \to \infty$

$$P_n\left(\frac{x}{\sqrt{n}} \right) = 1 - e^{-2x^2} \left(1 - \frac{2x}{3\sqrt{n}} + O\left(\frac{1}{n} \right) \right). \tag{2.10}$$

Computations in [Bir51], [Mil 56] show that for $0.001 \leqq \varepsilon \leqq 0.1$ the values of $d_{n,\varepsilon}$ computed from (2.10) are greater than those given by the exact formula (2.8), and are in close agreement for $n \geqq 50$.

3. List schedules. Let L denote a particular list schedule, and let $L(\bar{X})$ be the finishing time achieved by this schedule on m processors. The worst case performance of any list schedule can be measured by the following result, due to Graham [Gra76, Thm. 5.3]. It has a very simple proof which we repeat here.

THEOREM 3.1. (Graham).

$$\frac{L(\bar{X})}{OPT(\bar{X})} \leqq 1 + (m-1) \frac{\max_i X_i}{\sum_{i=1}^n X_i}. \tag{3.1}$$

Proof. Write $X_{(n)}$ for $\max_i X_i$. Let f be the index of the last task to finish under L, and let S_f be the starting time of f. Then $L(\bar{X}) = S_f + X_f$. No processor can be idle before time $S_f = L(\bar{X}) - X_f$, and a fortiori no processor can be idle before time $L(\bar{X}) - X_{(n)}$. Since at least one processor is busy for $L(\bar{X})$ units of time, it follows that

$$\sum_{i=1}^n X_i \geqq L(\bar{X}) + (m-1)(L(\bar{X}) - X_{(n)}).$$

This inequality and the observation that

$$OPT(\bar{X}) \geqq \frac{\sum_{i=1}^{n} X_i}{m} \tag{3.2}$$

imply

$$OPT(\bar{X}) \geqq \frac{1}{m}(L(\bar{X}) + (m-1)(L(\bar{X}) - X_{(n)}))$$

which yields

$$\frac{L(\bar{X})}{OPT(\bar{X})} \leqq 1 + \frac{(m-1)}{m} \frac{X_{(n)}}{OPT(\bar{X})} \leqq 1 + (m-1)\frac{X_{(n)}}{\sum_{i=1}^{n} X_i}$$

where we have used (3.2) once more. The last inequality proves the theorem. □

Theorem 2.1 and Graham's theorem are used to prove the main result:

THEOREM 3.2. *For all $\varepsilon > 0$ and for any positive integer n, if $X_1, \cdots, X_n$ are task processing times sampled from a uniform distribution on any interval $[A, B]$ with $A \geqq 0$, then for any list schedule L*

$$\mathbf{P}\left[\frac{L(\bar{X})}{OPT(\bar{X})} < 1 + \frac{2(m-1)}{n} \frac{1}{1-2d_{n,\varepsilon}}\right] \geqq 1 - \varepsilon. \tag{3.3}$$

Proof. By (2.9), it will be enough to show that for arbitrary d

$$\mathbf{P}\left[\frac{L(\bar{X})}{OPT(\bar{X})} < 1 + \frac{2(m-1)}{n} \frac{1}{1-2d}\right] \geqq P_n(d). \tag{3.4}$$

Set $R = B - A$ and assume $R > 0$. Now if the X_i are chosen from a uniform distribution on $[A, B]$, then the random variables

$$Y_{(i)} = \frac{X_{(i)} - A}{R}$$

are the order statistics of a sample of size n from the uniform distribution on $[0, 1]$. Theorem 2.1 then yields

$$\mathbf{P}\left[\max_i \left(\frac{i}{n} - \left(\frac{X_{(i)} - A}{R}\right)\right) \leqq d\right] = P_n(d),$$

so that

$$\mathbf{P}\left[\max_i \left(\frac{R}{n} i + A - X_{(i)}\right) \leqq dR\right] = P_n(d). \tag{3.5}$$

Before using (3.5), let us derive some implications from the inequality

$$\max_i \left(\frac{R}{n} i + A - X_{(i)}\right) \leqq dR. \tag{3.6}$$

In what comes below, we will assume that this inequality holds, and only later return to the probabilistic argument.

The inequality (3.6) is equivalent to

$$\frac{R}{n} i + A - X_{(i)} \leqq dR \quad \text{for all } i.$$

Summing over all i in this inequality yields

$$\frac{R(n+1)}{2}+An-dRn\leqq\sum_{i=1}^{n}X_i. \tag{3.7}$$

Putting (3.7) into Graham's result (3.1) gives us

$$\frac{L(\bar{X})}{OPT(\bar{X})}\leqq 1+(m-1)\frac{X_{(n)}}{R(n+1)/2+An-dRn}$$

and since $X_{(n)}\leqq R+A$ we obtain

$$\frac{L(\bar{X})}{OPT(\bar{X})}\leqq 1+(m-1)\frac{R+A}{(R/2+A)n-dRn+R/2} \tag{3.8}$$

$$<1+(m-1)\frac{R+A}{(R/2+A)n-dRn}.$$

Dividing the numerator and denominator of (3.8) by $(R/2+A)n$ and using the fact that $(R/2+A)=(R/2)(1+2A/R)$, we obtain

$$\frac{L(\bar{X})}{OPT(\bar{X})}<1+\frac{2(m-1)}{n}\,\frac{(1+A/R)/(1+2A/R)}{1-2d/(1+2A/R)}. \tag{3.9}$$

The right side of (3.9) can be bounded, using $A/R\geqq 0$, to yield

$$\frac{L(\bar{X})}{OPT(\bar{X})}<1+\frac{2(m-1)}{n}\,\frac{1}{1-2d} \tag{3.10}$$

in which A and B play no role.

To summarize: the inequality (3.6) implies (3.10). Thus the probability of the event represented by (3.10) must dominate the probability of the event represented by (3.6). But by (3.5), the latter probability is $P_n(d)$. Thus the probability of (3.10) is at least $P_n(d)$, and (3.4) is proved. □

COROLLARY 3.3. *For all $\varepsilon>0$ and for all sufficiently large n, if $X_1,\cdots,X_n$ are task processing times sampled from a uniform distribution on any interval $[A, B]$ with $A\geqq 0$, then for any list schedule L*

$$\mathbf{P}\left[\frac{L(\bar{X})}{OPT(\bar{X})}<1+\frac{4(m-1)}{n}\right]\geqq 1-\varepsilon. \tag{3.11}$$

Proof. From (3.4) with $d=\frac{1}{4}$, the probability in (3.11) is at least $P_n(\frac{1}{4})$. By Smirnov's result (2.10), $P_n(\frac{1}{4})$ can be made greater than $1-\varepsilon$ for all sufficiently large n. □

4. Examples and conclusion. Since *LPT* is an example of a list schedule, Theorem 3.2 extends our knowledge about the behavior of the ratio $LPT(\bar{X})/OPT(\bar{X})$ beyond the information given by the deterministic bound (1.1), under the uniformity assumptions. Let us denote the bound in Theorem 3.2 by

$$\beta_m(n,\varepsilon)=1+\frac{2(m-1)}{n}\,\frac{1}{1-2d_{n,\varepsilon}}. \tag{4.1}$$

In the following examples we numerically examine the sharpness of (4.1) for particular values of m and n.

Example 1. Suppose we seek 99% confidence bounds on $LPT(\bar{X})/OPT(\bar{X})$ and there are $m=2$ processors. Figure 1 plots $\beta_2(n, 0.01)$ against n for $n=20(1)100$. Also

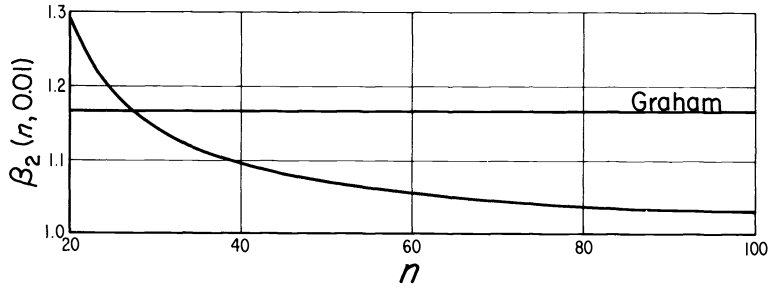


FIG. 1. 99% *confidence bounds on list scheduling for* $m = 2$ *processors. For comparison, Graham's deterministic bound* $\frac{7}{6}$ *for LPT scheduling is shown. The crossover occurs at* $n = 28$.

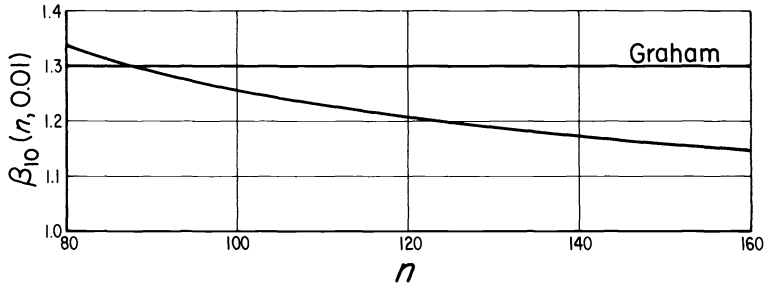


FIG. 2. 99% *confidence bounds on list scheduling for* $m = 10$ *processors. For comparison, Graham's deterministic bound* $\frac{13}{10}$ *for LPT scheduling is shown. The crossover occurs at* $n = 89$.

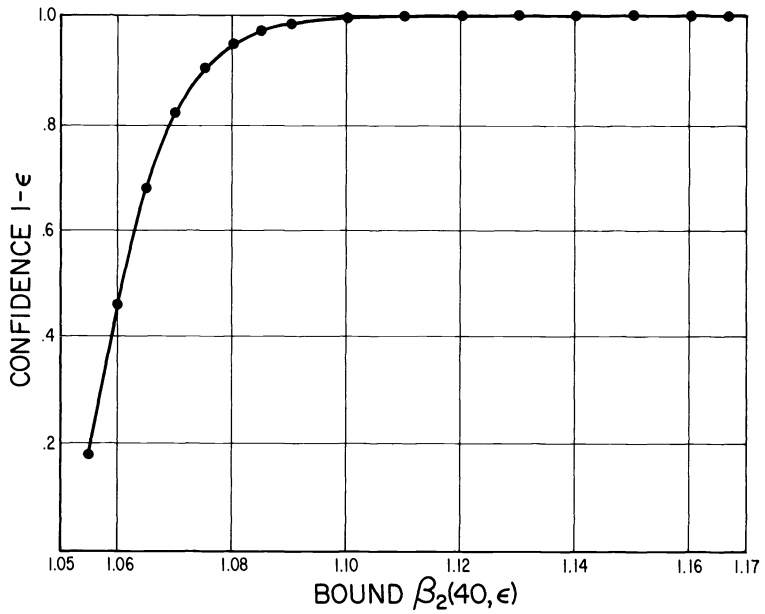


FIG. 3. *For* $m = 2$ *processors and a sample size* $n = 40$, *the graph depicts the tradeoff between bound values* $\beta_2(40, \varepsilon)$ *and confidence in these bounds* $1 - \varepsilon$.

shown is the bound 7/6 from (1.1), which holds with probability 1. Theorem 3.2 begins to yield better information than (1.1) at $n = 28$ where the bound is 1.162, with probability at least 0.99. The bound improves to 1.029 at $n = 100$ with the same confidence. □

Example 2. Again consider the 99% confidence level, but let there be $m = 10$ processors. The bound (1.1) is now 1.3 with probability 1. Figure 2 plots $\beta_{10}(n, 0.01)$ against n for $n = 80(1)160$. Here the crossover with the deterministic bound occurs at $n = 89$, where the bound is 1.296 with probability at least 0.99. At $n = 150$ the bound has decreased to 1.159 with the same confidence. □

Example 3. To see how the tightness of bound (4.1) varies with confidence $1 - \varepsilon$, we show in Fig. 3 the probability of various bounding values for $m = 2$ processors and sample size $n = 40$. Figure 3 gives values of the bound $\beta_2(40, \varepsilon)$ on the abscissa. On the ordinate is the minimum probability $1 - \varepsilon$ of achieving this bound. □

The above examples show that Theorem 3.2 yields sharp information on the trade-off between performance bounds and the certainty about performance. Nothing in the proof of Theorem 3.2 made use of any special property of the *LPT* schedule, and the result holds for all list schedules. But in an *LPT* schedule, the last completed task will be short with high probability, and perhaps the inequality (3.8) could be improved. Thus it is possible that a more refined argument, based on the special properties of the *LPT* policy, will yield even better bounds for *LPT*, especially for large m. Bounds on the expected makespan for *LPT* appear in [Coff84], but bounds on the distributional behavior await discovery.

REFERENCES

[Bir51] Z. W. BIRNBAUM, AND F. H. TINGEY, *One-sided confidence contours for probability distribution functions*, Ann. Math. Statist., 22 (1951), pp. 592-596.

[Cof84] E. G. COFFMAN, JR., G. N. FREDERICKSON AND G. S. LUEKER, *A note on expected makespans for largest-first sequences of independent tasks on two processors*, Math. Oper. Res., 9, 2(1984), pp. 260-266.

[Cof85] E. G. COFFMAN, JR. AND E. N. GILBERT, *On the expected relative performance of list scheduling*, Oper. Res., 33 (1985), pp. 548-561.

[Con80] W. J. CONOVER, *Practical Nonparametric Statistics*, 2nd ed., Wiley, New York, 1980.

[Dur73] J. DURBIN, *Distribution Theory For Tests Based on the Sample Distribution Function*, Society for Industrial and Applied Mathematics, Philadelphia, 1973.

[Gar79] M. R. GAREY, AND D. S. JOHNSON, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, W. H. Freeman, San Francisco, 1979.

[Gra69] R. L. GRAHAM, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416-429.

[Gra76] ———, *Bounds on the performance of scheduling algorithms*, in Computer and Job-Shop Scheduling Theory, E. G. Coffman, ed., John Wiley, New York, 1976.

[Mil56] L. H. MILLER, *Table of percentage points of Kolmogorov statistics*, J. Amer. Statist. Assoc., 51, (1956), pp. 111-121.

[Owe62] D. B. OWEN, *Handbook of Statistical Tables*, Addison-Wesley, Reading, MA, 1962.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
007

MINIMAL REPRESENTATION OF DIRECTED HYPERGRAPHS*

G. AUSIELLO†, A. D'ATRI† AND D. SACCÀ‡

Abstract. In this paper the problem of minimal representations for particular classes of directed hypergraphs is analyzed. Various concepts of minimal representations of directed hypergraphs (called minimal equivalent hypergraphs) are introduced as extensions to the concepts of transitive reduction and minimum equivalent graph of directed graphs. In particular, we consider equivalent hypergraphs which are minimal with respect to all parameters which may be adopted to characterize a given hypergraph (number of hyperarcs, number of adjacency lists required for the representation, length of the overall description, etc.). The relationships among the various concepts of minimality are discussed and their computational properties are analyzed. In order to derive such results, a graph representation of hypergraphs is introduced.

AMS (MOS) subject classifications. 05C65, 68E10

Key words. algorithm, closure, computational complexity, (directed) graph, (directed) hypergraph, minimal equivalent hypergraph, minimum equivalent graph, NP-complete, transitive reduction

1. Introduction.

Hypergraphs [5] are a generalization of the concept of graph which has been extensively used for representing structures and concepts in several areas of computer science (see, for example, [6], [7], [9], [11], [15]). In this paper we consider a particular class of directed hypergraphs which are a natural generalization of directed graphs. In such hypergraphs the hyperarcs are directed from a nonempty set of nodes to a single node and they may be interpreted as a relationship between sets of objects and a single object. Such relationships may be often encountered in computer science. For example, in problem solving [12] the connection between a problem P and the set of problems whose solutions are needed to solve problem P is a relationship of this kind (usually represented by means of and-or graphs). Similarly in a Petri net [13] the marking of a place is determined by the simultaneous presence of marks in all places that are required to activate a transition. Finally the same situation arises in the representation of some data dependencies in relational data bases [14]. Given a set of attributes U, a set of functional dependencies is a relation over $\mathbf{P}(U) \times U$. A functional dependency from X to i (denoted $X \to i$) where $X \subseteq U$ and $i \in U$ means that, given the values of all attributes in X, the value of the attribute i is uniquely determined.

For example, if A, B, C, D, E are attribute names, the following set of functional dependencies:

$$\{A, B\} \to C, \quad \{A, B\} \to D, \quad \{B\} \to E, \quad \{E\} \to C$$

represents implication relationships between attribute values, that is the fact that a pair of values over A and B univocally determines the values over C and D, etc. Also in this case a directed hypergraph is the most immediate graphical representation for such a relationship.

In several applications of directed hypergraphs, analogously to what happens in the case of graphs, the following concepts assume an important role: the concept of

* Received by the editors January 18, 1983, and in final revised form January 24, 1985. This work was supported by CASMEZ under grant PS. 35-92/IND, and by MPI within the National Projects on Theory of Algorithms and Formal Aspects for Databases.

† Dipartimento di Informatica e Sistemistica, Università di Roma La Sapienza, Via Eudossiana 18, 00184 Roma, Italy.

‡ CRAI, Via Bernini 5, 87030 Rende, Italy.

traversal (i.e., hyperarc connection leading from a set of nodes to a single node), the concept of closure (i.e., representation of all connections over a hypergraph), the concept of equivalence (i.e., hypergraphs with the same closure) and finally the concept of minimal representation among equivalent hypergraphs. In the case of functional dependencies, for example, the concept of closure, equivalence and minimality have been investigated with the aim of providing the set of functional dependencies which is minimal among all sets equivalent to the given one according to particular minimality criteria [2], [10].

In this paper we introduce various concepts of minimality among equivalent directed hypergraphs, their relationships and their complexity properties. Minimal representations of hypergraphs are defined with respect to all parameters which may be adopted to characterize a hypergraph (number of hyperarcs, number of adjacency lists required for the representation, length of the overall description, etc.). Some results may be found in [2], [4], [10] due to the strict correspondence between directed hypergraphs and sets of functional dependencies. In this paper we extend such results in order to give a complete treatment of all concepts of minimality which are relevant not only in the context of functional dependencies but also in a more general environment.

A formulation based on directed labelled graphs (*FD*-graphs, previously introduced in [2] for the design of efficient algorithms for functional dependency manipulation) is used throughout this paper as a representation of hypergraphs in order to prove the desired results. This formalism is very convenient not only because it provides a ground for the unified treatment of the various results but also because it allows one to state more clearly such concepts as redundancy of arcs and nodes in hypergraphs and because it may be used to obtain economy in the description and computation of the hypergraph closure.

The paper is organized as follows. In § 2 we present the basic definition of hypergraphs, hypergraph closure, equivalent hypergraphs and their representation by means of *FD*-graphs. In § 3 we present various concepts of minimal equivalent hypergraphs and prove all relationships among such concepts. Finally, in § 4, we discuss the complexity of finding minimal equivalent hypergraphs.

2. Hypergraphs and their graphical representation. Various definitions of hypergraphs have been introduced in the literature. In particular, in [6] a very general concept of hypergraph (directed recursive labelnode hypergraph) has been defined; on the other hand, in [5] a simpler definition has been given which generalizes the definition of undirected graph by allowing edges to be composed by an arbitrary number of nodes. In this paper, we shall deal with hypergraphs which are generalizations of directed graphs.

A *directed hypergraph* $\mathbf{H}$ is a pair $\langle N, H\rangle$, where N is the set of *nodes* and H is the set of *hyperarcs*, and a hyperarc is an ordered pair (X, i) such that X is a nonempty subset of N and $i \in N$.

Given a directed hypergraph, we call *source set* a set of nodes that appears as the left side of at least one hyperarc.

From now on we shall refer to directed hypergraphs simply as hypergraphs.

Example 1. The hypergraph $\mathbf{H} = \langle \{A, B, C, D, E, F\}, \{(\{A, B\}, C), (\{B\}, D), (\{C, D\}, E), (\{C, D\}, F)\}\rangle$ is shown in Fig. 1, where hyperarcs are represented by arrows. The source sets of $\mathbf{H}$ are $\{A, B\}$, $\{B\}$, $\{C, D\}$. □

The basic parameters which will be taken into account in order to evaluate the algorithms presented in this paper will be the following: the number of nodes of the

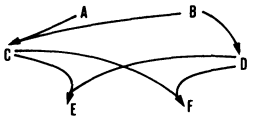


FIG. 1. *Directed hypergraph.*

hypergraph (n), the number of hyperarcs (m), the number of source sets (s), the *source area* (a), that is the sum of cardinalities of all source sets, and the overall length of description of the hypergraph ($|\mathbf{H}|$).

In the previous example we have:

$$n=6, \quad m=4, \quad s=3, \quad a=5.$$

As far as the length of the description is concerned, if we assume a representation based on adjacency lists (where for every source set the list of adjacent nodes is given) we have $|\mathbf{H}| \simeq a+m$. According to the same representation, the number of source sets corresponds to the number of adjacency lists.

In order to simplify the notation, here and in the following, nodes will be denoted by the first latin upper case letter $A, B, \cdots$ and sets of nodes will be expressed by concatenating the names of nodes (e.g., AB instead of $\{A, B\}$ and, in particular, A instead of $\{A\}$ when no ambiguity may arise). Besides, the last *latin* upper case letters $X, Y, \cdots, Z$ will be used to denote sets of nodes. In this case concatenation will stand for union (XY stands for $X \cup Y$) and the cardinality of X will be denoted by $|X|$.

In this paper we are mainly concerned in discussing the equivalence between hypergraphs, based on the concept of traversal and closure of hypergraphs.

Given a hypergraph $\mathbf{H}=\langle N, H\rangle$, the *closure* of $\mathbf{H}$, denoted by $\mathbf{H}^+$, is the hypergraph $\langle N, H^+\rangle$ such that (X, i) is in H^+ if one of the following conditions holds:

a) $(X, i) \in H$, or

b) $i \in X$ (*extended reflexivity*), or

c) there exists a set of nodes $Y=\{n_1, \cdots, n_q\}$ such that for each j, $1 \leqq j \leqq q$, (X, n_j) is a hyperarc in H^+, and (Y, i) is a hyperarc in H (*extended transitivity*).

Note that when X and Y are singletons, the extended reflexivity and transitivity rules coincide with the usual definitions of reflexivity and transitivity as given for graphs.

Example 2. Let us consider the hypergraph $\mathbf{H}$ of Fig. 1. Some of the hyperarcs in $\mathbf{H}^+$ are: (AB, C), (AB, A), (AB, E). □

Two hypergraphs are *equivalent* if they have the same closure.

The main problem we shall deal with throughout the paper is the problem of finding a minimal representation of a given hypergraph, that is an equivalent hypergraph which has fewer hyperarcs or some other kind of minimality property. Notice that the problem of finding a minimal representation in the case of hypergraphs is generally more complex than in the case of graphs because, while in the case of graphs the number of arcs in the closure is at most quadratic in the number of nodes, in the case of hypergraphs the number of hyperarcs in the closure is always exponential in the number of nodes.

In many cases, in order to deal with problems on hypergraphs, a graphical representation has been introduced (for instance, in [5] bipartite graphs are used to represent undirected hypergraphs). Similarly, in order to approach minimal equivalent hypergraph problems, we present a graphical representation of directed hypergraphs, which has been previously introduced in [2] for the manipulation of functional dependencies in relational databases.

Given a hypergraph $\mathbf{H}=\langle N, H\rangle$ the *FD-graph of* $\mathbf{H}$ is the labelled graph $G_H = \langle N_H, A_f, A_d\rangle$ where:

- $N_H = N \cup N_c$ is a set of *nodes*, where N will be called the set of *simple nodes* and $N_c = \{X | X$ is a source set in $\mathbf{H}$ such that $|X| > 1\}$ will be called the set of *compound nodes*;
- $A_f \subseteq N_H \times N$ is the set of arcs (labelled f) $\{(X, i)|$ there exists a hyperarc (X, i) in $H\}$, that will be called the set of *full arcs*;
- $A_d \subseteq N_c \times N$ is the set of arcs (labelled d) $\{(X, j)|$ for every $X \in N_c$ and $j \in X\}$ that will be called the set of *dotted arcs*.

Example 3. Let us consider the hypergraph of Fig. 1. Its *FD*-graph representation is given in Fig. 2. In this case the set of simple nodes is $N = \{A, B, C, D, E, F\}$ and the set of compound nodes is $N_c = \{AB, CD\}$. □

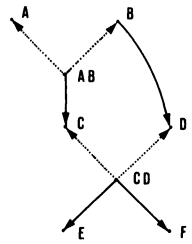


FIG. 2. *FD-graph of the hypergraph in Fig. 1.*

Given a hypergraph $\mathbf{H}$ with n nodes, m hyperarcs, s source sets, s' source singletons (source sets with cardinality equal to 1) and source area a, $\mathbf{H}$ will be represented by an *FD*-graph with n simple nodes, $n_c = s - s'$ compound nodes, m full arcs and $m_d = a - s'$ dotted arcs. If we consider the length of the description of the *FD*-graph, we may easily assume that it coincides with the length $|\mathbf{H}| \simeq a + m$ of the description of the given hypergraph.

The use of *FD*-graphs and of their closure in some cases allows us to determine minimal equivalent hypergraphs without falling into the exponential explosion of the hypergraph closure because, as it will be shown below, the *FD*-graph closure grows only at most quadratically. More precisely, in order to find a minimal representation $\mathbf{H}'$ of a hypergraph $\mathbf{H}$, we first give the *FD*-graph representation G_H of $\mathbf{H}$, then we determine the closure G_H^+ of G_H (instead of $\mathbf{H}^+$) and, finally, we use this closure to find a reduced representation $G_{H'}$ of G_H (that we shall call minimal covering) corresponding to $\mathbf{H}'$.

The sequence of transformations that we may go through in such cases is given in Fig. 3 (continuous line).

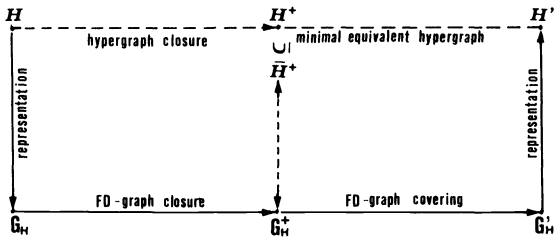


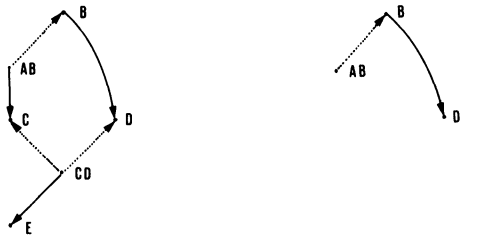
FIG. 3. *The sequence of transformations to determine minimal equivalent hypergraphs.*

Given an *FD*-graph $G_H = \langle N_H, A_f, A_d \rangle$ and a pair of nodes $i, j \in N_H$, an *FD-path* $\langle i, j \rangle$ from i to j is a minimal subgraph $\bar{G}_H = \langle \bar{N}_H, \bar{A}_f, \bar{A}_d \rangle$ of G_H such that $i, j \in \bar{N}_H$ and either $\bar{A}_f \cup \bar{A}_d = \{(i, j)\}$ or one of the following possibilities holds:

- j is a simple node and there exists a node k such that $(k, j) \in \bar{A}_f \cup \bar{A}_d$ and there is an *FD*-path $\langle i, k \rangle$ in $\bar{G}_H$ (*transitivity*);
- j is a compound node with component nodes $m_1, \cdots, m_q$ and $(j, m_1), \cdots, (j, m_q) \in \bar{A}_d$, and there are *FD*-paths $\langle i, m_k \rangle$ in $\bar{G}_H$, for $k = 1, \cdots, q$ and $m_k \neq i$ (*union*).

Furthermore an *FD*-path $\langle i, j \rangle$ is *dotted* if all arcs leaving i in the *FD*-path are dotted, otherwise it is *full.*

Example 4. In Fig. 4a a full *FD*-path and in Fig. 4b a dotted *FD*-path of the *FD*-graph of Fig. 2 are shown. □



a) *Full FD-path* $\langle AB, E \rangle$. b) *Dotted FD-path* $\langle AB, D \rangle$.

FIG. 4. *Some FD-paths of the FD-graph in Fig. 2.*

We observe that, due to the minimality requirement, a compound node without outgoing full arcs can only be either a source or a target node of *FD*-paths to which it belongs.

Given an *FD*-graph $G_H = \langle N_H, A_f, A_d \rangle$ we define *closure* of G_H the labelled directed graph $G_H^+ = \langle N_H, A_f^+, A_d^+ \rangle$ where an arc (i, j) is

- in A_d^+ iff there exists a dotted *FD*-path $\langle i, j \rangle$ in G_H;
- in A_f^+ iff $(i, j) \notin A_d^+$ and there exists a full *FD*-path $\langle i, j \rangle$ in G_H.

In [2] an algorithm is shown which, given an *FD*-graph G_H and a node i, provides the sets of nodes which may be reached from the node i by means of a full or dotted *FD*-path in time $O(m + m_d)$. This algorithm is an extension to *FD*-graphs of the usual transitive closure algorithm for graphs. The only substantial modification concerns the application of the union rule that is implemented by associating a counter with every compound node j. This counter keeps track of the number of component nodes of j which are currently reached from the source node i. By applying this algorithm to all the nodes, we may determine the closure of G_H in time $O((n + n_c) \times (m + m_d))$. In terms of the parameters of the hypergraph we have that the closure algorithm runs in time $O(s \times |\mathbf{H}|)$.

The closure of an *FD*-graph is a succinct representation of the closure of the corresponding hypergraph in the sense expressed in the following theorem.

THEOREM 1. *Let* $\mathbf{H} = \langle N, H \rangle$ *be a hypergraph and* $G_H = \langle N_H, A_f, A_d \rangle$ *the corresponding FD-graph. Given a pair of nodes* $i, j \in N_H$ *where* j *is a simple node and* $j \neq i$, *the arc* (i, j) *is in* G_H^+ *if and only if there exists a corresponding hyperarc in* $\mathbf{H}^+$.

Proof. Only if part. Since every arc in G_H^+ incident into a simple node is either in G_H or is derived by applying the transitivity and the union rules, it is easy to observe that the corresponding hyperarc is either in $\mathbf{H}$ or is derived in $\mathbf{H}^+$ by applying the extended transitivity and the reflexivity rules.

If part. Let the hyperarc (i, j) be in $\mathbf{H}^+$, where j is a single node. We prove by induction on the construction of H^+ that the arc (i, j) is in G_H^+. The following cases may arise (the first two cases are the basis of the induction):

- either (i, j) is a hyperarc of $\mathbf{H}$, then (i, j) is a full arc in G_H and, hence, in G_H^+;
- or (i, j) is a hyperarc of $\mathbf{H}^+$ obtained by reflexivity, then the dotted arc (i, j) appears in G_H and, hence, in G_H^+;
- or there exists a set of simple nodes $Y = \{n_1, \cdots, n_q\}$ such that for each k, $1 \leqq k \leqq q$, (i, n_k) is in H^+, and (Y, j) is in H. In this case, by inductive hypothesis there exist *FD*-paths $\langle i, n_k \rangle$ in G_H, for $k = 1, \cdots, m$ and $n_k \neq i$; then by union rule (or by transitivity rule if $q = 1$) there exists the *FD*-path $\langle i, j \rangle$ and, hence, (i, j) is in G_H^+. □

Notice that the hypergraph corresponding to G_H^+ is the hypergraph $\bar{\mathbf{H}}^+$ in Fig. 3 which is equivalent to $\mathbf{H}$.

3. Minimal equivalent hypergraphs. Several definitions of minimality in an equivalence class of hypergraphs may be introduced with respect to the various parameters we have introduced to characterize a hypergraph. Before defining minimal equivalent hypergraphs, we introduce the following concepts of redundancy.

Let $\mathbf{H} = \langle N, H \rangle$ be a hypergraph:

- a node j is *redundant in a source set* X of $\mathbf{H}$ if $j \in X$ and $(X - j, j) \in H^+$;
- a hyperarc (X, j) is *redundant* if (X, j) is in the closure of the hypergraph obtained from $\mathbf{H}$ by eliminating (X, j);
- $\mathbf{H}$ is *nonredundant* if it has neither redundant nodes in any source set nor redundant hyperarcs.

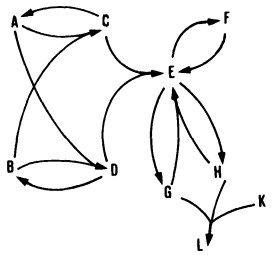
PROPOSITION 1. *Let* $\mathbf{H}$ *be a hypergraph. The hypergraph* $\mathbf{H}'$ *obtained from* $\mathbf{H}$ *by eliminating a redundant hyperarc or a redundant node in a source set is equivalent to* $\mathbf{H}$.

Proof. In case of elimination of a redundant hyperarc, the proof is trivial. Let us now assume that $\mathbf{H}$ has a redundant node j in the source set X. The hypergraph $\mathbf{H}'$ is obtained from $\mathbf{H}$ by replacing every hyperarc (X, i) in $\mathbf{H}$ with $(X - j, i)$. We have to prove that $\mathbf{H}$ is equivalent to $\mathbf{H}'$, thus $\mathbf{H}^+ = \mathbf{H}'^+$, where $\mathbf{H}^+$ and $\mathbf{H}'^+$ are the closures of $\mathbf{H}^+$ and $\mathbf{H}'^+$, respectively. To this end, it is sufficient to show that for each hyperarc (X, i) in $\mathbf{H}$, both (X, i) is in $\mathbf{H}'^+$ and $(X - j, i)$ is in $\mathbf{H}^+$ (in fact, $\mathbf{H}$ and $\mathbf{H}'$ only differ in such hyperarcs). Since $(X - j, i)$ is in $\mathbf{H}'$ by construction and for each r in $X - j$, (X, r) is in $\mathbf{H}'^+$ by extended reflexivity, the hyperarc (X, i) is in $\mathbf{H}'^+$ by extended transitivity. On the other hand, since (X, i) is in $\mathbf{H}$ by assumption, $(X - j, j)$ is in $\mathbf{H}^+$ by definition of redundant node, and for each r in $X - j$, $(X - j, r)$ is in $\mathbf{H}^+$ by extended reflexivity, the hyperarc $(X - j, i)$ is in $\mathbf{H}^+$ by extended transitivity. This concludes the proof. □

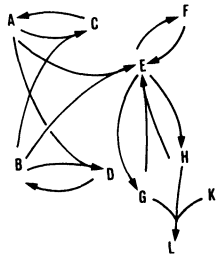
We now introduce several definitions of minimality for hypergraphs. Let $\mathbf{H}$ be a nonredundant hypergraph. We say that $\mathbf{H}$ is:

- *Source-minimum* (*SM*) if there exists no hypergraph equivalent to $\mathbf{H}$ with fewer source sets;
- *Hyperarc-minimum* (*HM*) if there exists no hypergraph equivalent to $\mathbf{H}$ with fewer hyperarcs;
- *Source-hyperarc-minimum* (*SHM*) if there exists no hypergraph $\mathbf{H}$ equivalent to $\mathbf{H}$ such that $\bar{s} + \bar{m} < s + m$, where s and m ($\bar{s}$ and $\bar{m}$) are the number of source sets and hyperarcs of $\mathbf{H}$ ($\mathbf{H}'$);
- *Source-area-minimum* (*SAM*) if there exists no hypergraph equivalent to $\mathbf{H}$ with smaller source *area*;
- *Optimum* (*O*) if there exists no hypergraph equivalent to $\mathbf{H}$ with smaller size (i.e., source area + number of hyperarcs).

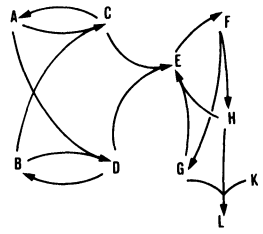
Example 5. In Fig. 5 we have: a) a nonredundant hypergraph **H**, b) an *SM*-hypergraph equivalent to **H** obtained from it by replacing the hyperarcs (CD, E) with the hyperarc (AB, E), c) a *HM*-hypergraph equivalent to **H** obtained from it by replacing the hyperarcs (F, E) (E, G), (E, H) with the hyperarcs (F, G), (F, H), d) an *SHM*-hypergraph equivalent to **H** obtained from it by combining the above replacements, e) an *SAM*-hypergraph equivalent to **H** obtained from the *SM*-hypergraph in b) by replacing the hyperarc (HGK, L) with (FK, L), f) an *O*-hypergraph equivalent to **H** obtained from the *SHM*-hypergraph in d) by replacing the hyperarc (HGK, L) with (FK, L). □



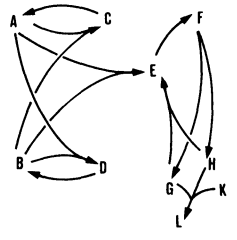
a) *Nonredundant hypergraph.*



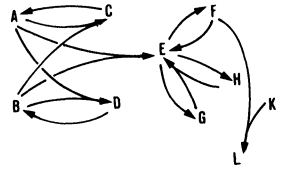
b) *SM hypergraph.*



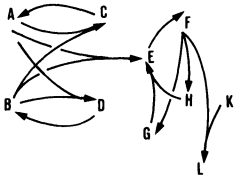
c) *HM hypergraph.*



d) *SHM hypergraph.*



e) *SAM hypergraph.*



f) *O hypergraph.*

FIG. 5. *Minimal equivalent hypergraphs.*

In this section we shall prove that all the above concepts of minimality are strongly related. Such relationships, summarized in Fig. 6 (where $A \Rightarrow B$ means that A-minimality implies B-minimality), imply that, given a hypergraph **H**, there exist hypergraphs equivalent to **H**, which are simultaneously minimal with respect to any combination of the criteria: number of hyperarcs, number of source sets, source area. In particular, it must be noted that an *O*-hypergraph equivalent to **H** can be found among the *SHM*-hypergraphs (or *SAM*-hypergraphs) equivalent to **H**, an *SHM*-hypergraph among the *HM*-hypergraphs (or *SM*-hypergraphs), and an *SAM*-hypergraph among *SM*-hypergraphs.

In order to prove the above relationships, we need some definitions and results for *FD*-graphs associated to nonredundant hypergraphs.

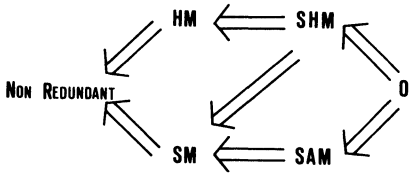


FIG. 6. *Relationships among concepts of minimality for hypergraphs.*

Let $G_H = \langle N_H, A_f, A_d \rangle$ be the *FD*-graph of the hypergraph **H**. We say that:

- a compound node k of G_H is *redundant* if for each full arc (k, j), there exists a dotted *FD*-path $\langle k, j \rangle$ in G_H;
- a dotted (full) arc (k, j) of G_H is *redundant* if there exists a dotted (full or dotted) *FD*-path $\langle k, j \rangle$, which does not contain the arc (k, j).
- a pair of nodes $i, j \in N_H$ are *equivalent* if there exist both *FD*-paths $\langle i, j \rangle$ and $\langle j, i \rangle$ in G_H;
- a compound node i of G_H is *superfluous* if there exists a dotted *FD*-path $\langle i, j \rangle$, where j is equivalent to i;
- G_H is *LR-minimum*[1] if it has neither redundant nodes and arcs nor superfluous nodes;
- $G_{H'}$ is a *covering*[1] of G_H if $\mathbf{H}'$ is equivalent to **H**, where $\mathbf{H}'$ is the hypergraph represented by $G_{H'}$.

We note that the definition of *FD*-graph covering given in [2] coincides with the definition given here because of the strict correspondence between directed hypergraphs and sets of functional dependencies in relational database theory. It follows that some of the results derived in [2] can be also used in this paper:

FACT 1 [2]. *Let G_H be an FD-graph.*

a) *Every FD-graph obtained from G_H by eliminating any redundant node together with all its outgoing arcs or any redundant arc is a covering of G_H.*

b) *Let i be a superfluous node in G_H and let j be a node equivalent to i such that there exists a dotted FD-path $\langle i, j \rangle$ in G_H. Let $G_{H'} = \langle N'_H, A'_f, A'_d \rangle$ be an FD-graph where:*

- $A'_f = (A_f \cup \{(j, k) | (i, k) \in A_f\}) \setminus \{(i, k) | (i, k) \in A_f\}$,
- $A'_d = A_d \setminus \{(i, k) | (i, k) \in A_d\}$,
- $N'_H = N_H \setminus \{i\}$.

Then $G_{H'}$ is a covering of G_H.

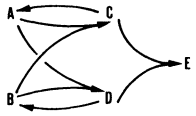
c) *Let $G_{H'}$ be a covering of G_H. Let $G^+_{H'}$, G^+_H be the closures of $G_{H'}$ and G_H, respectively. Let i, j be two nodes both in $G_{H'}$ and in G_H. If (i, j) is a full arc in G^+_H and a dotted arc in $G^+_{H'}$, then every dotted FD-path $\langle i, j \rangle$ in $G_{H'}$ contains a node k equivalent to i.*

From now on, by *elimination of a redundant node* in an *FD*-graph we shall also mean the elimination of all arcs leaving the redundant node (notice that, by definition of *FD*-graph, compound nodes do not have incoming arcs). Besides, by *elimination of a superfluous node* we shall mean the procedure indicated in Fact 1b.

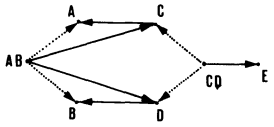
We point out that a covering $G_{H'}$ of an *FD*-graph G_H may differ from G_H not only in the set of arcs but also in the set of compound nodes; on the other hand, G_H and $G_{H'}$ have the same set of simple nodes.

Example 6. In the *FD*-graph of Fig. 7b (corresponding to the hypergraph of Fig. 7a), the nodes *AB* and *CD* are equivalent and the node *CD* is superfluous. The

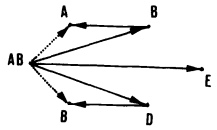
[1] The name is due to the properties of corresponding definitions given in [2] for *FD*-graphs representing functional dependencies.



a) *Hypergraph* **H**.



b) *FD-graph* G_H *of* **H**.



c) *LR-minimum covering of* G_H.

FIG. 7

FD-graph in Fig. 7c is an *LR*-minimum covering of the *FD*-graph in Fig. 7b and is obtained from it by eliminating the superfluous note *CD*. □

We now introduce a technical lemma which outlines a structural property of *LR*-minimum *FD*-graphs that will be used to derive the main results. We note that some of these properties (namely, part a) and b) of the lemma) were also proved in [2] but they will also be presented here in order to provide a ground for the proof of part c).

LEMMA 1. *Let* **H** *and* **H'** *be two equivalent hypergraphs and let* $G_H = \langle N_H, A_f, A_d \rangle$, $G_{H'} = \langle N_H, A'_f, A'_d \rangle$ *be their corresponding FD-graphs. If both* G_H *and* $G_{H'}$ *are LR-minimum then*

a) *there exists a bijection* ϕ: $\bar{N}_{H'} \leftrightarrow \bar{N}_H$, *where* $\bar{N}_H = N_H \setminus N_{H'}$ *and* $\bar{N}_{H'} = N_{H'} \setminus N_H$;

b) *for each* $i \in \bar{N}_{H'}$, $\phi(i)$ *is equivalent to* i *in* $G_{H''} = \langle N_H \cup N_{H'}, A_f, A_d \cup A'_d \rangle$;

c) *for each* $i \in \bar{N}_{H'}$, *there exists a dotted FD-path* $\langle \phi(i), i \rangle$ *in* $G_{H''}$.

Remark. All nodes in $\bar{N}_H$ and in $\bar{N}_{H'}$ are compound since G_H and $G_{H'}$ have the same set of simple nodes.

Proof. Let $G_H^+ = \langle N_H, A_f^+, A_d^+ \rangle$ and $G_{H'}^+ = \langle N_{H'}, A_f'^+, A_d'^+ \rangle$ be the closure of G_H and $G_{H'}$, respectively.

a) We define the bijection ϕ by associating to each node i in $N_{H'}$ a node $j = \phi(i)$ in $\bar{N}_H$ as follows. Let us construct the *FD*-graph $G_{\bar{H}}$ by adding i and its outgoing dotted arcs to G_H (recall that i is a compound node). Let $G_{\bar{H}}^+ = \langle N_{\bar{H}}, \bar{A}_f^+, \bar{A}_d^+ \rangle$ be the closure of $G_{\bar{H}}$. By Fact 1a, $G_{\bar{H}}$ is a covering of G_H and, then, of $G_{H'}$. Since i is nonredundant in $G_{H'}$ by hypothesis, there exists at least one simple node r such that (i, r) is in A'_f and then in $A_f'^+$. In addition, since $G_{\bar{H}}$ is a covering of $G_{H'}$ and since all arcs leaving i in $G_{\bar{H}}$ are dotted, the arc (i, r) is in $\bar{A}_d^+$. Hence, by Fact 1c, every dotted *FD*-path $\langle i, r \rangle$ in $G_{\bar{H}}$ contains at least a node k equivalent to i. Let $j = \phi(i)$ be a node in N_H equivalent to i and such that there is a dotted *FD*-path $\langle i, j \rangle$ in $G_{\bar{H}}$ that does not contain any other node equivalent to i. Such a node obviously exists. We now show that j is in $\bar{N}_H$ by contradiction. In fact, if j was also in $N_{H'}$, by Fact 1c the arc (i, j) would be in $A_d'^+$, since there is a dotted *FD*-path $\langle i, j \rangle$ in $G_{\bar{H}}$ which does not contain any other node equivalent to i. Hence, i would be superfluous in $G_{H'}$ (contradiction with the hypothesis that $G_{H'}$ is *LR*-minimum). We now prove that ϕ is bijective by showing that ϕ is injective and that $|\bar{N}_{H'}| = |\bar{N}_H|$. Let $\bar{i}$ be a node in $\bar{N}_{H'}$ different from i. Let us suppose, by contradiction, that $\phi(\bar{i}) = \phi(i) = j$. Hence, i and $\bar{i}$ are equivalent in $G_{H'}$. Let $G_{\bar{H}'}$ be the *FD*-graph obtained from $G_{H'}$ by adding the node j and all its outgoing dotted arcs (notice that j is compound since it is in $N_{\bar{H}}$) and let $G_{\bar{H}'}^+ = \langle N_{\bar{H}'}, \bar{A}_f'^+, \bar{A}_d'^+ \rangle$ be the closure of $G_{\bar{H}'}$. Since j is equivalent to i and

$\bar{i}$ in $G_{\bar{H}'}$ and since all arcs leaving j in $G_{\bar{H}'}$ are dotted, (j, i) and $(j, \bar{i})$ are in $\bar{A}'^{+}_{d}$. Furthermore, since the dotted *FD*-paths $\langle i, j\rangle$ and $\langle \bar{i}, j\rangle$ in $G_{\bar{H}}$ do not contain other nodes equivalent to i or $\bar{i}$ and since $G_{\bar{H}}$ is a covering of $G_{\bar{H}'}$, by Fact 1c there exist also dotted *FD*-paths $\langle i, j\rangle$ and $\langle \bar{i}, j\rangle$ in $G_{\bar{H}'}$. Now, without loss of generality, we suppose that the dotted *FD*-path $\langle j, i\rangle$ in $G_{\bar{H}'}$ does not contain the node $\bar{i}$ (otherwise we could refer to $\langle j, \bar{i}\rangle$). Since in $G_{\bar{H}'}$ there exist the dotted *FD*-paths $\langle \bar{i}, j\rangle$ and $\langle j, i\rangle$ and since $\langle j, i\rangle$ does not contain the node $\bar{i}$, there also exists a dotted *FD*-path $\langle \bar{i}, i\rangle$ in $G_{\bar{H}'}$ (see the definition of *FD*-path). Furthermore, this *FD*-path does not contain the node j since all arcs leaving j in $G_{\bar{H}'}$ are dotted. Hence, the dotted *FD*-path $\langle \bar{i}, i\rangle$ is also in $G_{H'}$ because $G_{\bar{H}'}$ differs from $G_{H'}$ only in the compound node j. It follows that $\bar{i}$ is superfluous in $G_{H'}$ and we get a contradiction with the hypothesis that $G_{H'}$ is *LR*-minimum. Therefore, ϕ is injective and, hence, $|\bar{N}_{H'}| \leqq |\bar{N}_{H}|$. If we exchange G_H with $G_{H'}$ and conversely in our argument, we also obtain $|\bar{N}_{H}| \leqq |\bar{N}_{H'}|$. It follows that $|\bar{N}_{H}| = |\bar{N}_{H'}|$ and ϕ is a bijection.

b) Let us consider the *FD*-graph $G_{H''}$ defined in the statement of the lemma. We observe that since i and $j = \phi(i)$ are equivalent in $G_{\bar{H}}$ and since $G_{\bar{H}}$ is a covering of $G_{H''}$ by Fact 1a, i and j are equivalent in $G_{H''}$ by Theorem 1.

c) Finally we prove that there is a dotted *FD*-path $\langle j, i\rangle$ in $G_{H''}$, where $j = \phi(i)$. First of all we recall that there is a dotted *FD*-path $\langle j, i\rangle$ in $G_{\bar{H}'}$. We now show that this *FD*-path does not contain any node equivalent to j by contradiction. Let us suppose that there exists a node i' equivalent to j in $\langle j, i\rangle$. Without loss of generality, we can suppose that the dotted *FD*-path $\langle j, i'\rangle$ does not contain any node equivalent to j. Since there is also a dotted *FD*-path $\langle i, j\rangle$ in $G_{\bar{H}'}$ (as we have proved before), there exists a dotted *FD*-path $\langle i, i'\rangle$ in $G_{\bar{H}'}$. This *FD*-path is also in $G_{H'}$ since $G_{\bar{H}'}$ differs from $G_{H'}$ only in the compound node j that has no outgoing full arcs. It follows that i is superfluous (contradiction with the hypothesis that $G_{H'}$ is *LR*-minimum). Therefore the dotted *FD*-path $\langle j, i\rangle$ in $G_{\bar{H}'}$ does not contain any node equivalent to j. Hence, since $G_{\bar{H}'}$ is a covering of $G_{H''}$, by Fact 1c there is a dotted *FD*-path $\langle j, i\rangle$ in $G_{H''}$ and this concludes the proof. □

The next proposition establishes the correspondence between *LR*-minimum *FD*-graph and source-minimum hypergraph. This result is useful both for proving the relationships among the various definitions of minimality for hypergraphs and for applying the computational results proven in [2] for *LR*-minimum *FD*-graphs to source minimum hypergraphs.

PROPOSITION 2. *Let* $\dot{\mathbf{H}}$ *be a hypergraph and let* G_H *be the FD-graph of* $\mathbf{H}$.

a) $\mathbf{H}$ *is nonredundant iff* G_H *has neither redundant nodes nor redundant arcs.*

b) $\mathbf{H}$ *is source minimum iff* G_H *is LR-minimum.*

Proof. a) By Theorem 1 and the definition of *FD*-graph, it is easy to see that a hyperarc (X, i) in $\mathbf{H}$ is redundant in $\mathbf{H}$ if and only if the full arc (X, i) is redundant in $G_{\mathbf{H}}$ (and, in particular, X is a redundant node in G_H if and only if all hyperarcs leaving the source set X in $\mathbf{H}$ are redundant) and that a node j is redundant in a source set X of $\mathbf{H}$ if and only if the dotted arc (X, j) is redundant in $G_{\mathbf{H}}$.

b) *Only if part.* Let $\mathbf{H}$ be *SM*. By part a) of the proposition, G_H has neither redundant nodes nor redundant arcs. In addition, G_H has no superfluous nodes, because otherwise we could find a hypergraph equivalent to $\mathbf{H}$ with fewer source sets by eliminating the superfluous nodes (by Fact 1b). Hence G_H is *LR*-minimum.

If part. Let G_H be *LR*-minimum. Let $\mathbf{H}'$ be an *SM*-hypergraph equivalent to $\mathbf{H}$ and let $G_{H'}$ be the corresponding *FD*-graph. By the only if part of this proposition $G_{H'}$ is *LR*-minimum. Hence, by Lemma 1, G_H and $G_{H'}$ have the same set of simple nodes and the same number of compound nodes. Furthermore, by Theorem 1 the

outdegree of a simple node i in G_H is equal to 0 if and only if the outdegree of i is 0 in $G_{H'}$. Hence, G_H and in $G_{H'}$ have the same number of nodes with outdegree > 0. This means that **H** and **H'** have the same number of source sets, i.e., **H** is *SM*. □

We are now ready to prove the results concerning implications among minimality concepts (see Fig. 6).

THEOREM 2. *Let* **H** *be a hypergraph.*

a) *If* **H** *is SHM then* **H** *is also SM.*

b) *If* **H** *is SHM then* **H** *is also HM.*

c) *If* **H** *is SAM then* **H** *is also SM.*

d) *If* **H** *is O then* **H** *is also SHM.*

e) *If* **H** *is O then* **H** *is also SAM.*

Proof. Let G_H be the *FD*-graph of **H**.

a) (*SHM* $\Rightarrow$ *SM*). Let us proceed by contradiction by supposing that **H** is *SHM* but not *SM*. By Proposition 2b, G_H is not *LR*-minimum, and, by Proposition 2a, since **H** is nonredundant, G_H has neither redundant nodes nor redundant arcs. Hence G_H has at least one superfluous node. By eliminating such a node, by Fact 1b, we determine a covering $G_{H'}$ of G_H with fewer nodes. Hence the corresponding hypergraph **H'** is equivalent to **H** and has the same number of hyperarcs as **H** but a smaller number of source sets (contradiction with the fact that **H** is *SHM*).

b) (*SHM* $\Rightarrow$ *HM*). Let us proceed by contradiction by supposing that **H** is *SHM* but not *HM*. Then there exists a hypergraph **H'** equivalent to **H** with fewer hyperarcs and more source sets than **H**, since **H** is *SM* by part a) of the theorem. Since **H'** is not *SM*, by Proposition 2b, $G_{H'}$ is not *LR*-minimum. Furthermore, since $G_{H'}$ has no redundant arcs, $G_{H'}$ has at least one superfluous node. Let $G_{H''}$ be the *FD*-graph obtained from $G_{H'}$ by eliminating all superfluous nodes. By definition and by Fact 1b, $G_{H''}$ is *LR*-minimum and is a covering of G_H. Hence, by Proposition 2b, **H''** is *SM* and equivalent to **H** and has less hyperarcs than **H** (contradiction).

c) (*SAM* $\Rightarrow$ *SM*). Since **H** is nonredundant by definition of *SAM* hypergraph, by Proposition 2a, G_H has neither redundant nodes nor redundant arcs. Moreover, G_H has no superfluous nodes because otherwise, by eliminating such nodes, by Fact 1b, the so-obtained *FD*-graph would be the representation of an equivalent hypergraph with a smaller source area than **H** (contradiction). Hence G_H is *LR*-minimum and, by Proposition 2b, **H** is *SM*.

d) (*O* $\Rightarrow$ *SAM*). Let us proceed by contradiction by supposing that **H** is optimal, but that there exists an equivalent hypergraph **H'** with a smaller source area (and, then, more hyperarcs) than **H**. Let us consider the *FD*-graphs $G_H = \langle N_H, A_f, A_d \rangle$ and $G_{H'} = \langle N_{H'}, A'_f, A'_d \rangle$ associated to **H** and **H'**, respectively. We construct the *FD*-graph $G_{H''} = \langle N_H \cup N_{H'}, A_f, A'_d \cup A_d \rangle$. By Fact 1a, $G_{H''}$ is a covering of G_H and $G_{H'}$. Furthermore, both G_H and $G_{H'}$ are obviously *LR*-minimum. Hence, by Lemma 1 c) every compound node in $N_H \setminus N_{H'}$ is superfluous in $G_{H''}$. By eliminating such superfluous nodes, we obtain an *FD*-graph with the same number of full arcs as G_H and the same set of nodes as $G_{H'}$. By Fact 1b, this *FD*-graph is a covering of $G_{H''}$ and, then, of G_H; therefore, the hypergraph represented by this *FD*-graph has the same number of hyperarcs as **H** but smaller source area (contradiction).

e) (*O* $\Rightarrow$ *SHM*). Once again we proceed by contradiction by supposing that **H** is optimal, but that there exists an equivalent hypergraph **H'** with less hyperarcs (and, then, a larger source area) than **H**. By repeating the same argument used in the proof of part d) of the theorem, we would find a hypergraph with the same source area as **H** but less hyperarcs (contradiction). □

Remark. None of the other implications among concepts of hypergraph minimality hold (except those obtained by transitivity).

Example 7. Let us again consider the equivalent hypergraphs in Fig. 5. We note that: a) the *SM* (*HM*) hypergraph in Fig. 5b (in Fig. 5c) is not *HM* (*SM*), b) the *SHM* (*SAM*) hypergraph in Fig. 5d (in Fig. 5e) is not *SAM* (*SHM* or *HM*). □

4. Finding minimal equivalent hypergraphs. In this section, we discuss the complexity of finding minimal representations of hypergraphs, thus hypergraphs which are equivalent to a given one and have some of the minimality properties defined in the preceding section.

As we have already observed, the problem of determining minimal equivalent hypergraphs is more complex than in the case of graphs, essentially because the closure of a hypergraph has an exponential number of hyperarcs and because, in the case of hypergraphs, we can define minimality with respect to different parameters. In fact, in the case of graphs, the only concept of minimality is the minimality with respect to the number of arcs (*transitive reduction*).

The first results concerning the complexity of determining minimal equivalent hypergraphs may be derived as immediate consequences of results obtained for sets of functional dependencies. The first result concerns the complexity of determining an *SM* hypergraph.

PROPOSITION 3. *Given a hypergraph* $\mathbf{H}=\langle N, H\rangle$, *finding an SM-hypergraph equivalent to* **H** *can be done in* $O(|\mathbf{H}|^2)$.

Proof. An *SM*-hypergraph equivalent to **H** can be obtained by the following steps:

1. determine the *FD*-graph G_H of **H**;
2. eliminate redundant nodes;
3. eliminate superfluous nodes;
4. eliminate redundant arcs;
5. derive the hypergraph $\mathbf{H}'$ corresponding to the reduced *FD*-graph $G_{H'}$.

Steps 2–4 correspond to the algorithm presented in [2] for finding an *LR*-minimum covering of an *FD*-graph in time quadratic in the size of G_H. Hence, considering that the size of G_H is equal to the size of **H** and that steps 1 and 5 can be obviously performed in linear time, the result is proved. □

We point out that also a nonredundant representation of a hypergraph **H** can be found in time quadratic in $|\mathbf{H}|$. In fact, to this end, by Proposition 1, it is enough to perform all steps of the algorithm in the proof of Proposition 3 but step 3.

A second result derived from database theory concerns the complexity of finding an *O* equivalent hypergraph.

PROPOSITION 4. *Given a hypergraph* $\mathbf{H}=\langle N, H\rangle$ *and an integer* k, *the problem of deciding whether there exists a hypergraph* $\mathbf{H}'$ *equivalent to* **H** *with* $a'+m'\leqq k$, *where* a' *and* m' *are respectively the source area and the number of hyperarcs of* H', *is* NP-*complete.*

Proof. See the result in [10] concerning the complexity of determining an optimum covering of a set of functional dependencies.[2] □

If we now take into consideration Fig. 6, we may note that a characterization of the complexity of all other minimality criteria is needed. The next results provide a more precise borderline between tractable and intractable in this domain. To this end, considering the relationships among all minimality concepts stated in Theorem 2, it is

[2] In [10] it has also been proved that an optimum covering of functional dependencies is also *LR*-minimum (i.e., an *O* hypergraph is also *SM*). Actually, this implication is also a direct consequence of Theorem 2.

sufficient to prove the intractability for the cases of *HM* and *SAM* equivalent hypergraphs (actually, Proposition 4 may be seen as a direct consequence of the next results).

Let us first of all consider the problem of finding a minimal equivalent hypergraph with the minimum number of hyperarcs (*HM*). This problem corresponds to the transitive reduction of a graph. From the computational point of view it is interesting to observe that the complexity of this problem increases dramatically when we go from graphs to hypergraphs. In fact, given a graph $G=\langle N, A\rangle$, the problem of finding the transitive reduction may be solved in polynomial time $O(|N| \cdot |A|)$ [1]. Instead, in the case of hypergraphs, we shall prove the following theorem.

THEOREM 3. *Let* **H** *be a hypergraph and let k be a positive integer. The problem of whether there exists a hypergraph* **H'** *such that* **H'** *is equivalent to* **H** *and has no more than k hyperarcs is* NP-*complete.*

Proof. Since the problem is clearly in NP, we only have to prove its NP-hardness. To this end, we give a polynomial reduction from the set-covering problem [8] to the given problem. Let an instance of the set covering problem be given: let $S=\{s_1, \cdots, s_n\}$ be a set of elements and $S_1, \cdots, S_m$ be a family of subsets of S such that $\mathbf{U}_{i=1,\cdots,m} S_i = S$. The set-covering problem is the problem of deciding whether there exists a sub-family of at most h sets $S_{i1}, \cdots, S_{ih}$ such that $\mathbf{U}_{j=1,\cdots,h} S_{ij} = S$. Given the above instance, we may construct a hypergraph **H** whose nodes are $\bar{s}_1, \cdots, \bar{s}_n, \bar{S}_1, \cdots, \bar{S}_m, \bar{T}$ and for every $s_j \in S_i$ there is a corresponding hyperarc $(\bar{S}_i, \bar{s}_j)$; besides, for each i, $1 \leqq i \leqq m$, there are the hyperarc $(\{\bar{s}_1, \cdots, \bar{s}_n\}, \bar{S}_i)$ and the hyperarc $(\bar{T}, \bar{S}_i)$ (see Fig. 8). The problem consists in deciding whether there exists a hypergraph **H'** equivalent to **H** with a number of hyperarcs less or equal to $k=\sum_{i=1,\cdots,m}(|S_i|)+m+h$. Note that if we take out from the hypergraph the node $\bar{T}$ and the hyperarcs leaving it the remaining hypergraph is nonredundant and no equivalent hypergraph with a smaller number of hyperarcs may exist. Hence the only redundant hyperarcs may be those leaving $\bar{T}$. In fact, if the sets $S_{i1}, \cdots, S_{ih}$ provide a covering of S, all arcs leaving $\bar{T}$ and different from $(\bar{T}, \bar{S}_{i1}), \cdots, (\bar{T}, \bar{S}_{ih})$ are redundant and may be eliminated without changing the closure. Since the reduction from the instance of set covering to the instance of *HM* equivalent hypergraph problem can be obviously done in polynomial time, the theorem is proved. □

Let us now consider the problem of finding a minimal equivalent hypergraph with the minimum source area (*SAM*). This problem has no meaning in the case of graphs since the fact that the outdegree of a node is zero or one is invariant in all equivalent graphs.

THEOREM 4. *Let* **H** *be a hypergraph and let k be a positive integer. The problem of whether there exists a hypergraph* **H'** *such that* **H'** *is equivalent to* **H** *and has a source area less than k* is NP-*complete.*

Proof. Also in this case the problem is clearly in NP. In order to prove that it is NP-hard, we use a slight modification of the proof of Theorem 3. Let us again consider the hypergraph in Fig. 8. We construct a new hypergraph **H** as follows. First we add

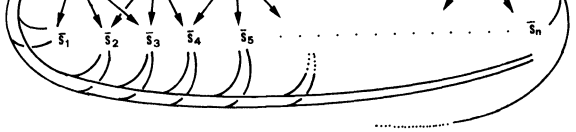


FIG. 8. *Hypergraph associated with an instance of the set-covering problem.*

a new node $\bar{T}_1$ and then we replace the hyperarcs $(\bar{T}, \bar{S}_1), \cdots, (\bar{T}, \bar{S}_m)$ with the hyperarc $(\bar{S}_1 \cdots \bar{S}_m \bar{T}_1, \bar{T})$. This latter hyperarc may contain redundant nodes. If we eliminate such nodes, by Proposition 2b, we obtain an *SM* equivalent hypergraph of **H**, since its *FD*-graph representation G_H is *LR*-minimum. In fact, in G_H neither nodes nor arcs are redundant and no node is superfluous because there are no equivalent nodes. Notice that if we did not add the node $\bar{T}_1$ in G_H, the node $\bar{S}_1 \cdots \bar{S}_m$ would have been superfluous with respect to the node $\bar{s}_1 \cdots \bar{s}_n$. The instance of the set-covering problem given in the proof of Theorem 3 has a solution (i.e., it admits a covering of no more than h sets) if and only if there exists a hypergraph which is equivalent to **H** and has a source area not larger than $k = n + m + 1 + h$. □

We conclude by mentioning that in case of minimal equivalent subhypergraphs of a given hypergraph **H** (i.e., HM-hypergraphs obtained from **H** by deleting some hyperarcs without modifying the closure), the problem is still intractable because of the NP-completeness of the minimal equivalent subgraph problem [8]. On the other hand, a nonredundant equivalent subhypergraph can be easily found in quadratic time by iteratively deleting redundant nodes and arcs in the associated *FD*-graph representation. Finally, we note that a stronger concept of directed hypergraph equivalence has been studied in [3].

REFERENCES

[1] A. V. AHO, M. R. GAREY AND J. D. ULLMAN, *The transitive reduction of a directed graph*, this Journal, 1 (1972), pp. 131-137.

[2] G. AUSIELLO, A. D'ATRI AND D. SACCÀ, *Graph algorithms for functional dependency manipulation*, J. Assoc. Comput. Mach., 30 (1983), pp. 752-766.

[3] ———, *Strongly equivalent directed hypergraphs*, in Analysis and Design of Algorithms for Combinatorial Problems, G. Ausiello and M. Lucertini, eds., Annals of Discrete Mathematics, 25 (1985), pp. 1-26.

[4] C. BEERI AND P. A. BERNSTEIN, *Computational problems related to the design of normal form relational schemes*, ACM TODS, 4 (1979), pp. 30-59.

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

[6] H. BOLEY, *Directed recursive labelnode hypergraphs: a new representation language*, Artificial Intelligence, 9 (1977), pp. 49-85.

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

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

[9] S. GNESI, U. MONTANARI AND A. MARTELLI, *Dynamic programming as graph searching: an algebraic approach*, J. Assoc. Comput. Mach., 28 (1981), pp. 737-751.

[10] D. MAIER, *Minimum covers in the relational data base model*, J. Assoc. Comput. Mach., 27 (1980), pp. 664-674.

[11] D. MAIER AND J. D. ULLMAN, *Connections in acyclic hypergraphs*, Proc. 1st ACM Symposium on Principles of Data Base Systems, Los Angeles, Ca., 1982, pp. 34-39.

[12] N. J. NILSSON, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

[13] C. A. PETRI, *Communication with automata*, Tech. Rep. RADC-TR-65377, Vol 1, Griffith Air Force Base, New York, 1966.

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

[15] M. YANNAKAKIS, *A theory of safe locking policies in Database Systems*, J. Assoc. Comput. Mach., 29 (1982), pp. 718-740.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
008

REPRESENTATIONS AND PARALLEL COMPUTATIONS FOR RATIONAL FUNCTIONS*

JOACHIM VON ZUR GATHEN†

Abstract. Representations of univariate rational functions over a given base of polynomials are considered, and a fast parallel algorithm for converting from one base representation to another is given. Special cases of this conversion include the following symbolic manipulation problems: Taylor expansion, partial fraction decomposition, Chinese remainder algorithm, elementary symmetric functions, Padé approximation, and various interpolation problems. If n is the input size, then all algorithms run in parallel time $O(\log^2 n)$ and use $n^{O(1)}$ processors. They work over an arbitrary field.

Key words. parallel processing, algebraic computing, symbolic manipulation, interpolation, Chinese remainder algorithm, Padé approximation

1. Introduction. This work forms part of an endeavour to understand the power of parallelism in symbolic computation: for which problems in algebraic manipulation with a polynomial-time sequential solution do fast parallel algorithms exist? Answers to this question may help to understand the power of parallelism in general: one may compare the power of different models of concurrent computation by testing on which models these algorithms can be implemented.

In this paper we present fast parallel algorithms for various problems in algebraic computation. It soon became apparent that the algorithms for all the problems considered here would follow the same pattern, namely conversion between different representations of the given rational function. So we start by introducing in § 2 the notion of representations of rational functions in a given "base" of polynomials. This notion encompasses several ways of representing rational functions, which are especially familiar if the rational function is a polynomial: the sequence of coefficients, a list of values, Taylor series, and a general list of values in "Hermite format". It turns out that if numerator and denominator degrees are correctly specified, then usually (but not always) such representations also determine a rational function uniquely.

We describe in § 3 two fast parallel algorithms that convert the standard coefficient representation of a rational function into a "base representation", and vice versa. Combining them we get an algorithm that converts the representation of a rational function in one base into that in another base. Section 4 discusses the existence question for representations. This turns out to be slightly less straightforward than one might expect, and is illustrated by the well-known fact that the rational functions required in Padé approximation or rational interpolation may fail to exist.

Section 5 presents as application of the general conversion algorithms fast parallel methods for the following problems in symbolic manipulation: Taylor expansion, partial fraction decomposition, Chinese remainder algorithm, elementary symmetric functions, Padé approximation, and various interpolation problems. As our model of parallel computation we can take an arithmetic network, which is a directed acyclic graph such that each edge either carries arithmetic values from the ground field or Boolean values, and with the following nodes: arithmetic operations ($+, -, \times, /$, fetch-

* Received by the editors June 28, 1983, and in final revised form December 10, 1984. An abstract of this paper appeared in the Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, Tucson, Arizona, 1983, pp. 133–137. This work was partially supported by Natural Sciences and Engineering Research Council of Canada grant 3-650-126-40 and Schweizerischer Nationalfonds grant 2175-0.83.

† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4.

ing a constant), tests ($a = 0$?) of an arithmetic value a, Boolean operations, and selection of one of two arithmetic inputs according to the value of a further Boolean input. The algorithms can also be implemented on an algebraic PRAM. The networks all have depth (= parallel time) $O(\log^2 n)$ and size (= number of processors) $n^{O(1)}$, where n is the input size. They work over an arbitrary ground field.

The representations discussed so far are sufficient to solve the computational problems of § 5. But their theoretical shortcomings warrant a generalization, so that in § 6 "Laurent representations" are introduced. They have the desired feature that rational functions now always have a unique representation.

The dual relationship between evaluation at many points and interpolation has been observed for a long time; also the fact that both computational problems consist in converting the representation of a polynomial from one format to another; see e.g. Strassen [1974]. However, one usually employs two quite different-looking sequential algorithms to solve the two problems. Besides the unification resulting from our approach even for polynomials, the fact of including rational functions into this framework seems to be even more interesting from a computational point of view, making bedfellows of such distinct-looking problems as Hermite interpolation and Padé approximation. We leave as an open question how well this approach translates into the sequential setting.

2. Representations of rational functions. Let F be an arbitrary field. A sequence $B = (b_1, \cdots, b_p)$ of pairwise relatively prime polynomials $b_1, \cdots, b_p \in F[x] \setminus F$ is called a base. A sequence $N = (n_1, \cdots, n_p) \in \mathbf{N}^p$ with $n_i \geqq 1$ is called a precision (for B), and $n = \sum_{1 \leqq i \leqq p} n_i \deg b_i$ is the total precision of (B, N). A sequence

$$r = (r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1})$$

such that for all i, j with $1 \leqq i \leqq p$ and $0 \leqq j < n_i$

$$r_{ij} \in F[x],$$
$$\deg r_{ij} < \deg b_i,$$

is called a *representative* in base B with precision N, or (B, N)-representative for short. $R(B, N)$ is the set of all (B, N)-representatives. $R(B, N)$ can be identified with F^n.

DEFINITION 2.1. Let $r \in R(B, N)$, and $g, h \in F[x]$ with $h \neq 0$. Then r is called a *weak (B, N)-representation of* (g, h) if for $1 \leqq i \leqq p$ we have

$$g \equiv h \sum_{0 \leqq j < n_i} r_{ij} b_i^j \mod b_i^{n_i}.$$

r is called a *(B, N)-representation of* $f = g/h \in F(x)$ if in addition

$$\gcd(b_1 \cdots b_p, h) = 1. \qquad \square$$

In other words, if we set

$$r_i = \sum_{0 \leqq j < n_i} r_{ij} b_i^j,$$

then $f \equiv r_i \mod b_i^{n_i}$ is given with "precision $n_i \deg b_i$", and r_i is developed according to powers of b_i.

One verifies that for given B, N, f, r the property "r is a (B, N)-representation of f" does not depend on the choice of polynomials g, h such that $f = g/h$ and

$\gcd(b_1 \cdots b_p, h) = 1$. Ideally, we would like our representations to have two properties:

Representability:	Given (B, N), every $f \in F(x)$ has a unique (B, N)-representation.
Convertibility:	There exists a fast parallel algorithm for converting representations from one base into another base.

As they stand, both properties fail to hold; but fortunately this failure is only marginal. Representability is discussed in § 4 and Remark 5.4, and it turns out that almost all rational functions have a unique representation. The second conversion algorithm of § 3 may return a pair of polynomials of which the input representation is only a "weak representation"; this is the reason for introducing the latter. Again, for almost all inputs this weakness does not appear. Note that it does not make sense to speak of weak representations of a rational function, since the above congruence alone would allow any (B, N)-representative r to represent any rational function $f = g/h \in F(x)$, by writing $f = (bg)/(bh)$ with $b = b_1^{n_1} \cdots b_p^{n_p}$. The gcd condition rules these unwanted representations out.

In § 6, we introduce the more general "Laurent representations". These then have both desired properties, and thus are the appropriate objects for this theory. Though inappropriate from a theoretical point of view, there are two reasons for introducing the representations and weak representations as above: Firstly, they are easier to define and work with, and secondly, they are sufficient to discuss the applications in § 5.

If $N = (n_1, \cdots, n_p)$ and $M = (m_1, \cdots, m_p)$ are two precisions for B, $m_i \leqq n_i$ for all i, and $r \in R(B, N)$ is a representation of f, then one obtains a representation in $R(B, M)$ of f by truncating the r_{ij}'s with $j \geqq m_i$.

Before proceeding with the general development, let us look at a few familiar representations that fit into this framework. As examples we will use $n = 5$ and the two rational functions

$$f_1 = x^4 - x^3 + 2x^2 - 3x - 2 \in F[x], \qquad f_2 = \frac{-7x^2 + x + 2}{x^2 + x - 1} \in F(x).$$

When B and N are given, we will write $r = \rho(f)$ if r is a (B, N)-representation of f.

1. *Sequence of coefficients.* In this most familiar of all representations we have $p = 1$, $B = (x)$ and $N = (n)$. For a polynomial $f = \sum_{0 \leqq j} f_j x^j \in F[x]$, the (B, N)-representation $\rho(f) = (f_0, \cdots, f_{n-1})$ is simply the sequence of the first n coefficients (irrespective of the degree of f). For a rational function f (with nonzero constant term in the denominator polynomial), $\rho(f)$ is an initial segment of the power series expansion of f. In the example, $N = (5)$ and $\rho(f_1) = \rho(f_2) = (-2, -3, 2, -1, 1)$.

2. *List of values.* Given are $a_1, \cdots, a_n \in F$ pairwise distinct, and $p = n$, $B = (x - a_1, \cdots, x - a_n)$ and $N = (1, \cdots, 1)$. Then $r_i = f(a_i)$ is the value of f at a_i. For the example, let $(a_1, \cdots, a_5) = (-2, -1, 0, 1, 2)$ (and assume that $\operatorname{char} F \neq 5$, so that $\gcd(x - 2, x^2 + x - 1) = 1$). Then $B = (x + 2, x + 1, x, x - 1, x - 2)$, $N = (1, 1, 1, 1, 1)$, $\rho(f_1) = (36, 5, -2, -3, 8)$, and $\rho(f_2) = (-28, 6, -2, -4, \frac{-24}{5})$.

3. *Taylor expansion.* Here some $a \in F$ is given, and $p = 1$, $B = (x - a)$, $N = (n)$, and $r = (r_0, \cdots, r_{n-1})$ where

$$f \equiv \sum_{0 \leqq j < n} r_j (x - a)^j \mod (x - a)^n,$$

so that $r_0, \cdots, r_{n-1}$ are the first n coefficients of the Taylor expansion of f at a. (It is assumed that f can be written with a denominator that does not vanish at a.) If

char $F = 0$, then

$$r_j = \frac{1}{j!}\left(\frac{d^j f}{dx^j}\right)(a).$$

Note that these Taylor coefficients are well-defined for any field F, whereas the expression on the right-hand side above does not make sense if char $F \leqq j$.

The sequence of coefficients is nothing but the Taylor expansion at 0. In the example, if we choose $a = 2$ and $n = 5$ (and char $F \neq 5$), then

$$\rho(f_1) = (8, 25, 20, 7, 1) \quad \text{and} \quad \rho(f_2) = \left(\frac{-24}{5}, \frac{-3}{5}, \frac{4}{25}, \frac{-1}{25}, \frac{1}{125}\right).$$

4. *General list of values.* As a common generalization of the last two cases, we get the general list of values of the rational function and some of its derivatives in "Hermite format". We have $a_1, \cdots, a_p \in F$ pairwise distinct, $B = (x - a_1, \cdots, x - a_p)$ and $N = (n_1, \cdots, n_p)$ with $n_1 + \cdots + n_p = n$. Then

$$\rho(f) = (r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1}),$$

where

$$r_i = \sum_{0 \leqq j < n_i} r_{ij}(x - a_i)^j \equiv f \mod (x - a_i)^{n_i}$$

is the initial segment of length n_i of the Taylor expansion of f at a_i. In the example, we choose $p = 2$, $a_1 = -1$, $a_2 = 2$, $B = (x + 1, x - 2)$ and $N = (2, 3)$ (and again assume char $F \neq 5$). Then

$$\rho(f_1) = (5, -14; 8, 25, 20) \quad \text{and} \quad \rho(f_2) = \left(6, -21; \frac{-24}{5}, \frac{-3}{5}, \frac{4}{25}\right).$$

Remark. Note that the standard encoding of a rational function $f = g/h$ by the coefficients of g and h is not a representation in our sense. However, it will play a crucial role in the conversion algorithms of the next section.

3. Conversion algorithms. We now describe two algorithms: the first one converts a rational function that is given as the quotient of two polynomials (and these polynomials are represented by their coefficient sequences) into its representation in base B with precision N, and the second one performs the inverse conversion. Thus the standard representation by coefficients plays a special role: The basic parts r_{ij} are given by their coefficients, and our conversions from one representation to another pass via this standard representation.

We will make use of the Extended Euclidean Scheme of two polynomials $a_0, a_1 \in F[x]$, where $a_1 \neq 0$:

$$\begin{aligned} a_0 &= q_1 a_1 + a_2, & s_2 a_0 + t_2 a_1 &= a_2, \\ &\vdots & &\vdots \\ a_{l-2} &= q_{l-1} a_{l-1} + a_l, & s_l a_0 + t_l a_1 &= a_l, \\ a_{l-1} &= q_l a_l, & s_{l+1} a_0 + t_{l+1} a_1 &= a_{l+1}, \end{aligned}$$

where the following conditions are satisfied for $2 \leqq k \leqq l + 1$: $a_k, q_k, s_k, t_k \in F[x]$, $a_{l+1} = 0$, $a_{k-1} = q_k a_k + a_{k+1}$, $\deg a_k < \deg a_{k-1}$, $s_0 = 1$, $t_0 = 0$, $s_1 = 0$, $t_1 = 1$, $s_k = s_{k-2} - q_{k-1} s_{k-1}$ and $t_k = t_{k-2} - q_{k-1} t_{k-1}$. Thus the q's are the quotients and the a's the remainders of Euclid's algorithm, the s's and t's are the "continuants" or "convergents", and $\gcd(a_0, a_1)$ is

the unique monic scalar multiple of a_i. (By convention, all gcd's of polynomials in $F[x]$ are monic.) Also note that $\gcd(s_k, t_k) = 1$ and $\deg a_{k-1} + \deg t_k = \deg a_0$ for all $k \geqq 1$.

ALGORITHM STANDARD-TO-REPRESENTATION. (Standard coefficients to base representation.)

Input: A base $B = (b_1, \cdots, b_p)$ with a precision $N = (n_1, \cdots, n_p)$ and total precision n, and a pair (g, h) of polynomials in $F[x]$ such that $\gcd(b_1 \cdots b_p, h) = 1$. All input polynomials are given by their coefficient vectors.

Output: A (B, N)-representation r of $f = g/h$.

1. For all i, $1 \leqq i \leqq p$, do steps 2, 3, 4.
2. Compute $s_i, t_i \in F[x]$ such that
   $s_i b_i^{n_i} + t_i h = 1$,
   $\deg t_i < n_i \deg b_i$.
3. Compute $r_i \in F[x]$ such that
   $r_i \equiv g t_i \bmod b_i^{n_i}$,
   $\deg r_i < n_i \deg b_i$.
4. For all j, $0 \leqq j < n_i$, compute $u_{ij}, v_{ij}, r_{ij} \in F[x]$ such that
   $r_i = u_{ij} b_i^j + v_{ij}$,
   $\deg v_{ij} < j \deg b_i$,
   $r_{ij} = u_{ij} - u_{i,j+1} b_i$.
   (Division with remainder of r_i by b_i^j. Use $u_{i0} = r_i$ and $u_{i,n_i} = 0$.)
5. Return
   $r = (r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1})$.

ALGORITHM REPRESENTATION-TO-STANDARD. (Base representation to standard coefficients.)

Input: A base $B = (b_1, \cdots, b_p)$ with a precision $N = (n_1, \cdots, n_p)$ and total precision n, a representative $r \in R(B, N)$ and $d \in \mathbf{N}$ with $d < n$. (This number d serves as a bound on the degree of the denominator polynomial.) Again all input polynomials are given by their coefficients.

Output: The coefficients of two polynomials $g, h \in F[x]$ such that r is a weak (B, N)-representation of (g, h), $\deg h \leqq d$, $\deg g < n - d$, h is monic and $\gcd(b_1^{n_1} \cdots b_p^{n_p}, h) = \gcd(g, h)$.

1. For all i, $1 \leqq i \leqq p$, compute $r_i = \sum_{0 \leqq j < n_i} r_{ij} b_i^j$.
2. For all i, $1 \leqq i \leqq p$, compute $v_i, e_i \in F[x]$ such that

$$v_i b_1^{n_1} \cdots b_{i-1}^{n_{i-1}} b_{i+1}^{n_{i+1}} \cdots b_p^{n_p} \equiv 1 \bmod b_i^{n_i},$$

$$\deg v_i < n_i \deg b_i,$$

$$e_i = v_i b_1^{n_1} \cdots b_{i-1}^{n_{i-1}} b_{i+1}^{n_{i+1}} \cdots b_p^{n_p}.$$

3. Compute $\omega \in F[x]$ such that

$$w \equiv \sum_{1 \leqq i \leqq p} r_i e_i \bmod b_1^{n_1} \cdots b_p^{n_p},$$

   $\deg w < \sum n_i \deg b_i = n$.
   (Then for all i, $w \equiv r_i \bmod b_i^{n_i}$.)
4. Compute the length l and the entries a_k, s_k, t_k, where $2 \leqq k \leqq l+1$, of the Extended Euclidean Scheme of $(a_0, a_1) = (b_1^{n_1} \cdots b_p^{n_p}, w)$.
5. Determine k, $1 \leqq k \leqq l+1$, such that $\deg a_k < n - d \leqq \deg a_{k-1}$, the leading coefficient λ of t_k, and return $g = \lambda^{-1} a_k$, $h = \lambda^{-1} t_k$.

THEOREM 3.1. *The algorithms STANDARD-TO-REPRESENTATION and REPRESENTATION-TO-STANDARD work correctly as described. They can be performed on arithmetic networks of depth* $O(\log^2 n)$ *and size* $n^{O(1)}$ *for inputs that have total precision at most n and (for STANDARD-TO-REPRESENTATION)* $\deg g + \deg h < n$.

Proof. We first consider algorithm *STANDARD-TO-REPRESENTATION*. The depth and size bounds follow from the results in von zur Gathen [1984b].

To prove correctness, we note that for $1 \leqq i \leqq p$

$$\sum_{0 \leqq j < n_i} r_{ij} b_i^j = \sum_{0 \leqq j < n_i} (u_{ij} - u_{i,j+1} b_i) b_i^j$$

$$= \sum_{0 \leqq j < n_i} (u_{ij} b_i^j - u_{i,j+1} b_i^{j+1}) = u_{i0} - u_{in_i} b_i^{n_i} = r_i,$$

and the polynomial

$$r_{ij} b_i^j = u_{ij} b_i^j - u_{i,j+1} b_i^{j+1} = -v_{ij} + v_{i,j+1}$$

has degree $< (j+1) \deg b_i$, and hence $\deg r_{ij} < \deg b_i$. This shows that $r \in R(B, N)$, and

$$g - hr_i \equiv g - hgt_i \equiv g(1 - ht_i) \equiv 0 \bmod b_i^{n_i}$$

implies that r is a weak (B, N)-representation of (g, h). Thus algorithm *STANDARD-TO-REPRESENTATION* works correctly.

To prove correctness of algorithm *REPRESENTATION-TO-STANDARD*, we first note that v_i as required in step 2 exists, since $b_1, \cdots, b_p$ are pairwise relatively prime. For $1 \leqq i, j \leqq p$ we have

$$e_i \equiv \delta_{ij} \bmod b_j^{n_j}$$

(so that the e_i's are "mutually orthogonal idempotents" in $F[x]/(b_1^{n_1} \cdots b_p^{n_p})$), and $w \equiv r_i \bmod b_i^{n_i}$. (This is the "Lagrange version" of the Chinese remainder algorithm.)

Clearly $s_k b_1^{n_1} \cdots b_p^{n_p} + t_k w = a_k$ implies that

$$g \equiv hw \equiv h \sum_{0 \leqq j < n_i} r_{ij} b_i^j \bmod b_i^{n_i}$$

for all i. Also

$$\gcd(g, h) = \gcd(b_1^{n_1} \cdots b_p^{n_p}, h),$$

since $\gcd(s_k, t_k) = 1$. (For later use, it is convenient to make t_k monic.) Furthermore, $\deg a_{k-1} + \deg t_k = n$ and $n - d \leqq \deg a_{k-1}$ imply the required degree bound $\deg h \leqq d$. The depth estimate $O(\log^2 n)$ follows from von zur Gathen [1984b]. □

Remark 3.2. Division with remainder of two polynomials in $F[x]$ of degree at most n can be performed in depth $O(\log n)$ and size $n^{O(1)}$, and also the iterated product $p = p_1 \cdots p_n$ of polynomials with $\deg p_i \leqq n$. This was proved by Reif [1983] in the case that F supports the Fast Fourier Transform, and by Eberly [1984] in general (for P-uniform networks). Therefore all steps of the two conversion algorithms can be performed in depth $O(\log n)$, except possibly step 2 of *STANDARD-TO-REPRESENTATION*, and steps 2 and 4 of *REPRESENTATION-TO-STANDARD*. In particular, both conversion problems (i.e. computing the output from the input, as stated) are "log-depth reducible" to EES, the problem of computing all entries of the Extended Euclidean Scheme of two univariate polynomials. (The reduction is P-uniform, rather than the usually required log-space uniform.)

On the other hand, also EES is log-depth reducible to the problem of computing the conversion from representation to standard coefficients: given $a_0, a_1 \in F[x]$ with

$0 \leqq \deg a_1 < \deg a_0 \leqq n$, we can compute for all d, $0 \leqq d < n$, polynomials u_d, b_d such that

$$u_d a_1 \equiv b_d \bmod a_0,$$

$$\deg b_d < n - d, \qquad \deg u_d \leqq d, \quad u_d \text{ monic},$$

by the conversion from the $((a_0), (1))$-representation $r = (a_1)$ of a_1 to standard representation with degree bound d. Then we can compute $v_d \in F[x]$ such that

$$v_d a_0 + u_d a_1 = b_d.$$

It follows from von zur Gathen [1984b], Lemma 2.2, that the nonzero b_d of minimal degree is a scalar multiple of a_l, and together with those b_d for which $b_d \nmid b_{d-1}$ we get—up to scalar multiples—the remainders of the Extended Euclidean Scheme of (a_0, a_1), and v_d, u_d are the "convergents". It is easy then to also compute the quotients required for EES, and the correct scalar multipliers in depth $O(\log n)$ (see von zur Gathen [1984b]).

Thus the two problems EES and conversion from representation to standard coefficients are log-depth equivalent. In particular, the continued fraction decomposition $(q_1, \cdots, q_l)$ of $a_0/a_1 \in F(x)$ can be calculated via this conversion problem.

Remark 3.3. In this paper, we only consider algebraic complexity, using the model of arithmetic networks over an arbitrary field F. When F is $\mathbf{Q}$ or a finite field, it also makes sense to ask for Boolean circuits implementing the algorithms, with inputs now represented in some standard fashion over the alphabet $\{0, 1\}$, say. It follows from the results in Borodin, Cook, Pippenger [1983] (see also Eberly [1984]) that both conversion algorithms can be performed on log-space uniform families of Boolean circuits of depth $O(\log^2 n)$ and size $n^{O(1)}$. Note, however, that at the present time no $(\log n)^{O(1)}$ depth method is known to compute modular inverses (or even the gcd) of n-bit integers. When F is $\mathbf{Q}$ or some $\mathbf{Z}_p$, then the above Boolean circuits use a "redundant notation" u/v for field elements, with $u, v \in \mathbf{Z}$, $v > 0$, but not necessarily $\gcd(u, v) = 1$. If $F = \mathbf{Z}_p$, then we can also enforce $0 \leqq u, v < p$, but we do not know how to replace (u, v) by the unique w such that $w \equiv (u/v) \bmod p$ and $0 \leqq w < p$.

4. Representability. In this section, we discuss the questions of Representability and Convertibility as mentioned in § 2. We first provide a (large) subset $S(B) \subseteq F(x)$ such that every $f \in S(B)$ has a unique (B, N)-representation, and then a subset $T_{B,n,d} \subseteq F(x)$ such that on $T_{B,n,d}$, algorithms *STANDARD-TO-REPRESENTATION* and *REPRESENTATION-TO-STANDARD* compute functions that are inverse to each other.

Fix a base B and a precision N. If $f = g/h$ with $g, h \in F[x]$, $\gcd(g, h) = 1$ and $\gcd(b_1, h) \neq 1$, then f has of course no representation in base B. Namely, if r were a representation, then

$$\gcd(b_1, h) | g - hr_1,$$

and hence

$$\gcd(b_1, h) | g,$$

contradicting the assumption.

The example $(bg)/(bh)$ of § 2 showed that the congruence condition in Definition 2.1 is too weak for unique representation. To rule out this example, we introduced the gcd condition, which corresponds to

$$S(B) = \{f \in F(x): \exists g, h \in F[x] \text{ such that } f = g/h \text{ and } \gcd(b_1 \cdots b_p, h) = 1\}.$$

(This semilocal ring $S(B)$ is the intersection in $F(x)$ of all localizations $F[x]_{(q)}$, with

q running through the irreducible factors of $b_1 \cdots b_p$.) We now show that Representability holds for the elements of $S(B)$.

THEOREM 4.1. *Let B be a base, N a precision for B, $g, h \in F[x]$ with* $\gcd(g, h) = 1$, *and $f = g/h \in F(x)$. Then the following are equivalent*:

(i) *(g, h) has a unique weak (B, N)-representation,*

(ii) *f has a unique (B, N)-representation,*

(iii) $\gcd(b_1 \cdots b_p, h) = 1$,

(iv) $f \in S(B)$.

Proof. We will prove "(i)⇒(iv)⇒(iii)⇒(i)" and "(ii)⇔(iii)".

For "(i)⇒(iv)", assume $f \notin S(B)$, and that (g, h) has a weak (B, N)-representation $r = (r_{10}, \cdots, r_{p,n_p-1}) \in R(B, N)$. Then $\gcd(b_1 \cdots b_p, h) \neq 1$, and we take i and an irreducible polynomial $q \in F[x]$ such that $q | \gcd(b_i, h)$. Now change r to $\bar{r} \in R(B, N)$ by replacing r_{i,n_i-1} with $\bar{r}_{i,n_i-1} = r_{i,n_i-1} + b_i/q$. Then

$$hr_{i,n_i-1} b_i^{n_i-1} \equiv h\bar{r}_{i,n_i-1} b_i^{n_i-1} \bmod b_i^{n_i},$$

and $\bar{r} \neq r$ is also a weak (B, N)-representation of (g, h).

For "(iv)⇒(iii)", let $f \in S(B)$, and write $f = \bar{g}/\bar{h}$ with $\bar{g}, \bar{h} \in F[x]$ and $\gcd(b_1 \cdots b_p, \bar{h}) = 1$. Now assume that $\gcd(b_1 \cdots b_p, h) \neq 1$, say that $q | \gcd(b_1 \cdots b_p, h)$ for some irreducible $q \in F[x]$. Then from $\bar{g}h = g\bar{h}$ we get $q | g\bar{h}$ and hence $q | g$, a contradiction.

For "(iii)⇒(i)", we have seen in Theorem 3.1 that algorithm *STANDARD-TO-REPRESENTATION* with input (B, N, g, h) computes a (B, N)-representation of f, which is of course also a weak (B, N)-representation of (g, h). So now suppose that (g, h) has two weak (B, N)-representations $r, \bar{r} \in R(B, N)$, and let $r_i = \sum_{0 \leq j < n_i} r_{ij} b_i^j$, and similarly $\bar{r}_i$. Then for all i, $1 \leqq i \leqq p$, we have

$$b_i^{n_i} | h(r_i - \bar{r}_i), \qquad \gcd(b_i, h) = 1,$$

$$\deg(r_i - \bar{r}_i) < n_i \deg b_i,$$

which implies that

$$b_i^{n_i} | r_i - \bar{r}_i,$$

and hence $r_i = \bar{r}_i$. It follows that $r = \bar{r}$, and (i) is proven.

For "(iii)⇒(ii)", we first note that, assuming (iii), any $r \in R(B, N)$ is a weak (B, N)-representation of (g, h) if and only if it is a (B, N)-representation of f. Since (iii) implies (i), (ii) also follows.

For "(ii)⇒(iii)", we assume that f has a (unique) (B, N)-representation, and therefore $f = \bar{g}/\bar{h}$ with $\bar{g}, \bar{h} \in F[x]$ and $\gcd(b_1 \cdots b_p, \bar{h}) = 1$. Then $\bar{g}h = g\bar{h}$, and if $q | \gcd(b_i, h)$ for some i and $q \in F[x]$ irreducible, then $q | g$. (iii) is proven. □

Example 4.2. For the second property of algorithms *REPRESENTATION-TO-STANDARD* and *STANDARD-TO-REPRESENTATION* computing inverse functions, consider the example $p = 1$, $B = (x^3 - x)$, $N = (1)$, $d = 1$ and $r = (x^2 + 1)$ (and assume char $F \neq 2$). The output of algorithm *REPRESENTATION-TO-STANDARD* is $(-2x, -x)$. If we now simplify $f = -2x/-x$ to $f = 2$ and apply algorithm *STANDARD-TO-REPRESENTATION* with input B, N, $(2, 1)$, the output is $(2) \neq r$. This example makes it clear that the second property will not hold for all $r \in R(B, N)$. But we will see that it holds for "almost all" r.

Consider the following set $T_{B,n,d}$:

$$T_{B,n,d} = \{(g, h) \in F[x]^2 : \gcd(g, h) = \gcd(b_1 \cdots b_p, h) = 1,$$

$$h \text{ is monic, } \deg g < n - d, \deg h \leqq d\}.$$

Ignoring the data B, N, d, which remain unchanged throughout the algorithms (d does not even appear in *STANDARD-TO-REPRESENTATION*), we now write $\alpha(g, h) = r$ and $\beta(r) = (g, h)$ for the functions computed by the two algorithms *STANDARD-TO-REPRESENTATION* and *REPRESENTATION-TO-STANDARD*. For $(g, h) \in T_{B,n,d}$, we have $f = g/h \in S(B)$, and $r = \alpha(g, h)$ is a (B, N)-representation of f.

THEOREM 4.3. *Let B be a base, N a precision for B with total precision n, and $d < n$. Then the functions α and β are inverse bijections between the set $T_{B,n,d} \subseteq F(x)$ and its image in $R(B, N)$.*

Proof. Let $U = \alpha(T_{B,n,d})$ be the image of $T_{B,n,d}$ in $R(B, N)$. It is sufficient to show that $\beta \circ \alpha(\bar{g}, \bar{h}) = (\bar{g}, \bar{h})$ for any $(\bar{g}, \bar{h}) \in T_{B,n,d}$, since then $\alpha: T_{B,n,d} \to U$ is injective, and surjective by definition. Furthermore, β is a section of α, and hence its inverse.

So let $(g, h) = \beta \circ \alpha(\bar{g}, \bar{h}) = (\lambda^{-1} a_k, \lambda^{-1} t_k)$, using the notation of the algorithms, and write $b = b_1^{n_1} \cdots b_p^{n_p}$. We can assume $\bar{g} \neq 0$, and then $a_k \neq 0$. From

$$\bar{g} \equiv \bar{h} r_i \mod b_i^{n_i}$$

for all i, it follows that

$$\bar{g} \equiv \bar{h} w \mod b,$$

and there exists $s \in F[x]$ such that

$$sb + \bar{h} w = \bar{g},$$

$$\deg \bar{g} + \deg \bar{h} < n = \deg b.$$

By the uniqueness of the Extended Euclidean Scheme (see e.g. von zur Gathen [1984b], Lemma 2.2), there exists $m \in \{1, \cdots, l\}$ and $u \in F[x]$ such that

$$\bar{g} = u a_m, \qquad \bar{h} = u t_m,$$

$$\deg a_m \leqq \deg \bar{g} < \deg a_{m-1}.$$

Since $\deg a_m \leqq \deg \bar{g} < n - d \leqq \deg a_{k-1}$, we have $k \leqq m$. But on the other hand $n - \deg a_{m-1} = \deg t_m \leqq d$ implies that $n - d \leqq \deg a_{m-1}$, and therefore $m \leqq k$. It follows that $m = k$, and

$$\bar{g} = u \lambda^{-1} g, \qquad \bar{h} = u \lambda^{-1} h.$$

Since $\gcd(\bar{g}, \bar{h}) = 1$, $u \in F$. Both h and $\bar{h}$ are monic, therefore $u\lambda^{-1} = 1$. □

5. Applications. We can now reap the fruits of the work spent in setting up the previous notation. By putting together algorithms *REPRESENTATION-TO-STANDARD* and *STANDARD-TO-REPRESENTATION* (using different bases) we obtain a fast parallel algorithm for conversion from one base to another. This yields fast parallel algorithms for a number of important problems in symbolic manipulation (and also clarifies how these problems are related to each other).

INTERPOLATION (n) has as input a pair (a, r) where $a = (a_1, \cdots, a_n)$ and $r = (r_1, \cdots, r_n)$ have entries from F, and $a_i \neq a_j$ for $1 \leqq i < j \leqq n$. The output are the coefficients of the unique $f \in F[x]$ such that $\deg f < n$ and $f(a_i) = r_i$ for $1 \leqq i \leqq n$. This function is nothing but the conversion from $((x - a_1, \cdots, x - a_n), (1, \cdots, 1))$-representation to $((x), (n))$-representation. (In order to obtain the unique monic $g \in F[x]$ of degree n such that $g(a_i) = r_i$ for all i, one uses the interpolation polynomial f for the values $r_i - a_i^n$, and $g = x^n + f$.)

TAYLOR EXPANSION (n) has as input $a \in F$ and the coefficients $(g_0, \cdots, g_{n-d-1}; h_0, \cdots, h_d)$ of

$$f = \frac{\sum_{0 \leqq j < n-d} g_j x^j}{\sum_{0 \leqq j \leqq d} h_j x^j} \in F(x)$$

where $\sum h_j a^j \neq 0$. The output are the Taylor coefficients $f_0, \cdots, f_{n-1} \in F$ of f at a, so that $f \equiv \sum_{0 \leqq j < n} f_j (x-a)^j \bmod (x-a)^n$. This is the conversion from coefficients to $((x-a), (n))$-representation.

HERMITE INTERPOLATION (n) has an input of the form $(a, r_1, \cdots, r_p)$, where $a = (a_1, \cdots, a_p)$, $r_i = (r_{i0}, \cdots, r_{i,n_i-1})$, all $a_i, r_{ij} \in F$, and $a_i \neq a_j$ for $1 \leqq i < j \leqq p$, and $n_1 + \cdots + n_p = n$. The output are the coefficients of the unique polynomial $f \in F[x]$ such that $\deg f < n$ and

$$f \equiv \sum_{0 \leqq j < n_i} r_{ij} (x - a_i)^j \bmod (x - a_i)^{n_i}$$

for $1 \leqq i \leqq p$.

Thus the first n_i coefficients of the Taylor expansion of f at a_i are prescribed. If $\operatorname{char} F = 0$, this is equivalent to prescribing the values of the first n_i derivatives of f at a_i as

$$\left(\frac{d^j f}{dx^j}\right)(a_i) = j!\, r_{ij}.$$

HERMITE INTERPOLATION (n) is the conversion from $((x-a_1, \cdots, x-a_p), (n_1, \cdots, n_p))$-representation to $((x), (n))$-representation. INTERPOLATION is a special case of this problem. Again, one can also compute the unique monic interpolation polynomial of degree n.

CHINESE REMAINDER ALGORITHM (n) has as input a sequence $(b_1, \cdots, b_p, r_1, \cdots, r_p)$ of polynomials from $F[x]$ such that $\deg (b_1 \cdots b_p) = n$, $\deg r_i < \deg b_i$, the b_i's are nonconstant and pairwise relatively prime. The output are the coefficients of the unique $f \in F[x]$ such that $f \equiv r_i \bmod b_i$ for $1 \leqq i \leqq p$ and $\deg f < n$. This is the conversion from $((b_1, \cdots, b_p), (1, \cdots, 1))$-representation to $((x), (n))$-representation.

ELEMENTARY SYMMETRIC FUNCTIONS (n) has as input a sequence $(c_1, \cdots, c_n)$ with $c_i \in F$. Output are the elementary symmetric functions $s_j = \sigma_j(c_1, \cdots, c_n)$ for $1 \leqq j \leqq n$. Thus

$$t^n - s_1 t^{n-1} + \cdots + (-1)^n s_n = (t - c_1) \cdots (t - c_n),$$

where t is an indeterminate. This can be viewed as a special case of the monic version of HERMITE INTERPOLATION: Set $C = \{c_1, \cdots, c_n\}$, $p = \# C$, $\{a_1, \cdots, a_p\} = C$, and $r_{ij} = 0$ for $0 \leqq j < n_i$, where a_i occurs exactly n_i times among $c_1, \cdots, c_n$. The inverse function—root-finding—cannot be computed by a rational algorithm.

PARTIAL FRACTION DECOMPOSITION (n) has as input the coefficients of polynomials $b_1, \cdots, b_p$ and q, and $n_1, \cdots, n_p \in \mathbf{N}$ such that $b_1, \cdots, b_p$ are nonconstant and pairwise relatively prime, and $\deg q < \deg (b_1^{n_1} \cdots b_p^{n_p}) = n$. Output are the coefficients of the unique polynomials s_{ij} $(1 \leqq i \leqq p, 1 \leqq j < n_i)$ such that

$$\frac{q}{b_1^{n_1} \cdots b_p^{n_p}} = \sum_{\substack{1 \leqq i \leqq p \\ 1 \leqq j \leqq n_i}} \frac{s_{ij}}{b_i^j},$$

$$\deg s_{ij} < \deg b_i.$$

(For a slightly more general case, one would not assume $\deg q > n$; then one first has to perform a division with remainder to get a polynomial summand plus a problem of the above format.) We note that for all i

$$q \equiv \sum_{\substack{1 \leqq i \leqq p \\ 1 \leqq j \leqq n_i}} b_1^{n_1} \cdots b_{i-1}^{n_{i-1}} b_{i+1}^{n_{i+1}} \cdots b_p^{n_p} \; s_{ij} b_i^{n_i - j} \bmod b_i^{n_i}.$$

We first convert the $((x),(n))$-representation of q to the $((b_1, \cdots, b_p), (n_1, \cdots, n_p))$-representation r, then for all i, $1 \leqq i \leqq p$, take r_i and v_i as in steps 1 and 2 of *REPRESENTATION-TO-STANDARD*, and compute the $((b_i),(n_i))$-representation $(s_{i0}^*, \cdots, s_{i,n_i-1}^*)$ of $v_i r_i$. Then $s_{ij} = s_{i,n_i-j}^*$ for $1 \leqq j \leqq n_i$.

PADÉ APPROXIMATION (n) has as input a polynomial $f \in F[x]$ of degree $< n$, and $d \in \mathbf{N}$ with $0 \leqq d < n$. The output consists of the coefficients of polynomials $g, h \in F[x]$ such that $g \equiv fh \bmod x^n$, $\deg g < n - d$, $\deg h \leqq d$, and h is monic. Thus $g/h \equiv f \bmod x^n$ is a Padé approximant to f (provided $h(0) \neq 0$). This is the conversion from the $((x),(n))$-representation $(f_0, \cdots, f_{n-1})$ of $f = \sum f_i x^i$ to standard representation. (For sequential Padé computations, see Gragg [1972], Geddes [1979], Brent, Gustavson, Yun [1980], and for an overview of the theory Baker, Graves-Morris [1981].)

CAUCHY INTERPOLATION (n) has as input $d \in \mathbf{N}$ with $0 \leqq d < n$ and a pair (a, r) where $a = (a_1, \cdots, a_n)$, $r = (r_1, \cdots, r_n) \in F^n$, and $a_i \neq a_j$ for $1 \leqq i < j \leqq n$. The output consists of the coefficients of polynomials $g, h \in F[x]$ such that $g(a_i) = h(a_i) r_i$ for $1 \leqq i \leqq n$, $\deg g < n - d$, $\deg h \leqq d$, and h is monic. Thus $f = g/h$ is a rational function with prescribed denominator degree that interpolates the given values r_i at a_i, i.e. $f(a_i) = r_i$ (provided $h(a_i) \neq 0$). Cauchy [1821] had considered this problem and given an explicit solution by a closed formula similar to the Lagrange interpolation formula. For a modern treatment, see Gustavson, Yun [1979] and Knuth [1981, Exercise 4.7-13]. Algorithm *REPRESENTATION-TO-STANDARD* with base $B = (x - a_1, \cdots, x - a_n)$, precision $N = (1, \cdots, 1)$, and degree bound d computes a solution.

RATIONAL HERMITE INTERPOLATION (n) has an input of the form $(d, a, r_1, \cdots, r_p)$ where $0 \leqq d < n$, $a = (a_1, \cdots, a_p)$, $r_i = (r_{i0}, \cdots, r_{i,n_i-1})$, all $a_i, r_{ij} \in F$, $a_i \neq a_j$ for $1 \leqq i < j \leqq p$, and $n_1 + \cdots + n_p = n$. The output are the coefficients of polynomials $g, h \in F[x]$ such that

$$g \equiv h \cdot \sum_{0 \leqq j < n_i} r_{ij} (x - a_i)^j \bmod (x - a_i)^{n_i},$$

$\deg g < n - d$, $\deg h \leqq d$, and h is monic. Thus the initial segments of the Taylor expansion of the rational function $f = g/h$ at a_i are prescribed, i.e.

$$f \equiv \sum_{0 \leqq j < n_i} r_{ij} (x - a_i)^j \bmod (x - a_i)^{n_i}$$

(provided $h(a_i) \neq 0$). This problem simultaneously generalizes HERMITE INTERPOLATION (which has $d = 0$), PADÉ APPROXIMATION (which has $p = 1$ and $a_1 = 0$), and CAUCHY INTERPOLATION (which has $n_1 = \cdots = n_p = 1$). It can be computed by algorithm *REPRESENTATION-TO-STANDARD* with input $B = (x - a_1, \cdots, x - a_p)$, $N = (n_1, \cdots, n_p)$ and $r = (r_1; \cdots; r_p)$.

The following theorem tells us that all the above problems have a fast parallel solution.

THEOREM 5.1. *The following nine functions can be computed in depth* $O(\log^2 n)$ *and size* $n^{O(1)}$: *INTERPOLATION, TAYLOR EXPANSION, HERMITE INTERPOLATION, CHINESE REMAINDER ALGORITHM, ELEMENTARY SYMMETRIC FUNCTIONS, PARTIAL FRACTION DECOMPOSITION, PADÉ APPROXIMATION, CAUCHY INTERPOLATION, RATIONAL HERMITE INTERPOLATION.*

Proof. Use Theorem 3.1 and the fact that all these problems can be solved by algorithms *STANDARD-TO-REPRESENTATION* and *REPRESENTATION-TO-STANDARD.* □

Remark 5.2. Not unexpectedly, the above general result is not the best possible, at least in some cases dealing with polynomial problems. For a polynomial f, TAYLOR EXPANSION can be computed by calculating binomial coefficients and evaluating universal Taylor coefficients of f (see von zur Gathen [1984b, § 6]). This can be performed in depth $O(\log n)$, so that the corresponding statement of Theorem 5.1 is interesting only in the case of rational functions. Reif [1983] provides less obvious methods for interpolation and elementary symmetric functions that use only depth $O(\log n)$ and polynomial size. These methods assume that certain roots of unity are available in the ground field; they have been extended by Eberly [1984] to hold over arbitrary fields.

Remark 5.3. Clearly all our computational problems can be expressed by systems of linear equations, and these are always solvable in depth $O(\log^2 n)$ (Csanky [1976], Borodin, von zur Gathen, Hopcroft [1982], Berkowitz [1984]). However, for singular systems over arbitrary fields only a probabilistic algorithm is known. The above results amount to saying that in the cases considered here it is sufficient to solve nonsingular systems of linear equations; again this is well-known in some cases, such as interpolation by polynomials (Vandermonde matrix).

Remark 5.4. It is well-known that in some cases the rational interpolation problem does not have a solution. Theorem 4.3 shows that most $r \in R(B, N)$ are representations of a rational function $f = g/h$ with $\deg g < n - d$ and $\deg h \leqq d$ (g and h are polynomials). However, in Example 4.2 we have seen that not all $r \in R(B, N)$ are such representations. For a general discussion of this phenomenon, let $B = (b_1, \cdots, b_p)$ be a base, N a precision for B, n the total precision, $0 \leqq d < n$, and $r \in R(B, N)$. Set $a_0 = b_1^{n_1} \cdots b_p^{n_p}$, and let $a_1 \in F[x]$ be a polynomial such that $\deg a_1 < n$ and r is a (B, N)-representation of a_1. By the Chinese Remainder Theorem (or by Theorem 4.1), a_1 exists and is unique. Furthermore, let $a_k, s_k, t_k \in F[x]$ be the (unique) entries of the Extended Euclidean Scheme for (a_0, a_1) with $\deg a_k < n - d \leqq \deg a_{k-1}$. We then have

$$s_k a_0 + t_k a_1 = a_k,$$

$$t_k a_1 \equiv a_k \mod a_0.$$

Now assume that $\gcd(a_0, t_k) = 1$. Then r is indeed the (B, N)-representation of $f = a_k / t_k$, and $\gcd(a_k, t_k) = 1$. The latter property follows using the fact that $\gcd(s_k, t_k) = 1$. Thus "$\gcd(a_0, t_k) = 1$" is a sufficient condition which guarantees that r is the representation of a rational function. Note that it is always satisfied if $d = 0$, and then $k = 1$, $a_k = a_1$, $t_k = 1$.

This sufficient condition holds "almost everywhere" in the following sense. For fixed numbers $d, p, n_1, \cdots, n_p, m_1, \cdots, m_p$ the set of inputs $(b_1, \cdots, b_p, r)$ for algorithm *REPRESENTATION-TO-STANDARD* with $\deg b_i = m_i$ can be considered as a subset of F^m, where

$$m = p + \sum_{1 \leqq i \leqq p} (n_i + 1) m_i.$$

It is a Zariski-open dense subset, the only condition being that $\deg b_i = m_i$ and $\gcd(b_i, b_j) = 1$ for $1 \leqq i < j \leqq p$; the latter condition can be expressed by the nonvanishing of a resultant polynomial. Now there exists a nonzero polynomial P in m variables such that

$$P(b_1, \cdots, b_p, r) \neq 0 \Rightarrow \gcd(a_0, t_k) = 1,$$

using the above notation. Thus the sufficient condition holds "almost everywhere" in the strong sense of algebraic geometry, namely on a Zariski-open dense subset of the input set (assuming that F is infinite). (P can be chosen so that $P(b_1, \cdots, b_p, r) \neq 0$ will imply that the Euclidean Scheme for (a_0, a_1) is normal, i.e. that deg $q_k = 1$ for all k. But in fact, a polynomial P as above exists for every subset of the input space corresponding to some $D(d_1, \cdots, d_{l+1})$, using the notation from Strassen [1983, (5.3)], and such that P does not vanish identically on the subset.)

Example 5.5. Let us now look at a "bad case" for Padé approximation. Let $n \geqq 5$ and $r = x^{n-1} - x^{n-2} + 1$. Then $g = h = x^2$ is the only solution of the conditions

$$g, h \in F[x], \qquad g \equiv hr \mod x^n,$$

$$\deg g < n-2, \qquad \deg h \leqq 2, \quad h \text{ monic.}$$

In particular, there does not exist a Padé approximant $f = g/h \in F(x)$ satisfying the above conditions and gcd $(g, h) = 1$. Algorithm *REPRESENTATION-TO-STANDARD* with input $((x), (n))$, $d = 2$, r will output $(g, h) = (x^2, x^2)$. Similarly, Example 4.2 shows that there is no $f = g/h \in F(x)$ such that $f(1) = 2$, $f(0) = 1$, $f(-1) = 2$, and $g, h \in F[x]$, deg g, deg $h \leqq 1$ and gcd $(g, h) = 1$. Algorithm *REPRESENTATION-TO-STANDARD* will compute $(g, h) = (-2x, -x)$ which satisfies the conditions

$$g(1) = 2h(1), \quad g(0) = h(0), \quad g(-1) = 2h(-1).$$

This is a "fake solution" which does not yield an interpolating rational function, while "almost all" other such interpolation problems do have a solution. This phenomenon of nonexistence of solutions to the Padé approximation problem and for rational interpolation was discovered by Kronecker, who illustrated it with an example. In Padé's work [1892], this fundamental limitation does not appear. For further examples, see e.g. Graves-Morris [1980].

Historical remark 5.6. The general idea in this paper is to reduce an algebraic problem to systems of linear equations; for the latter we have fast parallel algorithms. This idea is in fact quite old. Jacobi [1846] reduces the rational interpolation problem (for the general Hermite format) to systems of linear equations, and also explicitly describes the solutions in terms of determinants of the inputs; he has several such descriptions. In particular, Jacobi seems to have been the first to assert the existence of what is now called Padé approximations. Jacobi also realized the close connection between the interpolation problem and general banded systems of linear equations in the Hankel format.

Cauchy [1821] had previously stated an explicit solution to the rational interpolation problem by a closed formula.

Kronecker [1881] has two methods for solving the general interpolation problem, one via continued fractions (i.e. essentially the Euclidean Scheme) and one via systems of linear equations. Both Jacobi and Kronecker are motivated by the elimination problem in algebraic geometry.

In hindsight, it is almost surprising how long it took to rediscover these methods and cast them into modern algorithms for algebraic manipulation: the subresultant algorithms of Collins [1967] and Brown [1971] for gcd's and the Padé computations of Geddes [1979] and Brent, Gustavson, Yun [1980]. The essential ingredients of these algorithms (and of those presented here) are already contained in the classical papers.

6. Laurent representations. The representations defined in § 2 were sufficient to describe a unified fast parallel solution to the problems in § 5. Limitations of these representations have been pointed out: not all rational functions have representations,

and representations may not be unique. In this section, we generalize the notion of representation slightly in order to deal more successfully with these issues.

Two generalizations suggest themselves: one can either allow quotients of polynomials in a representation (which would include the standard representation of a rational function as a quotient of two polynomials), or one can allow representations in a "Laurent format". We pursue the latter approach.

DEFINITION 6.1. Let $B=(b_1, \cdots, b_p) \in F[x]^p$ be a base, consisting of nonconstant pairwise relatively prime polynomials, and $N=(n_1, \cdots, n_p) \in \mathbf{N}^p$ a precision for B, with total precision $n=\sum_{1 \leqq i \leqq p} n_i \deg b_i$. A *Laurent* (B, N)*-representative* is a vector

$$r=(m_1, \cdots, m_p; r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1}),$$

where

$$m_i \in \mathbf{Z}, \quad r_{ij} \in F[x], \quad \deg r_{ij} < \deg b_i$$

for all i, j $(1 \leqq i \leqq p, 0 \leqq j < n_i)$. We write $r_i=\sum_{0 \leqq j<n_i} r_{ij} b_i^j$. Now let $g, h \in F[x]$ with $h \neq 0$. We say that r is a *weak Laurent* (B, N)*-representation of* (g, h) if for all i, $b_i^{m_i}$ divides g (if $m_i \geqq 0$), and

$$g b_i^{-m_i} \equiv h r_i \mod b_i^{n_i}.$$

We now present two fast parallel algorithms that perform the conversion between the standard and a Laurent representation of a rational function. Later we will strengthen the conditions on Laurent representations, and obtain a notion with the two crucial properties of Representability and Convertibility. However, for the description of the algorithm it is convenient to work with the weaker notion first.

ALGORITHM STANDARD-TO-LAURENT. (Standard to Laurent representation.)

Input: A base $B=(b_1, \cdots, b_p)$, a precision $N=(n_1, \cdots, n_p)$ for B, and two polynomials $g, h \in F[x]$. All input polynomials are given by their coefficient vectors.

Output: A weak Laurent (B, N)-representation r of (g, h).

1. If $g=0$, return $r=(0, \cdots, 0; 0, \cdots, 0)$ and terminate. Else compute $a=\gcd(g, h)$, the leading coefficient λ of h, and replace (g, h) by $(g/\lambda a, h/\lambda a)$. (By convention, the gcd is monic.)
2. For all i, $1 \leqq i \leqq p$, do steps 3, 4, and 5.
3. Compute $m_i \in \mathbf{Z}$ as follows. If $\gcd(b_i, h)=1$, then

   $$m_i=\max\{j \in \mathbf{N}: b_i^j | g\}.$$

   Else

   $$\gcd(h, b_i^{-m_i}) = \gcd(h, b_i^{-m_i+1}) \neq \gcd(h, b_i^{-m_i-1}).$$

   (Note that this condition determines $m_i < 0$ uniquely.)
4. Compute

   $$c_i=\gcd(b_i^{|m_i|}, h), \quad g_i=\frac{g b_i^{-m_i}}{c_i}, \quad h_i=\frac{h}{c_i} \in F[x].$$

   (Then $c_i=1$ if $\gcd(b_i, h)=1$, and $\gcd(g_i, h_i)=\gcd(b_i, h_i)=1$.)
5. Call algorithm *STANDARD-TO-REPRESENTATION* with input $((b_i), (n_i), (g_i, h_i))$ to return $(r_{i0}, \cdots, r_{i,n_i-1})$.
6. Return

   $$r=(m_1, \cdots, m_p; r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1}).$$

ALGORITHM LAURENT-TO-STANDARD. (Laurent representation to standard.)
Input: A base $B=(b_1, \cdots, b_p)$ and a precision $N=(n_1, \cdots, n_p)$ for B, with total precision n, $d \in \mathbf{N}$, $d<n$, and a Laurent (B, N)-representative

$$r=(m_1, \cdots, m_p; r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1}).$$

Output: Polynomials $g, h \in F[x]$ such that h is monic, and r is a weak Laurent (B, N)-representation of (g, h).

1. For all i, $1 \leqq i \leqq p$, set $e_i=\max\{0, -m_i\}$, and compute $t, u, s_i, t_i, u_i \in F[x]$ such that

$$t=b_1^{m_1+e_1} \cdots b_p^{m_p+e_p}, \qquad u=b_1^{e_1} \cdots b_p^{e_p},$$

$$t_i=\frac{t}{b_i^{m_i+e_i}}, \qquad u_i=\frac{u}{b_i^{e_i}},$$

$$s_i t_i \equiv u_i \sum_{0 \leqq j<n_i} r_{ij} b_i^j \bmod b_i^{n_i},$$

$$\deg s_i < n_i \deg b_i.$$

2. For all i, $1 \leqq i \leqq p$, compute the $((b_i), (n_i))$-representation $s_i'=(s_{i0}, \cdots, s_{i,n_i-1})$ of s_i as in step 4 of algorithm *STANDARD-TO-REPRESENTATION*. (Then $s_i=\sum_{0 \leqq j<n_i} s_{ij} b_i^j$ and $\deg s_{ij}<\deg b_i$.)
3. Call algorithm *REPRESENTATION-TO-STANDARD* with input $(B, N, (s_1', \cdots, s_p'), d)$ to compute $g', h' \in F[x]$ such that for all i

$$g' \equiv h' s_i \bmod b_i^{n_i},$$

$$\deg g' < n-d, \quad \deg h' \leqq d, \quad h' \text{ monic}.$$

4. Let μ be the leading coefficient of u, and return

$$g=\mu^{-1} g' t,$$

$$h=\mu^{-1} h' u.$$

THEOREM 6.2. *The algorithms STANDARD-TO-LAURENT and LAURENT-TO-STANDARD can be performed in depth $O(\log^2 n)$ and size $n^{O(1)}$ on inputs that have total precision at most n and (for STANDARD-TO-LAURENT) $\deg g+\deg h<n$. They work correctly as described in "Output". In fact, the following relations hold, using the notation of the algorithms, and $r_i=\sum_{0 \leqq j<n_i} r_{ij} b_i^j$:*

(i) In *STANDARD-TO-LAURENT*, if $\gcd(g, h)=1$, then for $1 \leqq i \leqq p$,

$$c_i=\gcd(b_i^{|m_i|}, h), \quad \gcd\left(\frac{h}{c_i}, b_i\right)=1, \quad g b_i^{-m_i} \in F[x],$$

$$g b_i^{-m_i} \equiv h r_i \bmod b_i^{n_i} c_i.$$

(ii) In *LAURENT-TO-STANDARD*, for $1 \leqq i \leqq p$, $g b_i^{-m_i} \in F[x]$, and

$$\deg g < n-d+\sum_{\substack{1 \leqq i \leqq p \\ 0<m_i}} m_i \deg b_i,$$

$$\deg h \leqq d+\sum_{\substack{1 \leqq i \leqq p \\ m_i<0}} |m_i| \deg b_i,$$

$$g b_i^{-m_i} \equiv h r_i \bmod b_i^{n_i+e_i}.$$

Proof. The bounds on the depth and size are clear. For the correctness proof for *STANDARD-TO-LAURENT*, we can assume $g \neq 0$ and $\gcd(g, h) = 1$. Obviously $b_i^{m_i} | g$ if $m_i > 0$. Suppose $q \in F[x]$ is irreducible and divides $\gcd(b_i, h)$, say

$$q^k | b_i,\ q^{k+1} \nmid b_i,\ q^l | h,\ q^{l+1} \nmid h,$$

with $k, l \in \mathbf{N}$ and $k, l > 0$. Then $(-m_i)k \geqq l$ and $q^l | b_i^{-m_i}$, hence $q^l | c_i$ and $q \nmid h/c_i$. Therefore $\gcd(b_i, h_i) = \gcd(b_i, h/c_i) = 1$, and

$$\frac{g b_i^{-m_i}}{c_i} = g_i \equiv h_i r_i = \frac{h}{c_i} r_i \mod b_i^{n_i},$$

using Theorem 3.1. Thus

$$g b_i^{-m_i} \equiv h r_i \mod b_i^{n_i} c_i,$$

and r is a weak Laurent (B, N)-representation of (g, h).

For *LAURENT-TO-STANDARD*, clearly h is monic, and we have

$$\deg g = \deg g' + \sum_{1 \leqq i \leqq p} (m_i + e_i) \deg b_i < n - d + \sum_{\substack{1 \leqq i \leqq p \\ 0 < m_i}} m_i \deg b_i,$$

$$\deg h = \deg h' + \sum_{1 \leqq i \leqq p} e_i \deg b_i \leqq d + \sum_{\substack{1 \leqq i \leqq p \\ m_i < 0}} (|m_i| \deg b_i).$$

If $m_i > 0$, then $e_i = 0$, and therefore $b_i^{m_i} | t$ and $g b_i^{-m_i} \in F[x]$. With $r_i = \sum_{0 \leqq j < n_i} r_{ij} b_i^j$, we have in any case

$$\mu u_i g b_i^{-m_i} = u_i g' t b_i^{-m_i} = u_i g' t_i b_i^{e_i} \equiv u_i h' s_i t_i b_i^{e_i}$$

$$\equiv u_i h' u_i r_i b_i^{e_i} = u h' u_i r_i = \mu h u_i r_i \mod b_i^{n_i + e_i}.$$

Since μu_i is a unit mod $b_i^{n_i + e_i}$, it follows that

$$g b_i^{-m_i} \equiv h r_i \mod b_i^{n_i + e_i}.$$ □

It is clear that Definition 6.1 of "weak Laurent representation" really is too weak. Under this notion, one function may have many representations. If $B = (x)$, $N = (2)$, $g = 1$, $h = x$, so that $g/h = 1/x = x^{-1}(1 + 0 \cdot x)$, then we would like to have $(-1; 1, 0)$ as (B, N)-representation of (g, h), but in fact $(-1; 1, a)$ is a weak Laurent (B, N)-representation of (g, h), for every $a \in F$. Not even the m_i's are uniquely determined: any $r = (m; a, b)$ with $m \leqq -2$ and $a, b \in F$ is a weak Laurent $((x), (2))$-representation of $(1, x^3)$. Even the somewhat stronger congruence in Theorem 6.2(i) does not completely determine the exponent m_i. As an example, take $p = 1$, $B = (b_1)$, $N = (2)$, $g = b_1 g_1$, $h = 1$ with $\deg g_1 < \deg b_1$. Then $c_1 = 1$, and both $(0; 0, g_1)$ and $(1; g_1, 0)$ are weak Laurent (B, N)-representations of (g, h). Thus some more stringent conditions are necessary to ensure uniqueness of representations.

DEFINITION 6.3. Let B be a base, N a precision, r a Laurent (B, N)-representative as in Definition 6.1, and $g, h \in F[x]$ with $h \neq 0$. We say that r is the *Laurent* (B, N)-*representation of* $f = g/h$ if the following conditions are satisfied:

- (L_1) $\gcd(g, h) = 1$.
- (L_2) For all i, $1 \leqq i \leqq p$, set $c_i = \gcd(b_i^{|m_i|}, h)$. Then $g b_i^{-m_i} \in F[x]$, and $\gcd(b_i, h/c_i) = 1$.
- (L_3) $r_{i0} \neq 0$.
- (L_4) For all i, $1 \leqq i \leqq p$,

$$g b_i^{-m_i} \equiv h r_i \mod b_i^{n_i} c_i.$$

Clearly this relation between f and r does not depend on the choice of g, h, since (L_1) determines them up to a scalar factor.

LEMMA 6.4. *Let B, N, r, g, h as in Definition 6.3 satisfy (L_1), (L_2), (L_4). Then (L_3) holds if and only if the following condition holds:*

(L_3') *For all i, $1 \leqq i \leqq p$, either $(\gcd(b_i, h) \neq 1$ and $m_i < 0$ and $\gcd(b_i^{-m_i}, h) = \gcd(b_i^{-m_i+1}, h) \neq \gcd(b_i^{-m_i-1}, h))$ or $(\gcd(b_i, h) = 1$ and $m_i \geqq 0$ and $b_i^{m_i} | g$ and $b_i^{m_i+1} \nmid g)$.*

Proof. Note that $r_{i0} \neq 0$ if and only if b_i does not divide r_i.

"$(L_3) \Rightarrow (L_3')$": Fix some i, $1 \leqq i \leqq p$. If $\gcd(b_i, h) = 1$, then $c_i = 1$ and $m_i \geqq 0$, since otherwise $b_i | g b_i^{-m_i}$ and $r_{i0} = 0$. (L_3') follows in this case. If $\gcd(b_i, h) \neq 1$, then $b_i \nmid g$ and $m_i \leqq 0$ by (L_2). Define $l < 0$ by $\gcd(b_i^{-l}, h) = \gcd(b_i^{-l+1}, h) \neq \gcd(b_i^{-l-1}, h))$. Then (L_2) implies that

$$c_i = \gcd(b_i^{|m_i|}, h) = \gcd(b_i^{|l|}, h)$$

and $m_i \leqq l$. If $m_i < l$, then $b_i | b_i^{-m_i}/c_i$, and (L_4) implies that $b_i | (h/c_i) r_i$, hence $b_i | r_i$. Thus $m_i = l$, and (L_3') follows.

"$(L_3') \Rightarrow (L_3)$": Fix some i, $1 \leqq i \leqq p$. If $m_i \geqq 0$, then $\gcd(b_i, h) = 1$ and $b_i \nmid g b_i^{-m_i}$, hence $b_i \nmid r_i$.

If $m_i < 0$, there exists an irreducible polynomial $q \in F[x]$ with multiplicity $l > 0$ in $c_i = \gcd(b_i^{-m_i}, h)$ such that $q^l \nmid b_i^{-m_i-1}$. Choose some such q and l, and let k be the multiplicity of q in b_i. Then $(-m_i - 1) \cdot k < l$, and the multiplicity of q in $b_i^{-m_i}/c_i$ is $(-m_i) \cdot k - l < k$. Since also $q \nmid g$, we have $q^k \nmid g b_i^{-m_i}/c_i$ and therefore $b_i \nmid (h/c_i) r_i$. □

Note that in Definition 6.3, m_i and c_i do not depend on N, and as in § 2, the Laurent (B, M)-representation for $M \leqq N$ is obtained by truncating the Laurent (B, N)-representation.

In order to compare Definitions 2.1 and 6.3, let B, N be as usual, $g, h \in F[x]$ with $\gcd(g, h) = 1$, $f = g/h \in F(x)$, $r = (m_1, \cdots, m_p; r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1})$ a Laurent (B, N)-representative. We assume $b_i^{m_i} | g$ if $m_i \geqq 0$, and define $c_i = \gcd(b_i^{|m_i|}, h)$ as in (L_2). Let $b = b_1^{n_1} \cdots b_p^{n_p}$, $c = c_1 \cdots c_p$, and $u \in F[x]$ such that

$$\forall i \quad u \prod_{j \neq i} b_j^{-m_j} \equiv 1 \mod b_i^{n_i} c_i, \qquad \deg u < \deg bc.$$

For simplicity, we assume $\gcd(u, h) = 1$. Set $\bar{g} = (gu/c) \prod_{1 \leqq i \leqq p} b_i^{-m_i}$, $\bar{h} = h/c \in F[x]$, $\bar{f} = \bar{g}/\bar{h} \in F(x)$, and $\bar{r} = (r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1}) \in R(B, N)$. Then $\gcd(\bar{g}, \bar{h}) = 1$, and we have:

1. r is a weak Laurent (B, N)-representation of (g, h) and (L_4) holds
   $\Leftrightarrow \bar{r}$ is a weak (B, N)-representation of $(\bar{g}, \bar{h})$,
2. r is the Laurent (B, N)-representation of f
   $\Leftrightarrow \bar{r}$ is a (B, N)-representation of $\bar{f}$ and (L_3') holds.

THEOREM 6.5. *Let B be a base, N a precision for B, and $f \in F(x)$. Then:*

(i) *f has a unique Laurent (B, N)-representation.*

(ii) *Given $g, h \in F[x]$ with $f = g/h$ and $\gcd(g, h) = 1$, algorithm STANDARD-TO-LAURENT computes the Laurent (B, N)-representation of f.*

Proof. (i) For an arbitrary $f \in F(x)$, write $f = g/h$ with $g, h \in F[x]$ and $\gcd(g, h) = 1$. Let r be the weak Laurent (B, N)-representation of (g, h) computed by algorithm *STANDARD-TO-LAURENT* with input B, N, g, h. Theorem 6.2(i) guarantees that (L_1), (L_2), (L_3'), (L_4) hold, hence r is the Laurent (B, N)-representation of f. This also proves claim (ii).

For the uniqueness, let $r^{(1)}$ and $r^{(2)}$ be two Laurent (B, N)-representations of $f = g/h$. Fix some i, $1 \leqq i \leqq p$. First note that the value of m_i is determined by the

condition (L_3'). As usual, we write $r_i^{(k)} = \sum_{0 \leqq j < n_i} r_{ij}^{(k)} b_i^j$ for $k = 1, 2$. Then

$$hr_i^{(1)} \equiv gb_i^{-m_i} \equiv hr_i^{(2)} \mod b_i^{n_i} c_i$$

by (L_4), and hence

$$\frac{h}{c_i} r_i^{(1)} \equiv \frac{h}{c_i} r_i^{(2)} \mod b_i^{n_i},$$

$$r_i^{(1)} \equiv r_i^{(2)} \mod b_i^{n_i},$$

since $h/c_i \in F[x]$ is a unit mod $b_i^{n_i}$ (using (L_2)). Since $\deg r_i^{(k)} < \deg b_i^{n_i}$, it follows that $r_i^{(1)} = r_i^{(2)}$, hence $r^{(1)} = r^{(2)}$. □

Next we want to prove that the functions computed by the two conversion algorithms are inverses of each other. It turns out that, unlike in Theorem 4.3, we have to impose a condition on the base to guarantee this.

THEOREM 6.6. *Let $B = (b_1, \cdots, b_p)$ be a base. The functions computed by algorithms STANDARD-TO-LAURENT and LAURENT-TO-STANDARD are inverses of each other if and only if each b_i is irreducible.*

Proof. To be more precise, we claim the following for any base B. Let N be a precision for B with total precision n, $0 \leqq d < n$, $R_N \subseteq \mathbf{Z}^p \times F^n$ the set of Laurent (B, N)-representatives. By Theorem 6.5, we have a mapping $\rho_N : F(x) \to R_N$, which associates to each $f \in F(x)$ the unique Laurent (B, N)-representation $\rho_N(f) \in R_N$ of f. Let

$$T_{B,N,d} = \Bigg\{ (g, h) \in F[x]^2 : \gcd(g, h) = 1,\ h \neq 0 \text{ is monic},$$
$$\rho_N\left(\frac{g}{h}\right) = (m_1, \cdots, m_p; r_{10}, \cdots, r_{1,n_1-1}; \cdots; r_{p0}, \cdots, r_{p,n_p-1}) \in R_N,$$
$$\deg g < n - d + \sum_{0 < m_i} m_i \deg b_i,\ \deg h \leqq d + \sum_{m_i < 0} |m_i| \deg b_i \Bigg\}.$$

The bounds on the degrees of g and h are the same as those for the output of *LAURENT-TO-STANDARD*. From Theorem 6.5 it is clear that *STANDARD-TO-LAURENT* maps $(g, h) \in T_{B,N,d}$ to $\rho_N(g/h) \in R_N$. Let

$$U_{B,N,d} = \rho_N\left(\left\{f : \exists (g, h) \in T_{B,N,d} \quad f = \frac{g}{h}\right\}\right) \subseteq R_N.$$

For $(g, h) \in T_{B,N,d}$, we can execute algorithm *STANDARD-TO-LAURENT* with input (B, N, g, h) to get output $r = \rho_N(g/h) \in U_{B,N,d}$. On the other hand, for $r \in U_{B,N,d}$, we can execute algorithm *LAURENT-TO-STANDARD* with input (B, N, d, r) to get output (g, h). The claim of the theorem is that the two conversion algorithms give inverse bijections between $T_{B,N,d}$ and $U_{B,N,d}$ for all N and d if and only if each b_i is irreducible.

For the implication "$\Rightarrow$", we can assume that $b_1 = vw$ with $v, w \in F[x]$ non-constant. Choose $N = (2, 0, \cdots, 0)$ and $d = 1 + \deg v$. With input $g = 1$, $h = v$, *STANDARD-TO-LAURENT* will produce $r = (-1, 0, \cdots, 0; w, 0)$, and *LAURENT-TO-STANDARD* will yield $(w, b_1) \neq (1, v)$.

For the implication "$\Leftarrow$", it is sufficient to show that for any B, N, d, $(g, h) \in T_{B,N,d}$ and $r = \rho_N(g/h)$, algorithm *LAURENT-TO-STANDARD* with input (B, N, d, r) computes (g, h). We use the notation of the algorithm, and set

$$\bar{g} = \mu g t^{-1}, \qquad \bar{h} = \mu h u^{-1}.$$

Our goal is to show that $\bar{g}, \bar{h}$ are equal to g', h' as computed in step 3 of *LAURENT-TO-STANDARD*. First note that $\bar{g}, \bar{h} \in F[x]$: If $m_i \geqq 0$, then $e_i = 0$ and $b_i^{m_i} | g$ by condition (L_2). If $m_i < 0$, then $m_i + e_i = 0$ and $\gcd(b_i^{-m_i}, h) \neq \gcd(b_i^{-m_i-1}, h)$. Since b_i is irreducible, it follows that $\gcd(b_i^{-m_i}, h) = b_i^{-m_i}$, and $b_i^{e_i} | h$. Therefore $u | h$, and thus $\bar{g}, \bar{h} \in F[x]$. Now for any i, $1 \leqq i \leqq p$, we have $c_i = b_i^{e_i} = \gcd(b_i^{|m_i|}, h)$, and

$$gb_i^{-m_i} \equiv hr_i \bmod b_i^{n_i+e_i},$$

$$u_i t_i b_i^{e_i} \bar{g} = \mu u_i g b_i^{-m_i} \equiv \mu u_i h r_i \equiv \mu h s_i t_i = \bar{h} u s_i t_i = u_i t_i b_i^{e_i} \bar{h} s_i \bmod b_i^{n_i+e_i}.$$

Since u_i and t_i are units mod $b_i^{n_i+e_i}$, it follows that

$$\bar{g} \equiv \bar{h} s_i \bmod b_i^{n_i}. \tag{1}$$

As to the degrees, we have

$$\begin{aligned}
\deg \bar{g} &= \deg g + \sum_{1 \leqq i \leqq p} (-m_i - e_i) \deg b_i \\
&< n - d + \sum_{0 < m_i} m_i \deg b_i + \sum_{0 < m_i} (-m_i \deg b_i) = n - d, \\
\deg \bar{h} &= \deg h + \sum_{1 \leqq i \leqq p} (-e_i \deg b_i) \\
&\leqq d + \sum_{m_i < 0} (|m_i| \deg b_i) + \sum_{m_i < 0} (-|m_i| \deg b_i) = d,
\end{aligned}$$

since $(g, h) \in T_{B,N,d}$. Now these degree inequalities and (1) also hold for g', h' as computed in *LAURENT-TO-STANDARD*. These were obtained as scalar multiples of certain entries a_k, t_k of the Extended Euclidean Scheme of $(b_1^{n_1} \cdots b_p^{n_p}, w)$, with h' being monic, and $w \equiv s_i \bmod b_i^{n_i}$ for all i. Just as for Theorem 4.3, we now use the uniqueness property of the Extended Euclidean Scheme to get $m \in \{1, \cdots, l\}$ and $v \in F[x]$ such that $\bar{g} = v a_m$, $\bar{h} = v t_m$. Again the inequalities for the degrees force $m = k$, and $\gcd(\bar{g}, \bar{h}) = 1$ implies $\bar{g} = g'$, $\bar{h} = h'$. Therefore the polynomials $\mu^{-1} g' t$ and $\mu^{-1} h' u$ computed by *LAURENT-TO-STANDARD* equal g and h. □

Remark 6.7. The "counterexample" $b_1 = uv$, $g = 1$, $h = u$, where $(v, b_1) \neq (g, h)$ is returned by *LAURENT-TO-STANDARD*, is of course not very convincing, since $v / b_1 = g / h$. The obvious remedy—returning $(g/a, h/a)$ in step 4 of *LAURENT-TO-STANDARD*, with $a = \gcd(g, h)$—does not work as expected. It may happen that $a' = \gcd(g', h') \neq 1$, and that

$$\frac{g'}{a'} \not\equiv \frac{h'}{a'} s_i \bmod b_i^{n_i};$$

see §§ 4 and 5 for examples.

Remark 6.8. Do we gain greater generality by dropping the requirement that the base polynomials $b_1, \cdots, b_p$ be relatively prime? The answer is no, not really. For simplicity, we consider in the following $b_i^{n_i}$ as one of the base polynomials (rather than b_i), and thus assume that all exponents n_i are 1. Suppose that $c = \gcd(b_1, b_2) \neq 1$. Then clearly for a representation $r = (r_1; r_2; \cdots)$ of $f \in F(x)$, r_1 and r_2 have to agree mod c. Continuing this process of replacing (b_i, b_j) by $(b_i/c, b_j/c, c)$ if $c = \gcd(b_i, b_j) \neq 1$, one arrives at a pairwise relatively prime basis $c_1, \cdots, c_q$. (Termination is guaranteed by the fact that the sum of the degrees decreases with each such step.) Each c_k is in the "gcd-closure" of $b_1, \cdots, b_p$ (as defined by this process), and there exist exponents $e_{ik} \geqq 0$ such that $b_i = \prod_{1 \leqq k \leqq q} c_k^{e_{ik}}$ for all i. If $a_1, \cdots, a_s$ are the distinct monic irreducible polynomials dividing $b_1 \cdots b_p$, and d_l is the smallest positive multiplicity of a_l in

$b_1, \cdots, b_p$, then each c_k is the product of some $a_l^{d_l}$. In fact, if $S \subseteq \{1, \cdots, s\}$ is nonempty and such that

$$\forall l, m \in S \quad \forall i \leqq p \quad (a_l^{d_l} | b_i \Rightarrow a_m^{d_m} | b_i)$$

and maximal with this property, then $\prod_{l \in S} a_l^{d_l}$ is one of the c_k's. Conversely, every c_k is of this form.

For $i, j \leqq p$, $k \leqq q$, let $u_{ijk} = \min\{e_{ik}, e_{jk}\}$. Then for any $r_1, \cdots, r_p \in F[x]$ we have

$$\exists f \in F[x] \quad \forall i \leqq p \quad f \equiv r_i \bmod b_i \Leftrightarrow \forall i, j, k \quad r_i \equiv r_j \bmod c_k^{u_{ijk}}.$$

(For "$\Leftarrow$", simply interpolate $r_{i_k} \bmod c_k^{e_{i_k,k}}$, $1 \leqq k \leqq q$, with $e_{i_k,k} = \max\{e_{jk} : 1 \leqq j \leqq p\}$.)

It is, however, not clear how to calculate $c_1, \cdots, c_q$ fast in parallel from $b_1, \cdots, b_p$. With a "logarithmic" look at exponents of the a_l's, this problem is related to the following: Given $f_i = (f_{i1}, \cdots, f_{is}) \in \mathbf{N}^s$ for $1 \leqq i \leqq p$, use addition and the coordinate-wise minimum of vectors in $\mathbf{N}^s$ to compute an "orthogonal" basis $g_1, \cdots, g_q \in \mathbf{N}^s$ such that

$$\forall i \quad f_i \in \sum_{1 \leqq k \leqq q} g_k \mathbf{N},$$

$$\forall k \neq m \quad \min(g_k, g_m) = (0, \cdots, 0),$$

$$\forall k \quad g_k \in \text{min-closure of } f_1, \cdots, f_p.$$

(Of course the algorithm would not know the coordinates of the input vectors f_i.)

Remark 6.9. The concepts presented in this paper obviously apply to more general rings than $F[x]$, e.g. a Euclidean valuation ring R, where one has a valuation w and a division-with-remainder property with respect to w (see von zur Gathen [1984a, § 4]). The two prominent examples are our case $R = F[x]$ with $w(f) = 2^{\deg f}$, and $R = \mathbf{Z}$ with $w(a) = |a|$. The "interpolation problem" can be phrased as a "simultaneous approximation problem": finding $g, h \in R$ such that $g \equiv hr_i \bmod b_i^{n_i}$ corresponds to simultaneously approximating each r_i in the b_i-adic valuation with precision n_i (if each b_i is irreducible); the degree condition for polynomials translates into upper bounds for $w(g)$ and $w(h)$. For the general problem, we would be given further valuations $v_1, \cdots, v_p$ on R, precisions $\delta_1, \delta_2, \varepsilon_1, \cdots, \varepsilon_p \in \mathbf{R}$, and $r_1, \cdots, r_p \in R$, and want to compute $g, h \in R$ such that

$$v_i(g - hr_i) \leqq \varepsilon_i,$$

$$w(g) \leqq \delta_1, \qquad w(h) \leqq \delta_2.$$

This question of simultaneous approximation with respect to various valuations subsumes among others the usual Chinese remainder computations, solution of congruences by rational numbers (Miola [1982]), and Padé approximation of power series. If we consider for v_i the absolute value on $R = \mathbf{Z}$ instead of a b_i-adic valuation (and allow $r_i \in \mathbf{R}$), then it also subsumes approximation of a real number by rational numbers.

As of now, no $(\log n)^{O(1)}$ parallel computation for integer Chinese remaindering or the gcd of two n-bit integers is known. But even for sequential algorithms, it would be interesting to have a general approximation algorithm that solves all the above problems.

REFERENCES

G. A. BAKER AND P. GRAVES-MORRIS [1981], *Padé approximants*, Encyclopedia of Mathematics and Its Applications, vols. 13 and 14, Addison-Wesley, Reading, MA.

S. J. BERKOWITZ [1984], *On computing the determinant in small parallel time using a small number of processors*, Inform. Proc. Letters, 18, pp. 147-150.

A. BORODIN, S. COOK AND N. PIPPENGER [1983], *Parallel computation for well-endowed rings and space-bounded probabilistic machines*, Inform. and Control, 58, pp. 96-114.

A. BORODIN, J. VON ZUR GATHEN AND J. HOPCROFT [1982], *Fast parallel matrix and GCD computations*, Inform. and Control, 52, pp. 241-256.

R. P. BRENT, F. G. GUSTAVSON AND D. Y. Y. YUN [1980], *Fast solution of Toeplitz systems of equations and computation of Padé approximants*, J. Algorithms, 1, pp. 259-295.

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

A. CAUCHY [1821], *Cours d'analyse de l'école royale polytechnique (Analyse algébrique)*, in Oeuvres complètes, IIe série, tôme III, pp. 429-433.

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

L. CSANKY [1976], *Fast parallel matrix inversion algorithms*, this Journal, 5, pp. 618-623.

W. EBERLY [1984], *Very fast parallel matrix and polynomial arithmetic*, Proc. 25th Annual IEEE Symposium on Foundations of Computer Science, Singer Island FL, pp. 21-30.

J. VON ZUR GATHEN [1984a], *Hensel and Newton methods in valuation rings*, Math. Comp., 42, pp. 637-661.

——— [1984b], *Parallel algorithms for algebraic problems*, this Journal, 13 (1984), pp. 802-824.

K. O. GEDDES [1979], *Symbolic computation of Padé approximants*, ACM Trans. Math. Software, 5, pp. 218-233.

W. B. GRAGG [1972], *The Padé table and its relation to certain algorithms of numerical analysis*, SIAM Rev., 14, pp. 1-62.

P. R. GRAVES-MORRIS [1980], *Efficient reliable rational interpolation*, Proc. Conf. Padé and Rational Approximation, Theory and Applications, Amsterdam, pp. 28-63.

F. G. GUSTAVSON AND D. Y. Y. YUN [1979], *Fast algorithms for rational Hermite approximation and solution of Toeplitz systems*, IEEE Trans. Circuits and Systems, 26, pp. 750-755.

C. G. J. JACOBI [1846], *Ueber die Darstellung einer Reihe gegebner Werthe durch eine gebrochne rationale Function*, J. Reine Angew. Math., 30, pp. 127-156.

D. E. KNUTH [1981], *The Art of Computer Programming*, Vol. 2, 2nd ed., Addison-Wesley, Reading, MA.

E. KRONECKER [1881], *Zur Theorie der Elimination einer Variabeln aus zwei algebraischen Gleichungen*, Monatsberichte der Akademie der Wissenschaften, Berlin, pp. 535-600.

A. M. MIOLA [1982], *The conversion of Hensel codes to their rational equivalents*, SIGSAM Bull, 16, pp. 24-26.

H. PADÉ [1892], *Sur la représentation approchée d'une fonction par des fractions rationnelles*, Annales Scientifiques de l'Ecole Normale Supérieure, 3e série, 9, Supplément S3-S93.

J. REIF [1983], *Logarithmic depth circuits for algebraic functions*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, Tucson AZ, pp. 138-145.

V. STRASSEN [1974], *Some results in algebraic complexity theory*, Proc. International Congress of Mathematicians, Vancouver, pp. 497-501.

——— [1983], *The computational complexity of continued fractions*, this Journal, 12, pp. 1-27.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
009

ON THE COMPLEXITY OF NONCONVEX COVERING*

WOLFGANG MAASS†

Abstract. We study the problem of covering given points in Euclidean space with a minimum number of nonconvex objects of a given type. We concentrate on the one-dimensional case of this problem, whose computational complexity was previously unknown. We define a natural measure for the "degree of nonconvexity" of a nonconvex object. Our results show that for any fixed bound on the degree of nonconvexity of the covering objects the one-dimensional nonconvex covering problem can be solved in polynomial time. On the other hand without such bound on the degree of nonconvexity the one-dimensional nonconvex covering problem is NP-complete. We also consider the capacitated version of the nonconvex covering problem and we exhibit a useful property of minimum coverings by objects whose degree of nonconvexity is low.

Key words. NP-completeness, computational geometry, covering, nonconvexity, polynomial time algorithm, robotics

AMS(MOS) subject classifications. 03D15, 68C25

1. Introduction. In this paper we study the problem of covering given points in Euclidean space with a minimum number of nonconvex objects of a given type. We concentrate on the one-dimensional case of this problem, whose computational complexity was previously unknown. We further restrict our attention to rings—arguably the simplest nonconvex objects (it is not difficult to extend our algorithms to other types of nonconvex objects).

A number of researchers (see the discussion and references in Johnson [6, p. 185]) have shown that the following problem is NP-complete: Decide whether n given points in the Euclidean plane can be covered by k discs of a given radius w. We now look at a nonconvex variation of this problem. In the following, a ring (or annulus) of size $\langle r, w\rangle$ is the set of points that are enclosed by two concentric circles of radius r respectively $r+w$ (r and w will always be nonnegative integers in this paper). If we substitute in the two-dimensional covering problem the discs by rings of given size $\langle r, w\rangle$, the resulting nonconvex covering problem is still in NP. Thus the extension to nonconvex covering objects (rings) does not change the computational complexity of the problem: In *two* dimensions both the convex covering problem (with discs) and the nonconvex covering problem (with rings) are NP-complete.

In contrast to the preceding observation we show in this paper that in the one-dimensional case significant differences arise between the complexity of the convex and the nonconvex covering problem. In the one-dimensional case we assume that n points on a line are given. We assume that the covering rings have their centers on the same line. Thus the intersection of a ring of size $\langle r, w\rangle$ with this line consists of two closed intervals of length w which are separated by an (open) interval of length $2r$ in between. In the following discussion we will refer to such a pair of intervals as a "one-dimensional ring of size $\langle r, w\rangle$". The *one-dimensional ring cover problem* is the problem of computing for n given points on a line the positions of a minimum number of one-dimensional rings of given size $\langle r, w\rangle$ so that all given points are covered. This

* Received by the editors December 29, 1982, and in revised form, July 1, 1984. During the preparation of this paper the author was supported by the Heisenberg Program of the Deutsche Forschungsgemeinschaft, Bonn.

† Department of Mathematics and Computer Science Division, University of California, Berkeley, California 94720. Current address, Department of Mathematics, Statistics and Computer Science, University of Illinois, Chicago, Illinois 60680.

problem contains, for $r=0$ as a special case, the one-dimensional *convex* covering problem. A very simple algorithm (see the beginning of § 3) shows that this one-dimensional convex covering problem can be solved in linear time. In contrast to this we show in § 2 that the one-dimensional ring cover problem is strongly NP-complete. This intractability result comes somewhat unexpectedly insofar as almost all geometrical problems become tractable when they are restricted to one dimension.

In § 3 we close the gap between the previously mentioned two results. We identify the quotient r/w (which may be viewed as a natural measure for the degree of nonconvexity of a ring of size $\langle r, w\rangle$) as the key parameter that determines the complexity of the problem of covering with rings of size $\langle r, w\rangle$. Theorem 3.1 shows that not only for $r/w=0$ (convex case) but also for any fixed bound on this parameter r/w the corresponding one-dimensional ring cover problem can be solved in polynomial time. In § 4 we exhibit in addition a threshold for this parameter r/w (at $r/w=\frac{1}{2}$) where qualitative changes in the structure of minimum coverings by rings of size $\langle r, w\rangle$ take place.

Finally we consider in § 5 a capacitated version of the one-dimensional ring cover problem. We assume here that the number of points that may be "served" by each ring is bounded by a given capacity b. We show that for any fixed bounds on b and r/w this problem is also in P.

It is obvious that our algorithms can be generalized to cases where one covers with other nonconvex one-dimensional objects. We mention further generalizations at the end of § 3.

Finally we would like to mention two possible practical applications of the considered problems. In scheduling theory one may interpret the line as a time axis on which particular time points are given. If resources (for example work shifts) are to be scheduled so that all given time points are covered, one arrives at a one-dimensional covering problem. In certain realistic models where resources are only intermittently available (for example due to lunch breaks for workers or preventive maintenance of machines) this covering problem is nonconvex. For example one covers with one-dimensional rings of size $\langle \frac{1}{2}, 4\rangle$ if every eight-hour work shift is interrupted by a one-hour break in the middle. We refer to Bartholdi III [1] for a further discussion of this application.

There are other possible applications if one interprets the considered line as a line in space. We would like to mention two examples from robotics. In this area one might want to cover given points in space by certain geometrical objects that model the set of points which are reachable by the arm of a robot (for a fixed position of the base of the robot). This set of reachable points may be nonconvex because of imperfections in the robot arm (even for the human arm this set of reachable points forms a ring-like structure). Thus if one wants to compute for a given set of points in space the positions for a minimum number of robot arms so that all points can be reached by some robot arm, one arrives at a (convex or nonconvex) covering problem. Alternatively one might want to compute for one mobile robot a tour where each of a number of given points can be reached by the arm of the robot from some stop of the (base of the) robot. If the goal is to minimize the number of stops for the (base of the) robot, the same covering problem as before arises.

The results of this paper serve as a basis for a series of subsequent papers with Dorit Hochbaum, where we design polynomial time approximation schemes and fast approximation algorithms for one-dimensional and higher-dimensional covering problems [3]–[5].

2. Nonconvex covering in one dimension is NP-complete. We refer to the first section for a definition of the ring cover problem.

THEOREM 2.1. *The one-dimensional ring cover problem is strongly* NP-*complete.*

Proof. We first note that the considered problem is in NP. Here one uses the fact that it is sufficient to consider positions of rings where one of the four endpoints of the (one-dimensional) ring coincides with one of the given points.

In order to show that the considered problem is NP-*complete* we construct a polynomial time computable reduction from the NP-complete problem 3SAT (see Garey and Johnson [2]). The strategy is somewhat similar to the strategy of the reduction from 3SAT to 3-DIMENSIONAL MATCHING. However one has to work harder to construct suitable problem instances in only one dimension.

Before we define the desired reduction, we introduce an essential tool for the construction of suitable instances of the one-dimensional ring cover problem. This tool makes it possible to interconnect the coverability properties of various different clusters of points in the constructed instances. Consider a sequence $W_1, \cdots, W_{2k}$ of points on the line that are spaced $2r+w$ apart. For example assume that W_i has the coordinate $i \cdot (2r+w)$. Obviously one can cover all points in this sequence with k rings of size $\langle r, w\rangle$: The ith ring covers W_{2i-1} and W_{2i}. If W_1 does not need to be covered by the considered k rings (because it is already covered by some other ring), we can shift the k rings over a distance $2r+w$ to the right. In this case the ith ring covers W_{2i} and W_{2i+1}. Further the kth ring covers only one point: W_{2k}. Therefore we can use the other w-interval of the kth ring to cover some other point in the neighborhood of W_{2k}. Thus we see that a covering advantage at the beginning W_1 of the sequence causes a chain reaction in the covering of the sequence $W_1, \cdots, W_{2k}$ (the possibility of shifting all k rings to the right), which leads to a covering advantage at the last point W_{2k}: the kth ring has one interval free. In this sense the sequence $W_1, \cdots, W_{2k}$ can transmit covering advantages and therefore we call it a "wire". In the first situation (where the kth ring has to cover W_{2k-1} and W_{2k}), we say that the wire transmits the "signal 0". In the second situation where the kth ring only has to cover the last point W_{2k}, we say that the wire transmits the "signal 1".

So far we have made no use of the nonconvexity of the covering objects. Everything we have said remains true if we cover with (convex) intervals of length $2r+2w$ instead of rings. We now show that the nonconvexity of the covering objects allows us to run several wires in parallel, so that each can transmit a signal 0 or 1 without mutual interference. It turns out that the number of wires that we can run in parallel is proportional to r/w (this is the first indication of the importance of the parameter r/w for the complexity of the ring cover problem). Consider a second wire $V_1, \cdots, V_{2k}$ where point V_i has coordinate $i \cdot (2r+w)+p$. The "phase shift" p that occurs here is some integer with $w < p < 2r$. Obviously the wire $V_1, \cdots, V_{2k}$ has the same covering properties as the first wire $W_1, \cdots, W_{2k}$. But in addition the choice of the phase shift p guarantees that no ring can cover two points that belong to different ones of these two wires. This implies that the choice of a covering of one of the two wires has no consequence for the covering of the other wire. In the previously introduced terminology we can say that both wires can transmit a signal 0 or 1 without mutual interference. In the same way we can run d wires in parallel (for any natural number $d \leqq r/w$) that transmit signals 0 or 1 without mutual interference. We merely have to choose for the d wires phase shifts $p_1, \cdots, p_d$ so that for any $i \neq j$ we have $w < |p_i - p_j| < 2r$.

We now construct the desired reduction from 3SAT. Let F be an arbitrary instance of 3SAT, let $U = \{u_1, \cdots, u_n\}$ be the set of variables in F and let $C = \{c_1, \cdots, c_m\}$ be

the set of clauses in F. Each clause c_j is a disjunction of up to three (negated or unnegated) variables from U. The conjunction of all clauses in C yields the considered formula F. In the following we construct in polynomial time a set P of integers (which are interpreted as coordinates for points on a line) and integers r, w, M such that all points in P can be covered by M rings of size $\langle r, w\rangle$ if and only if F is satisfiable. The points in P fall into four classes of components according to their intended function: "truth-setting", "satisfaction testing", "wire" and "wire crossing".

Each truth setting component corresponds to a single variable $u_i \in U$ (we call it the u_i-component for this variable u_i). The points of each u_i-component can be covered in exactly two different ways by minimum coverings by rings of size $\langle r, w\rangle$. In this way each u_i-component forces any minimum covering of all points in P to make a choice between these two possible coverings of the u_i-component. This choice corresponds to the choice between setting $u_i = \text{true}$ or $u_i = \text{false}$ in a truth assignment to all variables in U.

Each satisfaction testing component in P corresponds to a single clause $c_j \in C$ (therefore we call it the c_j-component for this c_j). It is connected by up to three wires to those three or less u_i-components for which u_i or $\bar{u}_i$ occur in the clause c_j. The number M of rings that may be used for a covering of P will be chosen so small that a c_j-component can only be covered if one of the three wires transmits the signal 1 to the c_j-component. In this case the last ring that is used for the covering of this wire can use its "free" w-interval to cover the c_j-component (while its other w-interval covers the last point of that wire). According to this plan we just have to make sure that the wire from a u_i-component to this c_j-component transmits the signal 1 if and only if the chosen covering of the u_i-component corresponds to setting $u_i = \text{true}$ (in case that u_i occurs in c_j), respectively, to setting $u_i = \text{false}$ (in case that $\bar{u}_i$ occurs in c_j).

We set $w = 10$ and $r = 100w \cdot (4m + n)$. According to our outline up to $3m$ wires are needed. We fix a numbering of these wires and we reserve for the kth wire the "track" with phase shift $p_k = 100w \cdot k$. In general all points in P with a coordinate z such that $z \equiv p_k \bmod (2r + w)$ will belong to the kth wire (the only exceptions are points from "crossing components" that will be discussed below). For each u_i-component we reserve a track with phase shift $100w \cdot (3m + i)$. Each point in the u_i-component will have the property that it is within $3w$ of the u_i-track. Finally each c_j-component consists of a single point y such that $y \equiv 100 \cdot (3m + n + j) \bmod (2r + w)$.

We have now assigned to each wire, u_i-component and c_j-component a separate "track". No ring of size $\langle r, w\rangle$ can cover points that belong to two different tracks. Therefore the coverings of the different components are mutually independent, except for those pairs of components where we force a dependency via a wire. Such a wire connecting a u_i-component with a c_j-component begins on the track of the u_i-component (this means that the first points of the wire have the same phase shift as the u_i-component). Then it moves to its assigned track (see the assignment above) and stays on this track until the end, when it moves to the track of the c_j-component. In order to move a wire from one track to another, we use the fact that the points of a wire need not necessarily be spaced $2r + w$ apart. If we choose instead some distance $2r + w + d$ with $d \in [-w, +w]$ between successive points of a wire, the covering properties of the wire remain unchanged. However for $d < 0$ the wire moves towards a track with a smaller phase shift p. If we use this distance several times in the wire, the wire can reach in this way any other track. Similarly if we choose $d > 0$ the wire moves towards a track with a larger phase shift p.

If a wire leaves its assigned track and approaches some track that has been assigned to some other wire, the coverings of both wires may interfere. In order to avoid this

we use in these situations a special "crossing component" that allows a wire to cross the track of some other wire without interference.

Figure 1 provides a global picture of the construction for the case of the formula $F \equiv (u_1 \vee u_2) \wedge (\bar{u}_1 \vee u_2) \wedge (u_1 \vee u_3)$. The horizontal dimension of the diagram represents the actual line on which the points of P are located. The vertical dimension of the diagram is used to indicate the different tracks. Of course in reality all these tracks run along the same line (but with different phase shifts).

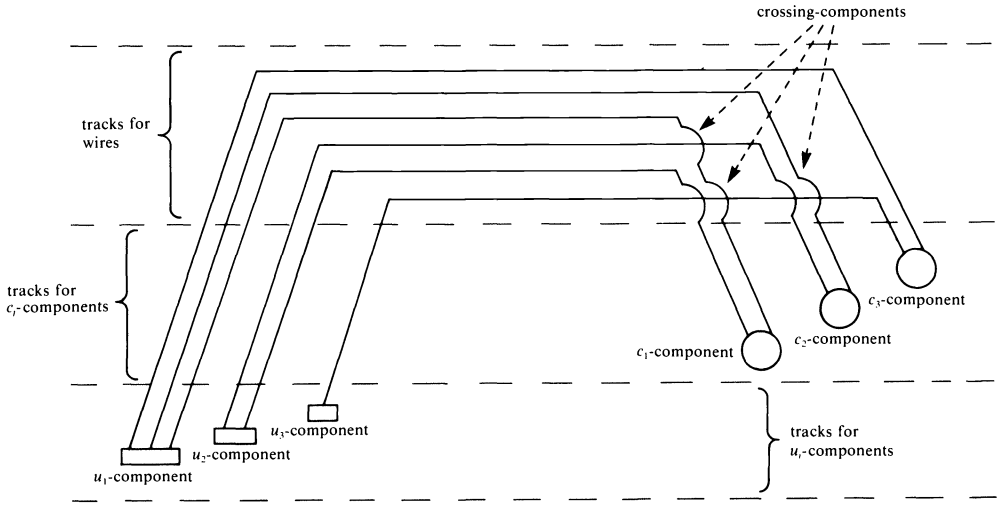
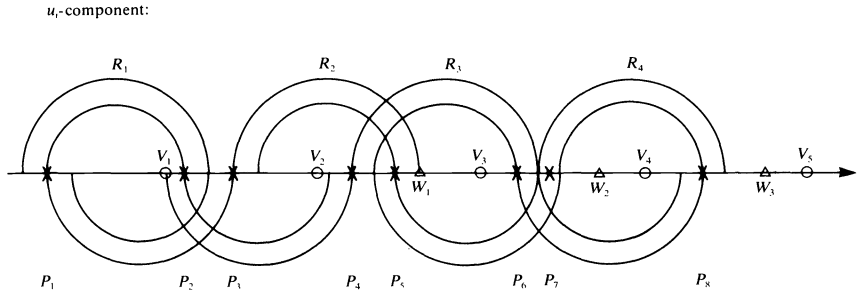


FIG. 1

We will now look at the design of u_i-components, c_j-components and crossing components in more detail.

Figure 2 shows a u_i-component for the simple case where u_i or $\bar{u}_i$ occur only in two clauses, say u_i occurs in c_1 and $\bar{u}_i$ in c_2. The u_i-component consists of the points $P_1, \cdots, P_8$, whose coordinates are also given in Fig. 2. The figure shows in addition



$P_1 = z$
$P_2 = z + 2r$
$P_3 = z + 2r + 2w$
$P_4 = z + 4r + w$
$P_5 = z + 4r + 3w$
$P_6 = z + 6r + 2w$
$P_7 = z + 6r + \frac{7}{2}w$
$P_8 = z + 8r + 4w$

$V_1 = z + 2r - w$
$V_2 = z + 4r - \frac{1}{2}w$
$V_3 = z + 6r + \frac{1}{2}w$
$V_4 = z + 8r + \frac{3}{2}w$
$V_5 = z + 10r + \frac{5}{2}w$

$W_1 = z + 4r + 4w$
$W_2 = z + 6r + \frac{11}{2}w$
$W_3 = z + 8r + \frac{13}{2}w$

FIG. 2

the initial segments of two wires that are attached to this u_i-component: Wire $W_1, W_2, \cdots$ leads to the c_1-component and wire $V_1, V_2, \cdots$ leads to the c_2-component. These two wires are not attached in the same way to the u_i-component. By assumption variable u_i occurs positively in c_1, therefore, the wire to the c_1-component has to transmit the signal 1 to the c_1-component if and only if the chosen minimum covering of the u_i-component corresponds to setting $u_i = \text{true}$. This means that the rings of the corresponding minimum covering of the u_i-component have to be able to cover in addition also the first point W_1 of this wire. Similarly only a covering of the u_i-component that corresponds to setting $u_i = \text{false}$ should be able to cover in addition V_1 (whereas it cannot cover in addition W_1), since u_i occurs negatively in c_2. In the total number M of allowed rings in the covering exactly four rings would be allocated to the covering of the points $P_1, \cdots, P_8$ of the u_i-component of Fig. 2. The previously mentioned two different minimum coverings of the u_i-component (that correspond to setting $u_i = \text{true}$ respectively $u_i = \text{false}$) are indicated in the upper respectively lower half of Fig. 2. The coordinate z of the point P_1 satisfies $z \equiv 100w \cdot (3m+i) \bmod (2r+w)$ (according to our earlier assignment of tracks).

In order to verify that the u_i-component in Fig. 2 has the desired properties, we consider any covering of the points $P_1, \cdots, P_8$ by four rings R_1, R_2, R_3, R_4 (numbered according to their location from left to right) of size $\langle r, w \rangle$. Two w-intervals from two different ones of these rings are needed to cover the points P_2 and P_3 because $w < |P_2 - P_3| < 2r$. The same fact holds for the pairs P_4, P_5 and P_6, P_7. Together this implies that P_1 is covered by the left interval of R_1 and P_8 is covered by the right interval of R_4. Further P_2, P_3 (P_4, P_5; P_6, P_7) are covered by the right interval of R_1 (R_2; R_3) together with the left interval of R_2 (R_3; R_4).

It is obvious that no ring R_i can cover with one w-interval both a point P_j and a point from $\{V_2, \cdots, V_5\}$ or $\{W_2, W_3, W_4\}$ (because all these wire points have distance bigger than w from every point P_j).

Finally assume that (like in the top half of Fig. 2) the point P_2 is covered by R_1. This implies that P_3 is covered by the left interval of R_2. Therefore the right interval of R_2 does not cover P_4 because $|P_3 - P_4| = 2r - w < 2r$. Thus P_4 is covered by R_3. From this we conclude that R_3 does not cover P_7 (since $|P_4 - P_7| > 2r + 2w$). Therefore R_3 covers P_6 and R_4 covers P_7. Altogether we see that the initial assumption that P_2 is covered by R_1 forces a structure of the covering where all points P_j are covered by the same rings as in the top half of Fig. 2. Further we see that in this case V_1 cannot be covered by $R_1, \cdots, R_4$: R_1 cannot cover V_1 because R_1 covers P_1 and $|P_1 - V_1| < 2r$; R_2 cannot cover V_1 because R_2 covers P_3 and $|V_1 - P_3| > w$.

Analogously one shows that in the case where R_1 covers P_3 (instead of P_2), all points P_i are covered by those rings that cover them in the bottom half of Fig. 2. In this case W_1 cannot be covered by $R_1, \cdots, R_4$ (the argument is the same as for V_1 in the previous case).

In the general case more wires may have to be attached to a u_i-component. Then, instead of just three pairs (P_2, P_3), (P_4, P_5), (P_6, P_7) one has to use a correspondingly larger number of pairs (P_{2k}, P_{2k+1}) with $|P_{2k} - P_{2k+1}| = 2w$ and $|P_{2k} - P_{2k+2}| = 2r + w$ (a few of these distances are changed slightly as described below). For all wires that lead to c_j-components such that u_i occurs positively in c_j one positions the first point W of such wire to the right of a pair of points (P_{2k}, P_{2k+1}) (like point W_1 in Fig. 2). One moves in this case the point P_{2k+3} over a distance $w/2$ to the left (like point P_7 in Fig. 2) from its previously indicated position. This small shift ensures that no ring covers both W and P_{2k+3}. Similarly for all wires that lead to a c_j-component such that $\bar{u}_i$ occurs in c_j, one positions the first point V of such wires to the left of a pair of points

(P_{2k}, P_{2k+1}). In this case one shifts the point $P_{2(k-1)}$ over distance $w/2$ to the right of its previously assigned position. Because of this shift no ring can cover both $P_{2(k-1)}$ and V. In order to avoid unexpected interferences between the locations where wires are attached to the u_i-component, one uses in the general case only every fourth pair (P_{2k}, P_{2k+1}) for attaching a wire to the u_i-component (by placing the first point of that wire to the left respectively to the right of this pair). Note that in order to save space we had to ignore this rule in Fig. 2.

Figure 3 shows a c_j-component in full detail. The component itself consists only of one point P_0 at a coordinate z such that $z \equiv 100w \cdot (3m+n+j) \bmod (2r+w)$. The end segments of up to three wires $\tilde{U}=(U_1, \cdots, U_e)$, $V=(V_1, \cdots, V_f)$, $W=(W_1, \cdots, W_g)$ are attached to the c_j-component. In the number M of rings that we allow for covering all points in P, no extra ring is allocated for the covering of any c_j-component. Therefore point P_0 can only be covered if the signal 1 is transmitted to the c_j-component through at least one of the wires $\tilde{U}$, V, W. More precisely the lengths e, f, g of these wires are even numbers and $e/2, f/2, g/2$ rings are allocated in M for the covering of these wires. Therefore if and only if the first point U_1 (V_1, W_1) of such a wire is covered by some other ring (necessarily a ring from the covering of the u_i-component to which this wire is attached), the last one of the rings that are allocated to this wire has to cover only its last point U_e (V_f, W_g). This last ring can cover then with its other w-interval the point P_0 of the c_j-component. In Fig. 3 we have indicated with broken lines the position of the last ring of each wire in the case where this wire transmits the signal 1 to the c_j-component. Figure 3 also shows (with solid lines) the case where the last ring that is allocated to a wire has to cover its last *two* points (this means that the corresponding wire transmits the signal 0 to the c_j-component). It is obvious from the coordinates that are given in Fig. 3 that no ring can cover points that belong to different wires.

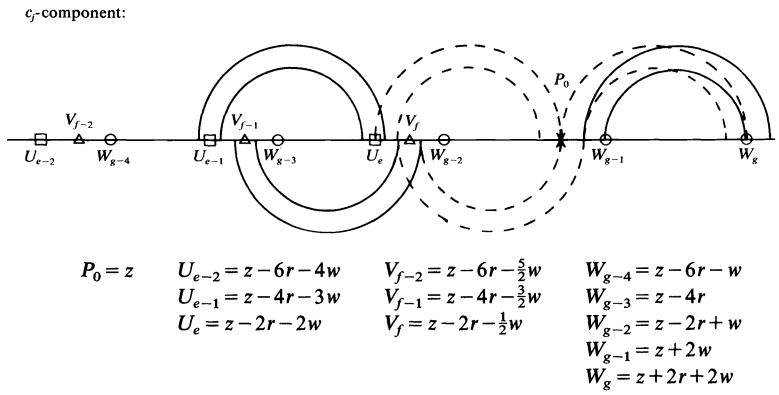
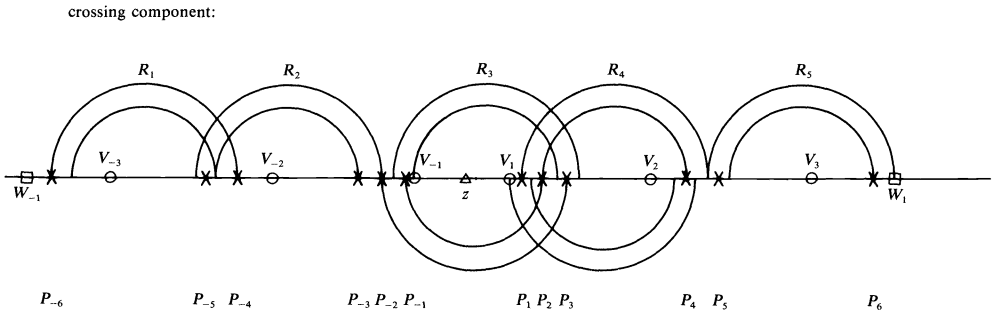


FIG. 3

Figure 4 shows the design of a "crossing component" for the crossing of two wires without interference. Such components are needed because a wire that connects a u_i-component with a c_j-component may have to cross other wires on its way to or from its regular assigned track (see Fig. 1). Figure 4 shows the component for the crossing of wires V and W (of these wires only the segments $V_{-3}, \cdots, V_3$ and W_{-1}, W_1 are indicated in the diagram). The crossing component consists of 12 points $P_{-6}, \cdots, P_6$. The left and the right half of the crossing component are drawn symmetrically (this will simplify the verification). The coordinate z of the middle of the component depends

on the track that has been assigned to the wire which is crossed by another wire with the help of the crossing component.

We first note that five rings $R_1, \cdots, R_5$ can be positioned in such a way that they cover $P_{-6}, \cdots, P_6$ and in addition a given one of the two points V_{-1}, V_1 and also a given one of the two points W_{-1}, W_1. The top half of Fig. 4 shows the positions of five rings $R_1, \cdots, R_5$ that cover $P_{-6}, \cdots, P_6$, W_1 and V_{-1}. The bottom half of Fig. 4 indicates a way of shifting R_3, R_4 so that together with the (unchanged) rings R_1, R_2, R_5 the five rings together now cover $P_{-6}, \cdots, P_6$, W_1 and V_1. In the case where W_{-1} has to be covered (instead of W_1) we use the fact that the component is symmetrical with respect to z. With the help of reflection at z we get from Fig. 4 the positions of five rings that cover $P_{-6}, \cdots, P_6$, W_{-1} and V_1 (respectively V_{-1}).



$P_1 = z + r - \frac{1}{2}w$ $\quad V_1 = z + r - w$ $\quad W_1 = z + 5r + \frac{7}{2}w$
$P_2 = z + r + \frac{1}{2}w$ $\quad V_2 = z + 3r - w$ $\quad W_2 = z + 7r + \frac{11}{2}w$
$P_3 = z + r + \frac{3}{2}w$ $\quad V_3 = z + 5r$ $\quad$ (this point is not shown)
$P_4 = z + 3r + \frac{1}{2}w$ $\quad V_4 = z + 7r$
$P_5 = z + 3r + 2w$
$P_6 = z + 5r + \frac{5}{2}w$

If $z + d$ is the coordinate of point P_i, then $z - d$ is the coordinate of point P_{-i} (analogously for the points V_{-i} and W_{-i}).

FIG. 4

One can see from the coordinates of the points in Fig. 4 that a ring that covers any of the points $P_{-6}, \cdots, P_6$ can reach no point that belongs to wire V or W except possibly some of the points V_{-1}, V_1, W_{-1}, W_1. Further in the total number M of rings that are allowed for the covering of all points in P only five rings are allocated for each crossing component. If six or more rings are used to cover $P_{-6}, \cdots, P_6$ then these rings may cover simultaneously all points in the set $\{V_{-1}, V_1, W_{-1}, W_1\}$. But by the design of the other components it is then impossible to cover with $M-6$ rings all the remaining points in P.

We now show that if any five rings $R_1, \cdots, R_5$ (numbered according to their position from left to right) cover all points $P_{-6}, \cdots, P_6$ then either W_{-1} or W_1 and either V_{-1} or V_1 are not covered by these rings. Obviously all the groups $\{P_{-5}, P_{-4}\}$, $\{P_{-3}, P_{-2}, P_{-1}\}$, $\{P_1, P_2, P_3\}$, $\{P_4, P_5\}$ contain two points whose distance d lies strictly between w and $2r$. Therefore for each of these groups at least two different rings must participate in the covering of this group. This implies that each of these groups is covered by precisely the same rings as in Fig. 4.

Assume for a contradiction that both V_{-1} and V_1 are covered by $R_1, \cdots, R_5$ (besides $P_{-6}, \cdots, P_6$). If R_2 covers V_{-1} and R_4 covers V_1 then R_3 has to cover both P_{-3} and P_3, although $|P_{-3} - P_3| > 2r + 2w$. Thus we may assume that R_3 covers V_{-1} (the

case where R_3 covers V_1 is symmetrical). Then the location of the left end of R_3 is at some coordinate $\geqq z-r$. Therefore the other interval of R_3 does not cover P_2. Further V_1 can only be covered by R_4. Therefore the left end of R_4 is located at some coordinate $\leqq r-w$. This implies that R_4 does not cover P_2. Thus P_2 remains uncovered, a contradiction.

Finally assume for a contradiction that both W_{-1} and W_1 are covered by $R_1, \cdots, R_5$ (besides $P_{-6}, \cdots, P_6$). Our preceding consideration implies that only R_1 can cover W_{-1} and that only R_5 can cover W_1. Therefore R_1 does not cover P_{-4} and R_5 does not cover P_4. Thus R_2 covers P_{-4} and R_4 covers P_4. This implies that R_2 does not cover P_{-3} and R_4 does not cover P_3. But R_3 cannot cover P_{-3} and P_3 since $|P_{-3}-P_3|>2r+2w$. Thus either P_{-3} or P_3 remains uncovered, a contradiction.

We have specified for each occurring component c the number M_c of rings that are allocated for this component among the M rings. M is then defined as the sum of these M_c (over all components c). P is defined as the union of all components (note that the number of components is polynomial in m and n). The preceding arguments imply that the given formula F is satisfiable if and only if all points in P can be covered by M rings of size $\langle r, w\rangle$ (for r and w as defined before).

Remark 2.2. In some sense it is easier to reduce PLANAR 3SAT instead of 3SAT to the considered problem (see Lichtenstein [7]): no crossing components are needed in this case. On the other hand one then has less control over the structure of the (planar) graph that has to be represented. This fact makes an explicit description of this variation of the proof very difficult.

3. Covering with rings of bounded degree of nonconvexity is in P. In order to demonstrate why covering with nonconvex objects is more difficult than covering with convex objects, we first give a simple algorithm for covering with convex objects in one dimension (we cover with one-dimensional rings of size $\langle r, w\rangle$ where $r/w=0$). In this algorithm one places intervals of length $2w$ successively so that their left endpoint coincides with the leftmost one of the given points that is not yet covered.

If one covers with nonconvex rings, one has several choices among positions of rings that cover the leftmost point that is not yet covered. One can either place this ring far to the right (so that its right end reaches as far as possible) or one can place it more to the left (so that the left end of the right w-interval covers additional points). In this way the number of reasonable choices for placing the first m rings grows exponentially in m. Therefore we use a different approach in the following polynomial time algorithm.

THEOREM 3.1. *There is an algorithm that computes for n given points on the line and a given ring size $\langle r, w\rangle$ a covering of the given points by a minimum number of rings of size $\langle r, w\rangle$ in $O(n^{O(\lceil r/w\rceil)})$ steps (respectively in $O(n)$ steps if $r/w=0$).*

Proof. Assume n points on the line and a ring size $\langle r, w\rangle$ with $r>0$ are given. The algorithm relies on the following definition.

DEFINITION 3.2. Consider an arrangement B of rings of size $\langle r, w\rangle$ on the line, where the leftmost ring has its center at C_L and the rightmost ring has its center at C_R. We call B a *block* if every given point in the interval (C_L+r+w, C_R-r-w) is covered by some ring in B.

LEMMA 3.3. *Consider any covering C of all given points by rings of size $\langle r, w\rangle$. Let B be a subset of rings from C. Let C_L be the center of the leftmost ring in B and let C_R be the center of the rightmost ring in B. Assume that every ring in C whose center is located in the interval (C_L, C_R) belongs to B. Then B is a block.*

The proof of Lemma 3.3 follows immediately from the definition of a block.

LEMMA 3.4. *For any block B the subset of the n given points that are covered by B can be characterized with the help of at most $8\lceil r/w\rceil+6$ of the given points (independent of the size of B).*

Proof of Lemma 3.4. Let C_L be the leftmost and C_R be the rightmost center of rings in B. Then all of the given points in (C_L+r+w, C_R-r-w) and none of the given points in $(-\infty, C_L-r-w)$ or $(C_R+r+w, +\infty)$ are covered by B. Within the interval $[C_L-r-w, C_L+r+w]$ the block B defines a set of at most $2\lceil r/w\rceil+1$ disjoint intervals of length $\geqq w$ so that in each such interval all given points are covered by B and any two such intervals are separated by intervals of uncovered points. Each of these up to $2\lceil r/w\rceil+1$ intervals can be characterized by the leftmost one and the rightmost one of the given points that it covers. Together this requires up to $2\cdot(2\lceil r/w\rceil+1)$ points. We need the same amount of information to characterize the right end of B. Finally we need two more points to describe the endpoints of the largest interval in the interior of B where all given points are covered.

To motivate our algorithm we consider any minimum covering C of all given points. We partition C into two blocks B_1 and B_2 where B_1 consists of the 2^m leftmost rings in C and m is maximal such that $2^m<|C|$. In the same way we partition each of the blocks B_1, B_2 into two blocks of about half its size. After at most $\lceil\log_2|C|\rceil$ iterations of this step we have broken down C into its "atoms": single rings. The following dynamic programming algorithm reverses the described process: we look at all possible ways of concatenating two smaller blocks so that they yield one larger block. We can do this in polynomial time because by Lemma 3.4 there exist at most $n^{8\lceil r/w\rceil+6}$ different subsets S of the set of all n given points so that some block of rings covers precisely the points in S.

ALGORITHM. We develop a table where we record for blocks of increasing lengths the subset of the n given points which is covered by each block. In addition we record for each block the number of rings that it uses and the locations of the centers of its leftmost and its rightmost ring. Thus each entry in the table requires at most $O((8\lceil r/w\rceil+9)\cdot\log n)$ bits. We also set up a list that allows us to check in $O((8\lceil r/w\rceil+9)\cdot\log n)$ steps whether a candidate entry for the table already appears in the table.

In the first row of the table we record for all blocks of length 1 the described data. It is sufficient to consider here only rings that are positioned in such a way that one of their four endpoints coincides with one of the given points.

In each subsequent row we record the described data for each block that arises as the union of two blocks from previous rows (unless we get an entry that appears already in the table).

After we have written $\lceil\log_2 n\rceil$ rows we give as output the first entry in the table where all n given points are covered such that no covering of all points with fewer rings has been recorded in the table.

To justify the algorithm we note that one can always shift a ring—without changing the set of given points that it covers—until one of its endpoints coincides with one of the given points. The correctness of the algorithm follows then from our preceding observations.

There are at most $O(n^{8\lceil r/w\rceil+9})$ entries in the table. Thus one has to check for at most $O(n^{16\lceil r/w\rceil+18})$ pairs of previously recorded entries whether they yield a new entry in the table. For each of these pairs one needs at most $O(\lceil r/w\rceil\cdot\log n)$ steps to check whether the union of the corresponding blocks yields a new block whose characteristic data do not yet appear in the table (and to compute the characteristic data of the new block). In this way we arrive at an upper bound of $O(n^{16\lceil r/w\rceil+18}\cdot\lceil r/w\rceil\cdot\log n)$ steps for the algorithm.

Remark 3.5. Theorem 3.1 shows that for any fixed bound on the degree of nonconvexity r/w of the covering rings the one-dimensional ring cover problem is in P. Easy variations of the proof show that this remains true if in addition certain parts of the line are "forbidden" as centers of rings. Further one can associate different costs with having centers of rings at different locations and then compute in polynomial time a covering of minimum cost. One can also extend these algorithms to the case where besides points on a line also certain whole intervals have to be covered.

In another possible extension one might assume that k different ring sizes $\langle r_1, w_1\rangle, \cdots, \langle r_k, w_k\rangle$ are given, where rings from any of these sizes may be used for a minimum covering. Also one can associate different costs with different ring sizes and compute a covering of minimum cost in polynomial time. Notice that this variation includes the case where rings of a certain size may be placed not only with their centers *on* the line but also at a number of different distances from the line (in our terminology each distance gives rise to a different ring size when we consider the intersection of such a two-dimensional ring with the considered line). In this extension the ratio $\max\{r_i | i \leqq k\}/\min\{w_i | i \leqq k\}$ appears in the degree of the polynomial time bound in place of r/w.

4. A property of minimum covers by rings of low nonconvexity. If one covers given points on the line by a minimum number of intervals (i.e. rings with $r/w = 0$), one can assume without loss of generality that the leftmost interval of the covering is positioned with its left end at the leftmost given point. Because of this property one can compute in one dimension minimum covers by convex objects in linear time (see the beginning of § 3). Unfortunately this property does not hold for minimum covers by rings of size $\langle r, w\rangle$ for any $r/w > 0$. We show in this section that nevertheless a more general property holds for rings with ratio $r/w \leqq \frac{1}{2}$ (and not for rings with any bigger ratio). The property says that for rings with ratio $r/w \leqq \frac{1}{2}$ one can assume without loss of generality that the leftmost ring of a minimum cover is positioned at one of *two* canonical positions, both of which are easy to compute. Thus one can answer certain questions about the position of the first ring of a minimum cover without computing a minimum cover. One can further use this structural property of minimum covers to design a fast approximation algorithm for covering with rings of ratio $r/w \leqq \frac{1}{2}$ (see [3] and [4]).

LEMMA 4.1. *The following implication holds if and only if $r/w \leqq \frac{1}{2}$: If there exists a minimum cover of given points by rings of size $\langle r, w\rangle$ where the leftmost ring has one of the given points in the gap between its two w-intervals, then there also exists a minimum cover by rings of size $\langle r, w\rangle$ where the leftmost ring is positioned with its left end at the leftmost given point.*

Proof. We first give a counterexample for the case $r/w > \frac{1}{2}$. We assume that three points a, b, c are given. We choose the distance d between b and c such that $w < d < 2r$ (this is possible if and only if $r/w > \frac{1}{2}$). The locations of the points a, b, c are indicated in Fig. 5. The two rings in the upper half of Fig. 5 form a minimum cover. The leftmost ring has point b in its gap. On the other hand if we position the leftmost ring of a

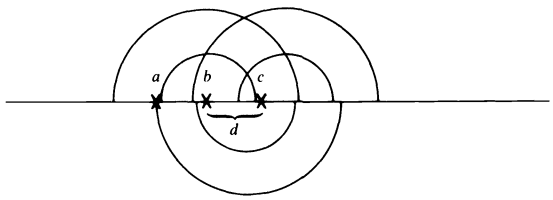


FIG. 5

covering with its left end at the leftmost given point a (as in the bottom half of Fig. 5) we need two more rings to cover b and c.

The positive result for $r/w \leqq 1/2$ follows from the following observation. Let $R_1, \cdots, R_k$ be any minimum cover by rings of size $\langle r, w \rangle$ with $r/w \leqq \frac{1}{2}$ (we assume that the rings are numbered from left to right). Assume that one of the given points lies in the inner disc I_1 (of radius r) of the first ring R_1. Let $\tilde{R}_1$ be a ring that is positioned with its left end at the leftmost given point and let $\tilde{I}_1$ be its inner disc (of radius r). We define point L as the left end of $\tilde{I}_1$. We consider two cases.

Case 1. The left end of ring R_2 is at point L or to the left of L. In this case we continue the cover that was started by $\tilde{R}_1$ with a ring $\tilde{R}_2$ that is positioned with its left end at L. Since $2r \leqq w$ the rings $\tilde{R}_1$ and $\tilde{R}_2$ together cover all points from the leftmost given point until as far as $2r+2w$ to the right of point L. It is obvious that all points that are covered by R_1 or R_2 fall into this interval.

Case 2. Otherwise. By assumption one of the given points lies in I_1 and without loss of generality this point is covered by R_2. Since this point is left of the right end of $\tilde{I}_1$, the left end of R_2 lies inside of $\tilde{I}_1$. Therefore all given points in $\tilde{I}_1$ are covered by R_2 because $2r \leqq w$. This implies that all given points that are covered by R_1 or R_2 are also covered by $\tilde{R}_1$ or R_2.

THEOREM 4.2. *Assume that points on a line and a ring size $\langle r, w \rangle$ with $r/w \leqq \frac{1}{2}$ are given. Then there is a minimum cover of these points by rings of size $\langle r, w \rangle$ where the leftmost ring R of this cover has at least one of the following two properties*:

(1) *The left end of R coincides with the leftmost given point.*

(2) *R is in the rightmost possible position where it covers the leftmost given point and has none of the given points in its gap.*

Proof. Let S be the leftmost ring of a minimum cover. If one of the given points falls into the gap of S the claim follows from Lemma 4.1. Otherwise the rightmost possible ring R that covers the leftmost given point and has none of the given points in its gap covers all points that are covered by S.

5. The capacitated ring cover problem. We now consider a capacitated version of the ring cover problem. We assume that in addition to the previously considered input data a natural number b is given, which we interpret as the "capacity" of a ring. In addition to a covering we now also have to assign to each of the given points one of the rings that cover this point (one says that the assigned ring "serves" this point). This assignment has to be arranged in such a way that no ring has to serve more than b points. The goal is again to minimize the number of rings that are used.

The capacitated version appears to be of interest for both of the possible applications that were described in the introduction. It also appears to be of some mathematical interest because the algorithm from the previous section does not readily extend to the capacitated problem. The reason for this difficulty is the fact that the degree of the polynomial time bound of the algorithm from Theorem 3.1 is proportional to the number of points that are needed to characterize which of the given points are covered by a block. In an extension of this algorithm to the capacitated case one also has to record for every block which of the points that it covers are served by rings in this block. If for example $n/10$ points lie at the fringe of a block, this may require up to $n/10$ data. Therefore even for a fixed bound on b and on r/w the resulting algorithm is no longer polynomial in the number n of given points.

We show below that there exist among all minimum solutions of the capacitated ring cover problem certain "normal" solutions. Normal solutions are characterized by the fact that they can be decomposed into blocks of a particular simple structure which

we call b-blocks (see Definition 5.4). The number of data that are needed to characterize the set of points that are served by the rings of a b-block is independent of n. Since there exist minimum solutions that are in addition normal, it is sufficient to record in the table of a dynamic programming algorithm only those sets of given points that are served by a b-block. In this way we arrive again at a polynomial time algorithm.

We now show that one can "normalize" any given solution to the capacitated ring cover problem without increasing the number of rings that are used. This normalization process consists of two steps. First we minimize the number of rings that serve points in both of their w-intervals. Then we change positions of rings and the assignment of points for those rings that serve now only points in one of their w-intervals in order to minimize the overlap of their "service areas". This second step of the normalization process uses the same method as the proof of the following result for the convex case.

THEOREM 5.1. *There is an algorithm that computes for n given points on a line, a given interval length d and a given capacity b in $O(n)$ steps the positions of a minimum number of intervals of length d together with an assignment of each given point to some interval that covers this point such that no interval serves more than b points.*

Proof. We extend the simple algorithm from the beginning of § 3. We place successively the next interval so that its left end coincides with the leftmost point that is not yet served. We assign to this interval the b leftmost points that it covers (there may be less than b points).

One shows by induction on n that this algorithm uses the minimum number of intervals. For the induction step consider any minimum solution C. It is possible to change the position of the leftmost interval in C and the assignment of points to this interval so that this interval serves the same points as the first interval that is placed by the algorithm. We can then apply the induction hypothesis to the remaining points, where we use the (previously slightly altered) rest of C for comparison.

The following is the desired result for the nonconvex case.

THEOREM 5.2. *There is an algorithm that computes for n given points on the line, a given ring size $\langle r, w\rangle$ and a given capacity b in $O(n^{O(\lceil r/w\rceil b^5)})$ steps a minimum solution to the capacitated ring cover problem.*

Proof. According to the outline at the beginning of this section we first show that among the minimum solutions of the considered problem there exist certain "normal" ones. Let C be a solution of the considered problem. In the first step of the normalization process we minimize the number of rings that serve points in both of their intervals. Thus let $\tilde{C}$ be the result of replacing—without increasing the number of rings that are used—the maximum possible number of rings in C that serve points in both of their intervals by rings that serve points in only one of their intervals (we change the assignment of points accordingly). Of course this has to be done in such a way that $\tilde{C}$ is also a solution of the considered problem.

LEMMA 5.3. *For every real number c there are in $\tilde{C}$ less than b^4+3b rings with center in $[c, c+w]$ that serve points in both of their intervals.*

Proof of Lemma 5.3. Assume for a contradiction that there are in $\tilde{C}$ at least b^4+3b rings with center in $[c, c+w]$ that serve points in both of their intervals. Each such ring R defines a triple of numbers $\langle x_R, y_R, z_R\rangle$ which are the numbers of points that ring R serves in each of the three intervals $[c-r-w, c-r]$, $(c-r, c-r+w]\cap(c-r, c+r)$, $[c+r, c+r+w]$. Since the numbers x_R, y_R, z_R range from 0 to b there are less than b^3+3 different triples of numbers that occur. Thus at least b of these rings have the same triple $\langle x, y, z\rangle$. We show that these b rings can be replaced by b rings that serve points in only one of their intervals. We position x rings with the left end at $c-r-w$ and assign to them those bx points that lie in $[c-r-w, c-r]$ and which

were served before by the b replaced rings. Analogously we position y, z, $b-x-y-z$ rings with the left end at $c-r$, $c+r$, $c+r+w$ respectively and we assign to them those points in the corresponding intervals $(c-r, c-r+w] \cap (c-r, c+r)$, $[c+r, c+r+w]$, $(c+r+w, c+r+2w]$ that were served before by the b replaced rings.

The possibility of this substitution contradicts the definition of $\tilde{C}$.

It may still occur that for example two rings R_1 and R_2 in $\tilde{C}$ serve only points in their left interval and R_1 is left of R_2, but R_1 serves some points that lie to the right of points that are served by R_2. Such overlap can make the description of the set of points that are served by a block arbitrarily long. Therefore we consider now the subset S of the n given points that are served in $\tilde{C}$ by rings that serve points in only one of their intervals. Say there are k such rings in $\tilde{C}$. We apply to the points in S the algorithm of Theorem 5.1, where we use intervals of length w. By Theorem 5.1 the algorithm uses exactly k such intervals. We now interpret each such interval as the left interval of a ring of size $\langle r, w\rangle$. After we have changed in this way those rings in $\tilde{C}$ that serve points in only one of their intervals, we call the resulting new covering of all n points C'. By construction C' uses no more rings than C. Further the properties of the algorithm from Theorem 5.1 guarantee that:

(I) If R_1 and R_2 are two rings in C' that serve points in only one of their intervals, then both rings are positioned with the left end at the leftmost point which they serve and if R_1 is positioned left of R_2, then all points that are served by R_1 lie to the left of every point that is served by R_2.

In addition C' has the same rings as $\tilde{C}$ that serve points in both of their intervals. Thus C' retains the property that was proved in Lemma 5.3 for $\tilde{C}$. Thus we have:

(II) For every real number c there are in C' less than b^4+3b rings with center in $[c, c+w]$ that serve points in both of their intervals.

We call a solution C' with properties (I) and (II) a *normal* solution. The preceding construction shows that there always exists a minimum solution that is in addition normal.

As in Theorem 3.1, the key for the dynamic programming algorithm is the definition of a relatively small class of "building blocks" from which one can build via concatenation an optimal solution.

DEFINITION 5.4. Consider an arrangement B of rings of size $\langle r, w\rangle$ such that no ring in B serves more than b points. Let $C_L(C_R)$ be the center of the leftmost (rightmost) ring in B. We call B a *b-block* if we have for $f(b, r/w) = (2^{\lceil r/w \rceil}+2) \cdot (b^5+3b^2)+b$

i) every point in (C_L+r+w, C_R-r-w) is served by a ring from B,

ii) at most $f(b, r/w)$ points in $[C_L-r-w, C_L+r+w] \cap [C_L-r-w, C_R-r-w)$ are not served by a ring from B,

iii) at most $f(b, r/w)$ points in $[C_R-r-w, C_R+r+w]$ are served by a ring from B.

It is obvious that the relevant properties of such b-block can be described with at most $2 \cdot f(b, r/w)+3$ data (each of which is essentially a number between 1 and n). Besides C_L, C_R and the number of rings that are used in the b-block this description includes those $2 \cdot f(b, r/w)$ points that are mentioned in part ii) respectively iii) of the definition.

LEMMA 5.5. *Let C be a normal solution of the considered problem. Consider a collection B of rings from C where the leftmost ring in B has its center at C_L, the rightmost center in B has its center at C_R and every ring in C that has its center in (C_L, C_R) belongs to B. Then B is a b-block.*

Proof of Lemma 5.5. Property i) is obvious. For property ii) we first consider those points in $I = [C_L-r-w, C_L+r+w] \cap [C_L-r-w, C_R-r-w)$ which are served by rings

in C that serve points in only one of their intervals. Except for possibly the first one, these rings have their centers in (C_L, C_R) and thus they belong to B (we use part (I) of the normality definition). This gives rise to at most b points in I that are not served by rings from B. Next we consider those points in I that are served by rings of C that serve points in both of their intervals. Unless their centers are in $[C_L-2r-2w, C_L]$, these rings necessarily belong to B. By part (II) of the normality definition there are at most $((2r+2w)/w)\cdot(b^4+3b)$ rings in C that have their centers in $[C_L-2r-2w, C_L]$ and serve points in both of their intervals. These rings serve at most $(2\lceil r/w\rceil+2)\cdot(b^5+3b^2)$ points. Property iii) is verified analogously.

Similarly as before it is sufficient to consider in the following algorithm only positions of rings where one of the given points coincides with one of the four endpoints of the ring.

ALGORITHM. In the first row of the table we write down the covering properties of all b-blocks that consists of one ring. In each subsequent row we list the characteristic data (consisting of $O(f(b, r/w)\cdot\log n)$ bits) for each new b-block which we get by taking the union of two b-blocks from previous rows.

After we have written down $\lceil\log_2 n\rceil$ rows we output the first b-block in the table that serves all n given points and such that no other b-block in the table serves all n points with fewer rings.

The correctness of the algorithm follows from the previous observations. In particular some minimum solution that is in addition normal will appear in the table. This ensures that the output is a minumum solution.

For the time analysis we note that there are at most $O(n^{2\cdot f(b,r/w)+3})$ entries in the table. Thus we consider at most $O(n^{4\cdot f(b,r/w)+6})$ pairs of b-blocks during the algorithm. For each pair we need at most $O(\log n\cdot f(b, r/w))$ steps to check whether its union forms a b-block whose characteristic data do not yet appear in the table. This leads to an upper bound of $O(n^{4\cdot f(b,r/w)+7}\cdot f(b, r/w))$ steps for the algorithm.

Remark 5.6. The proof of Theorem 2.1 implies that already for a fixed capacity $b\geqq 3$ the one-dimensional capacitated ring cover problem is NP-complete.

REFERENCES

[1] J. J. BARTHOLDI III, *A guaranteed-accuracy round-off algorithm for cyclic scheduling and set covering*, Oper. Res., 29 (1981), pp. 501-510.

[2] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, Freeman, San Francisco, 1979.

[3] D. HOCHBAUM AND W. MAASS, *Approximation schemes for covering and packing problems in robotics and* VLSI, Proc. STACS 84, Lecture Notes in Computer Science, 166, Springer, Berlin, 1984.

[4] ———, *Fast approximation algorithms for a nonconvex covering problem*, submitted.

[5] ———, *Approximation algorithms for covering and packing problems in image processing and* VLSI, J. Assoc. Comput. Mach. (1985).

[6] D. S. JOHNSON, *The* NP-*completeness column: an ongoing guide*, J. Algorithms, 3 (1982), pp. 182-195.

[7] D. LICHTENSTEIN, *Planar formulae and their uses*, this Journal, 11 (1982), pp. 329-343.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
010

LOG-LOGARITHMIC SELECTION RESOLUTION PROTOCOLS IN A MULTIPLE ACCESS CHANNEL*

DAN E. WILLARD†

Abstract. We propose two selection protols that run on multiple access channels in log-logarithmic expected time, and establish a complementary lower bound showing that the first protocols falls within an additive constant of optimality and that the second differs from optimality by less than any multiplicative factor infinitesimally greater than 1 as the size of the problem approaches infinity. It is difficult to second-guess the fast-changing electronics industry, but our mathematical analysis could be relevant outside the traditional interests of communications protocols to semaphore-like problems.

Key words. ethernet, multiple access channel, randomized time, selection protocol, semaphore

1. Introduction. Consider a multiple access channel that resembles an ethernet mathematically although its domain of application may be radically different. That is, assume N nodes are connected to a medium that generates the signals of 0, 1 and e in the respective cases where zero, one, and more than one nodes are broadcasting on it simultaneously. If Q of these N nodes wish to broadcast on this medium, then the purpose of a selection protocol is to issue the signal "1" in the least expected time. It is well known [MB76] that this problem is solvable in expected time $O(1)$ when Q's value is known in advance to all the participating nodes. We consider the case where Q's value is initially unknown and propose one selection protocol which runs in time $\log\log N + O(1)$, and a second protocol which runs in time $\log\log Q + o(\log\log Q)$ for the slightly different problem where N's value is unknown. (Throughout this paper $\log(\ )$ designates $\log_2(\ )$, and $\ln(\ )$ is natural logarithm.) This paper also establishes a lower bound $\log\log N$-$O(1)$ which shows that the first protocol is optimal to within an additive constant and that the second protocol differs from optimality by only the additive amount $o(\log\log Q)$.

The problem considered by us should not be confused with articles from the previous literature such as [Ca79], [GL82], [GW84], [Ma80], [Me73], [MB76], [MT80], [Pi79], [SH80]. One distinction is that some of the prior articles discuss protocols where Q's value is known in advance, and a more important distinction is that our protocol seeks to minimize the time for the first broadcast of "1" on the multipie access channel, whereas most of the previous literature studies how to enable all Q seeking nodes to serially access the communication channel in minimal time. The term *conflict resolution* refers to the traditional serial access problem, and our research problem is called *selection resolution.* [GW84] develops a lower bound $Q\log N/\log Q$ for worst-case conflict resolution. Taking $Q=2$, this theorem applied somewhat out of context implies $\Omega(\log N)$ lower bounds the worst-case selection time. Our upper bound $\log\log N + O(1)$ on expected selection time shows that selection is another example of a communications problems whose randomized time is asymptotically better than its worst-case performance. (A classic contrast of this type is the distinction between the randomized upper bound in [Ra80] and the lower bound for worst-case performance in [BFJLP78].)

Our technique could conceivably have practical implications under the new emerging electronic technologies. This is because the new emerging electronic technologies

* Received by the editors September 20, 1983, and in final revised form April 10, 1985.

† Department of Computer Science, State University of New York at Albany, Albany, New York 12222, and consultant for Bell Communications Research.

may soon change the mathematical implications of the communications literature by making cost-effective the use of multiple access ethernet-like devices as semaphores. For instance, suppose several processors wish to access a memory chip and these processors also are connected to a communication medium resembling our $(0, 1, e)$ device; these processors could communicate their needs for the critical resource via the device by a protocol that assigns the resource to the first processor responsible for the transmission of the signal "1" on the multiple access channel. This type of semaphore would be much faster than the methods discussed in [MFLJP78], [Di65], [Kn66], [Ra80], but it assumes a different type of hardware whose cost-effectiveness will ultimately depend on the directions of the new emerging electronic technologies.

These techniques could also be useful in other technologies, as we indicate in § 4. The main discussion in this paper is divided into two parts, where § 2 proves our log-logarithmic upper bounds and § 3 establishes the complementary lower bound.

2. The upper bounds. This section will display our two protocols which run in log-logarithmic time. In our discussion, t will denote one of the three signals of 0, 1 or e communicated on the multiple access channel, and TEST (q) an operation where each of the Q seeking nodes attempt a broadcast with a probability of $1/q$. $P(q, Q, t)$ will denote the probability that the resulting signal equals t; equations (2.1) through (2.3) indicate the values of $P(q, Q, t)$ following from Bernoulli's theorem [Fe68]:

$$P(q, Q, 0) = (1-1/q)^Q, \tag{2.1}$$

$$P(q, Q, 1) = (Q/q)(1-1/q)^{Q-1}, \tag{2.2}$$

$$P(q, Q, e) = 1 - P(q, Q, 0) - P(q, Q, 1). \tag{2.3}$$

The protocols considered in this paper will perform a sequence of trials TEST (q_1), TEST (q_2), TEST (q_3), $\cdots$. These trials will generate a signal sequence $t_1 t_2 t_3 \cdots$ which terminates when some $t_i = 1$, indicating an unique node has been assigned the critical resource. A theme in the communications literature is that TEST (q) has a high probability of producing the desired signal "1" when q and Q agree to within a factor of 2, and this probability is low otherwise (these facts follow from a straightforward analysis of equations (2.1) thru (2.3), and we will explain them more fully when we use them later). The main challenge for a selection protocol is to try to make the ith guess q_i come as close as possible to Q by using the test data $t_1 t_2 \cdots t_{i-1}$ to infer Q's likely value.

An algorithm that runs in expected time $\log Q$ is binary exponential backoff [MB76]. There are several minor variants of this protocol. One variant initially sets $q = 1$, increases q's by a factor of 2 whenever a trial produces TEST $(q) = e$, and finally terminates when some test produces the desired signal "1". Its expected time is $\cong \log Q$ essentially because q and Q's value agree within a factor of 2 after this many trials, and thereafter the probability is high that TEST $(q) = 1$.

Most of this section will focus on a protocol, called super exponential binary search (SEBS), which runs in time $\log\log N + O(1)$. A modified version of this protocol, called QSEB, running in time $\log\log Q + o(\log\log Q)$, will be outlined at the end of this section. It is uncertain which protocol has the more practical coefficient.

The protocol SEBS assumes that the only information initially known about Q is that $2 \leqq Q \leqq N$. It consists of the following two-phase search.

Phase 1. Attempt to guess which power of 2 between 2 and $2^{\lceil \log N \rceil}$ is the best guess of Q's actual value by using the procedure shown in Fig. 1 to use the data from the trials TEST (q_1), TEST (q_2), $\cdots$ to binary search the $\lceil \log N \rceil$ different eligible

A binary search to guess Q's value to the nearest factor of 2 in time $\lfloor \log \log N \rfloor + 1$:

(1) Initially set $L = 0$ and $U = \lceil \log N \rceil + 1$
(2) WHILE $U \neq L+1$ DO:
  (a) SET $i = \lceil (U+L)/2 \rceil$ and $q = 2^i$.
  (b) SET $t = \text{TEST}(q)$.
  (c) IF $t = 1$ THEN protocol terminates with the node issuing this signal assigned the critical resource. ELSE SET $U = i$ if $t = 0$, and SET $L = i$ if $t = e$.

END OF PROGRAM.

FIG. 1

powers of 2 in time $\leqq \lfloor \log \log N \rfloor + 1$. If any trial during this search produced the signal "1", then the protocol terminates immediately with the critical resource assigned to that node generating the signal "1". Otherwise, the protocol proceeds to the next step:

Phase 2. Let q_f denote the final value of q tested in Phase 1. Define q^* to:
(a) equal q_f iff $\text{TEST}(q_f)$ in Phase 1 equaled zero,
(b) and to equal $2 \cdot q_f$ in the alternate case where $\text{TEST}(q_f)$ equaled "e". Repeatedly perform the trial $\text{TEST}(q^*)$ until the multiple access channel emits the signal "1", indicating an unique node has been assigned to the critical resource.

The protocol above is of course quite simple but the proof of SEBS' expected time $\log \log N + O(1)$ is not. This is because the value of q^* in Phase 2 is determined only by random trials in Phase 1. The proof must show that bad luck in Phase 1 is sufficiently uncommon to guarantee Phase 2 runs in expected time $O(1)$, a result which will eventually enable us to prove SEBS runs in time $\log \log N + O(1)$ and is optimal up to an additive constant.

In our discussion, λ will denote the ratio Q/q. Since the Poisson distribution is the asymptotic limit of Bernoulli trials [Fe68], equations (2.1) through (2.3) certainly imply

$$\lim_{q \to \infty} P(q, \lambda q, 0) = e^{-\lambda}, \tag{2.4}$$

$$\lim_{q \to \infty} P(q, \lambda q, 1) = \lambda\, e^{-\lambda}, \tag{2.5}$$

$$\lim_{q \to \infty} P(q, \lambda q, e) = 1 - e^{-\lambda} - \lambda\, e^{-\lambda}. \tag{2.6}$$

Unfortunately, the fairly standard equations above are not the inequalities we need. We need inequalities which approximate the value of $P(q, Q, t)$ when q is a small number. These inequalities appear in the next several lemmas.

LEMMA 2.1. *Assume* $q \geqq 2$. *Then* $P(q, Q, 1) \geqq \lambda 4^{-\lambda}$.

Proof. Consider the function $f(x) = [\ln(1-x)]/x$. This function obtains its minimum value on the half-open interval $(0, 1/2]$ at the point $x = 1/2$. Thus all $q \geqq 2$ certainly satisfy

$$\ln(1 - 1/q) \geqq -(2 \ln 2)/q. \tag{2.7}$$

(The inequality in (2.7) is tight when $q = 2$.) Equations (2.2) and (2.7) imply

$$P(q, Q, 1) \geqq (Q/q)\, e^{-2(Q-1)(\ln 2)/q}. \tag{2.8}$$

Since $\lambda = Q/q$, the lemma follows from the fact that the right side of (2.8) exceeds $\lambda\, e^{-(2 \ln 2)\lambda}$. Q.E.D.

LEMMA 2.2. *Again assume* $q \geqq 2$. *Then* $P(q, Q, 0) \geqq 4^{-\lambda}$.

Proof. Follows by the same reasoning as Lemma 2.1. Q.E.D.

LEMMA 2.3. $P(q, Q, e) < \lambda^2/2$.

Proof. Let X and Y denote two particular nodes that wish to broadcast on the multiple access channel. The probability that both of them will attempt to signal "1" during the trial TEST (q) is clearly $1/q^2$. As there are $Q(Q-1)/2$ different pairs of nodes which could try to simultaneously broadcast, the probability of the signal "e" is $\leqq Q(Q-1)/(2q^2) < \lambda^2/2$. Q.E.D.

The next lemma will explain how the quantities $P(q, Q, t)$ are related to the performance of Phase 2 of SEBS.

LEMMA 2.4. *Assume* $Q \geqq 2$ *and* q *is a power of* 2 *satisfying* $q \leqq 2^{\lceil \log N \rceil + 1}$. *Then the probability that* q^* *in Phase* 2 *of* SEBS *equals* q *will be* $\leqq \text{MIN}\,[P(q, Q, 0); P(q/2, Q, e)]$.

Proof. Our verification is divided into the three cases where $4 \leqq q \leqq 2^{\lceil \log N \rceil}$, $q = 2$. and $q = 2^{\lceil \log N \rceil + 1}$. The second and third cases are boundary cases, and the first is therefore the most important of the three cases below:

Verification for the main case where $4 \leqq q \leqq 2^{\lceil \log N \rceil}$. The algorithmic definition of SEBS implies that Phase 2 will set q^* to equal q only when the trials in Phase 1 had TEST $(q) = 0$ and TEST $(q/2) = e$. Since these events occur within the respective probabilities $P(q, Q, 0)$ and $P(q/2, Q, e)$, the minimum of these numbers certainly bounds the probability that $q^* = q$.

Verification for the boundary case where $q = 2$. The algorithmic definition of SEBS implies Phase 2 sets $q^* = 2$ only when TEST $(2) = 0$ in Phase 1. The probability of this event is clearly $\leqq P(2, Q, 0)$. Furthermore since $P(1, Q, e) = 1$, this probability is also bounded by the minimum of these two numbers.

Verification for the boundary case where $q = 2^{\lceil \log N \rceil + 1}$. The algorithmic definition of SEBS implies Phase 2 sets $q^* = 2^{\lceil \log N \rceil + 1}$ only when TEST $(2^{\lceil \log N \rceil}) = e$ in Phase 1. The probability of this event is therefore bounded by $P(2^{\lceil \log N \rceil}, Q, e)$. Lemmas 2.2 and 2.3 further imply $P(2^{\lceil \log N \rceil + 1}, Q, 0) \geqq P(2^{\lceil \log N \rceil}, Q, e)$, implying this probability is also less than MIN $[P(2^{\lceil \log N \rceil + 1}, Q, 0); P(2^{\lceil \log N \rceil}, Q, e)]$. Q.E.D.

In the next several lemmas, $P^*(q, Q, 0)$ and $P^*(q, Q, e)$ will denote the ratios $P(q, Q, 0)/P(q, Q, 1)$ and $P(q/2, Q, e)/P(q, Q, 1)$, respectively. The next several lemmas will show how these quantities govern the expected time of Phase 2 of SEBS:

LEMMA 2.5. *Let* $Z(N)$ *denote all the powers of* 2 *that are* $\leqq 2^{\lceil \log N \rceil + 1}$. *Then if* Q *nodes wish to broadcast on the multiple access channel, the expected time of Phase* 2 *of* SEBS is $\leqq \sum_{q \in Z(N)} \text{MIN}\,[P^*(q, Q, 0); P^*(q, Q, e)]$.

Proof. The expected time of a particular invocation of Phase 2 depends on the value of q^* in this phase. If $q^* = q$, then each iteration of TEST (q^*) will generate the signal "1" with a probability $P(q, Q, 1)$. Standard probability theory [Fe68] implies the expected number of trials will then be $1/P(q, Q, 1)$.

Since Lemma 2.6 indicates MIN $[P(q, Q, 0); P(q/2, Q, e)]$ bounds the probability that $q^* = q$, the total contribution of all powers of 2 to the expected time is clearly $\leqq$

$$\sum_{q \in Z(N)} \text{MIN}\,[P(q, Q, 0); P(q/2, Q, e)]/P(q, Q, 1). \qquad \text{Q.E.D.}$$

Now we present three lemmas that will enable us to bound the summand from Lemma 2.5.

LEMMA 2.6. *Assume* $q \geqq 2$. *Then* $P^*(q, Q, 0) \leqq 1/\lambda$.

Proof. Follows by dividing equation (2.1) by (2.2) and using the fact λ is defined as Q/q. Q.E.D.

LEMMA 2.7. *Assume* $q \geqq 2$. *Then* $P^*(q, Q, e) \leqq 2\lambda \cdot 4^{\lambda}$.

Proof. Lemma 2.3 implies

$$P(q/2, Q, e) < 2\lambda^2. \tag{2.9}$$

Lemma 2.7 then follows by substituting (2.9) and Lemma 2.1 into the definition of $P^*(q, Q, e)$. Q.E.D.

LEMMA 2.8. *There exists a constant c whose value is independent of both Q and N such that the expected time of Phase* 2 *of* SEBS is $\leqq c$.

Proof. Let $Z(N)$ have the same meaning as in Lemma 2.5, and $Z^-(N)$ and $Z^+(N)$ denote those $q \in Z(N)$ which satisfy the inequalities $q \leqq 2^{3/2}Q$ and $q > 2^{3/2}Q$ respectively. Let S_1 and S_2 denote the following summands:

$$S_1 = \sum_{q \in Z^-(N)} P^*(q, Q, 0), \tag{2.10}$$

$$S_2 = \sum_{q \in Z^+(N)} P^*(q, Q, e). \tag{2.11}$$

Substituting Lemmas 2.6, 2.7 and the fact that λ is defined to equal Q/q into (2.10) and (2.11), we obtain

$$S_1 \leqq \sum_{q \in Z^-(N)} q/Q, \tag{2.12}$$

$$S_2 \leqq \sum_{q \in Z^+(N)} 2(Q/q)4^{Q/q}. \tag{2.13}$$

Since all $q \in Z^+(N)$ satisfy $q > Q$, (2.13) implies

$$S_2 \leqq \sum_{q \in Z^+(N)} 8Q/q. \tag{2.14}$$

Equations (2.12) and (2.14) imply S_1 and S_2 are each bounded by the summand:

$$\sum_{i=0}^{\infty} 2^{3/2-i} = 2^{5/2}. \tag{2.15}$$

Thus, (2.12)-(2.15) imply:

$$S_1 + S_2 < 2^{7/2}. \tag{2.16}$$

Furthermore, Lemma 2.5 combined with (2.10) and (2.11) implies $S_1 + S_2$ bounds the expected time of Phase 2. Hence, its time is bounded by the constant $2^{7/2}$. Q.E.D.

Comment 2.9. A more elaborate analysis than Lemmas 2.1 through 2.8 can produce a version of Lemma 2.8 with a substantially lower constant. We now present our first main theorem.

THEOREM 2.10. *The expected time of* SEBS *is* $\log\log N + O(1)$.

Proof. Easy, since Phase 1 of SEBS never requires more time than $\lfloor \log\log N \rfloor + 1$ and Lemma 2.8 shows Phase 2's additional cost is $O(1)$. Q.E.D.

Comment 2.11. The proof of SEBS's time $\log\log N + O(1)$ deliberately had a larger coefficient in the O-notation than necessary for the sake of simplifying the presentation. The actual time of this protocol is unlikely to appear in a closed algebraic form, and we would expect it to be crudely approximated as $\log\log N + e$.

Comment 2.12. Let $\log^{(1)} p$, $\log^{(2)} p$, $\log^{(3)} p, \cdots$ be abbreviations for the terms $\log p$, $\log\log p$, $\log\log\log p$, etc., and let $\log^* p$ denote the least integer i such that $\log^{(i)} p \leqq 1$. Define $LS(p, i)$ to be the quantity $\log^* p + \sum_{j=i}^{(\log^* p)} \log^{(j)} p$. Bentley and Yao [BY76] have shown that the quantity $LS(p, 1) + O(1)$ upper bounds the number of

greater-than comparisons needed to determine the value of a positive integer p for the difficult deterministic search problem where no upper bound on p's range of allowed values is initially known (the absence of such an upper bound makes binary search impossible). Their algorithm is relevant to multiple access selection conflicts because some selection problems contain no initially known upper bound N on Q's range of permissible values. In particular, we will use in such cases an algorithm called QSEBS, which is the same as SEBS except that it replaces the binary search in Phase 1 of SEBS with an application of the Bentley-Yao algorithm to guess which power of 2 best approximates Q's value. Following the guess that this power of 2 is say the number q^*, Phase 2 of QSEBS will use a procedure identical to SEBS, that repeatedly performs the operation TEST (q^*) until the multiple access channel issues the signal 1. The analysis of QSEBS is similar to that for SEBS, and it shows the procedure runs in time $LS(Q, 2) + O(1)$.

Comment 2.13. Some possible interesting modifications of SEBS or QSEB would be variants whose Phase 1 seeks to guess the closest power of $(1+\varepsilon)$ (rather than of 2) approximating Q's value. Indeed one could take the limit of this method as ε approaches zero for SEBS and thereby eliminate the need for Phase 2. Some preliminary simulations show that both variations of SEBS are efficient and neither is preferable for all values of Q. Both variations of SEBS can be proven to be optimal up to an additive constant, and our discussion has focussed on the power of 2 variation because its proof is shorter. Curiously, both variations can be proven to have distinctly suboptimal constants (because they do not correct adequately for some repeated sequences of "0" or "e" signals). At present the best protocol for controlling the additive constant is not known, and it is an open question. The clearly curious aspect of SEBS is that the next section will prove this protocol to be optimal up to an additive constant despite the extreme simplicity of its definition.

3. A log log N lower bound. A protocol is called *fair* iff all nodes desiring the channel attempt to broadcast with the same probability during each round. In general, any protocol is "fair" iff its trial during the ith round is an operation of the form TEST (q), for some $q \geqq 1$. All the protocols from the last section were thus fair.

In this section, we again assume that N nodes are connected to a multiple access channel and that Q of these N nodes wish to broadcast, for some positive integer $2 \leqq Q \leqq N$ whose value is initially unknown to our protocol. For any fair protocol A, we will show there exists some Q such that the protocol requires at least time $\log\log N - O(1)$ to produce the signal "1" when Q nodes desire the channel. This result will show that SEBS is optimal up to an additive constant and that QSEB differs from optimality by a quantity $LS(Q, 3) + O(1)$, which of course is bounded by $o(\log\log Q)$. Incidentally, the same result can be proven for protocols not necessarily fair with a more complicated analysis.

In our discussions s will denote a sequence of "0" and "e" signals whose length is denoted as $|s|$. Henceforth, A will denote an arbitrarily chosen fair protocol, and $q(s)$ the value of q which this protocol chooses immediately after the signal sequence s. That is, the $(|s|+1)$st test will consist of TEST $(q(s))$. This notation implies that each of the Q nodes desiring the channel will broadcast "1" with a probability $1/q(s)$ during the $(|s|+1)$st round, and that (2.2) gives the resulting probability of $P(q(s), Q, 1)$. The following two lemmas help establish our lower bound.

LEMMA 3.1. *Assume* $q(s) \geqq 2$. *Then* $P(q(s), Q, 1) \leqq 2 \cdot Q\, e^{-Q/q(s)}/q(s)$.

Proof. Since $\ln(1-1/q(s)) < -1/q(s)$, (3.1) must hold

$$[1-1/q(s)]^Q < e^{-Q/q(s)}. \tag{3.1}$$

Since $q(s) \geqq 2$, this inequality implies

$$[1-1/q(s)]^{Q-1} < 2\, e^{-Q/q(s)}. \tag{3.2}$$

The lemma now follows by substituting (3.2) into (2.2). Q.E.D.

LEMMA 3.2. $\sum_{Q=2}^{N} P(q(s), Q, 1)/Q \leqq 2$.

Proof. Lemma 3.1 implies

$$\sum_{Q=2}^{N} P(q(s), Q, 1)/Q \leqq 2 \cdot \sum_{Q=2}^{N} e^{-Q/q(s)}/q(s). \tag{3.3}$$

The assertion now follows because the right side of (3.3) is clearly less than

$$2 \int_0^{\infty} e^{-x/q(s)}\, dx/q(s) = 2. \tag{3.4}$$

Q.E.D.

In the remainder of our discussion, Z will denote the set of all sequences of the signals "0" and "e". Z^{INF} will denote the subset of Z whose sequences have lengths $\leqq (\log\log N)-1$, and Z^{SUP} the remaining elements whose length is $>$ this quantity; in other words, (3.5) and (3.6) define the two sets

$$Z^{\mathrm{INF}} = \{s \in Z \,|\, |s| \leqq (\log\log N)-1\}, \tag{3.5}$$

$$Z^{\mathrm{SUP}} = \{s \in Z \,|\, |s| > (\log\log N)-1\}. \tag{3.6}$$

If A is a protocol, Q the number of nodes desiring the critical resource and s a signal sequence, then $\mathrm{PROB}_A(s, Q)$ will denote the probability that A produces the sequence s when Q nodes desire the resource. Under this notation, the probability of transmitting the sequence s immediately followed by the signal "1" on the multiple access channel is:

$$\mathrm{Prob}_A(s, Q) \cdot P(q(s), Q, 1). \tag{3.7}$$

Let $P_A^{\mathrm{INF}}(Q)$ denote the probability that A emits the signal "1" in $\leqq \lfloor \log\log N \rfloor$ tests when Q nodes wish to use the multiple access channel. Equations (3.5) and (3.7) imply

$$P_A^{\mathrm{INF}}(Q) = \sum_{s \in Z^{\mathrm{INF}}} \mathrm{Prob}_A(s, Q) \cdot P(q(s), Q, 1). \tag{3.8}$$

Similarly, the quantity $P_A^{\mathrm{SUP}}(q)$ below is the probability that more than $\lfloor \log\log N \rfloor$ tests are required.

$$P_A^{\mathrm{SUP}}(Q) = \sum_{s \in Z^{\mathrm{SUP}}} \mathrm{Prob}_A(s, Q) \cdot P(q(s), Q, 1). \tag{3.9}$$

Let $E_A(Q)$ denote the expected number of tests performed by the protocol A when Q nodes desire the critical resource. Another consequence of our notation is that

$$E_A(Q) = \sum_{s \in Z} (|s|+1) \cdot \mathrm{Prob}_A(s, Q) \cdot P(q(s), Q, 1). \tag{3.10}$$

Since all members of Z^{SUP} satisfy $|s|+1 \geqq \lfloor \log\log N \rfloor$, (3.10) implies

$$E_A(Q) \geqq \lfloor \log\log N \rfloor \cdot P_A^{\mathrm{SUP}}(Q) + \sum_{s \in Z^{\mathrm{INF}}} (|s|+1) \cdot \mathrm{Prob}_A(s, Q) \cdot P(q(s), Q, 1). \tag{3.11}$$

Henceforth, $T_A(Q)$ will denote the summand

$$T_A(Q) = \sum_{s \in Z^{\mathrm{INF}}} \{\{\lfloor \log\log N \rfloor - |s| - 1\} \cdot \mathrm{Prob}_A(s, Q) \cdot P(q(s), Q, 1)\}. \tag{3.12}$$

Since $P_A^{\mathrm{SUP}}(Q) = 1 - P_A^{\mathrm{INF}}(Q)$, (3.11) and (3.12) imply

$$E_A(Q) \geqq \lfloor \log \log N \rfloor - T_A(Q). \tag{3.13}$$

Equation (3.13) shows an upper bound on $T_A(Q)$'s value will produce a lower bound on $E_A(Q)$'s value. The next three propositions will use this type of analysis to construct $E_A(Q)$'s lower bound.

LEMMA 3.3. *Let T_A^N denote the aggregate defined below*:

$$T_A^N = \sum_{Q=2}^{N} T_A(Q)/Q. \tag{3.14}$$

Then $T_A^N \leqq 2 \cdot \lfloor \log N \rfloor$.

Proof. Since $\mathrm{Prob}_A(s, Q) \leqq 1$, (3.12) implies

$$T_A(Q) \leqq \sum_{s \in Z^{\mathrm{INF}}} \{\lfloor \log \log N \rfloor - |s| - 1\} \cdot P(q(s), Q, 1). \tag{3.15}$$

Equation (3.14), (3.15) and Lemma 3.2 then imply

$$T_A^N \leqq 2 \cdot \sum_{s \in Z^{\mathrm{INF}}} \{\lfloor \log \log N \rfloor - |s| - 1\}. \tag{3.16}$$

The definition of Z^{INF} implies

(i) if $i \leqq (\log \log N) - 1$ then Z^{INF} has precisely 2^i sequences of length i; and

(ii) Z^{INF} has no sequences of length $> (\log \log N) - 1$.

Therefore, (3.16) implies

$$T_A^N \leqq 2 \cdot \sum_{i=0}^{\lfloor \log \log N \rfloor - 1} (\lfloor \log \log N \rfloor - i - 1) 2^i \leqq 2 \cdot \lfloor \log N \rfloor. \tag{3.17}$$

Q.E.D.

LEMMA 3.4. *Let t_A^N denote the minimum defined below*:

$$t_A^N = \underset{2 \leqq Q \leqq N}{\mathrm{MIN}} T_A(Q). \tag{3.18}$$

Then $t_A^N \leqq 2 \cdot \lfloor \log N \rfloor / (\ln (N+1) - \ln 2)$.

Proof. Equation (3.14) and standard techniques from calculus imply

$$T_A^N \geqq \sum_{Q=2}^{N} (t_A^N / Q) > t_A^N \cdot [\ln (N+1) - \ln 2]. \tag{3.19}$$

Since Lemma 3.3 indicated $T_A^N \leqq 2 \cdot \lfloor \log N \rfloor$, (3.18) implies $t_A^N \leqq 2 \cdot \lfloor \log N \rfloor / [\ln (N+1) - \ln 2]$. Q.E.D.

COROLLARY 3.5. $t_A^N \leqq 2/(\ln 3 - \ln 2)$.

Proof. An immediate consequence of Lemma 3.4 is that its declared value of t_A^N always satisfies this inequality. Q.E.D.

We now prove the main result of this section.

THEOREM 3.6. *Assume it is known that $2 \leqq Q \leqq N$ and that A is a fair protocol which terminates when the signal "1" is transmitted. Then there exists a constant c whose value is independent of A and N such that $E_A(Q) \geqq \lfloor \log \log N \rfloor - c$, for at least some Q.*

Proof. Let Q^* denote that particular Q with $t_A^N = T_A(Q^*)$. Then (3.13) and Corollary 3.5 imply

$$E_A(Q^*) \geqq \lfloor \log \log N \rfloor - t_A^N \geqq \lfloor \log \log N \rfloor - 2/(\ln 3 - \ln 2). \tag{3.20}$$

The latter equation shows one constant c satisfying Theorem 3.6 is $2/(\ln 3 - \ln 2)$.

Comment 3.7. A more refined analysis will show c's value is usually much lower. For instance, Lemma 3.4 implies $\lim_{N \to \infty} t_A^N = 2/\ln 2$, and a further factor 2 decrease

holds as Q approaches infinity in Lemma 3.1. These refinements reduce c's value to $1/\ln 2$, and other refinements were also omitted for the sake of brevity.

Comment 3.8. The combination of Theorems 2.10 and 3.6 indicate that SEBS provides a solution for the N-bounded selection problem that is optimal up to an additive constant. Theorem 3.6 implies that every protocol for the unbounded variant of this problem requires time $\log\log Q - O(1)$ for infinitely many different values Q. Thus QSEB's expected time $\mathrm{LS}(Q, 2) + O(1)$ differs from this criteria for optimality by $\leqq \mathrm{LS}(Q, 3) + O(1)$, which of course is asymptotically $\ll$ both $\log\log Q$ and $\mathrm{LS}(Q, 2)$. It is likely that our techniques could be combined with [BY76] to show that QSEB differs from optimality by a slightly smaller asymptote proportional to $\log^* Q$. Both [BY76] and the subsequent literature (for instance [Ri84], [RV80], [St80]) reported quantities at least as large as $o(\log^* i)$ separating the known upper and lower bounds for the deterministic unbounded search problem, and it is therefore unclear whether QSEB falls within precisely an additive constant of optimality.

4. Extensions. There are four possible generalizations of the results from §§ 2 and 3.

First, consider using p multiple access devices in parallel to connect the N nodes. These parallel devices could collectively constitute a hardware semaphore device that performs selection resolution in time $(\log\log N)/\log p$. This improvement in time would probably not justify many additional lines because $\log p$ is a very slowly growing function.

The second possibility would be a stripped communication device that only transmits two signals of "0" and "*". The latter signal would mean the system is either in the state "1" or "e". It would be impossible to resolve conflicts with 100% certainty with such a device. However, they could be resolved in time $(\log\log N) + O(|\log \varepsilon|)$ with certainty $1 - \varepsilon$.

Finally, consider p devices hooked in parallel, each with the communication ability $\{0, *\}$. Then the time for achieving the certainty $1 - \varepsilon$ would be $(\log\log N)/\log p + O[|\log \varepsilon|/p]$. As a mathematician, I cannot predict whether electrical engineers will ever use such a technology. However, parallel communication lines are certainly more tempting in the third case than in the first.

It may be useful to apply our protocols to an environment where various different processors compete for a resource by stating the priority of their request in the form of an integer between 1 and M. An easy generalization of § 2 shows the priority problem can be resolved in time $\log(M \cdot \log N)$.

Acknowledgment. I warmly thank my Bell Labs supervisor, B. Gopinath, for suggesting I study applications of randomized algorithms to semaphore-like problems.

REFERENCES

[BFJLP78] J. BURNS, M. FISCHER, P. JACKSON, N. LYNCH AND G. PETERSON, *Shared data requirements for implementing mutual exclusion using test-and-set primatives*, Univ. Washington report F8-08-08, Seattle, WA, 1978.

[BY76] J. BENTLEY AND A. YAO, *An almost optimal algorithm for unbounded search*, Inform. Proc. Lett., 5 (1976), pp. 82–87.

[Ca79] J. CAPETAANNKIS, *Tree algorithms for packet broadcast channels*, IEEE Trans. Inform. Theory, IT-25 (1979), pp. 505–515.

[Di65] E. DIJKSTRA, *Solutions of a problem in concurrent programming control*, Comm. ACM, 9 (1965), p. 569.

[Fe68] W. FELLER, *Introduction to Probability Theory*, John Wiley, New York, 1968.

[Gr83] A. GREENBERG, *Efficient algorithms for multiple access channels*, Ph.D. thesis, Univ. Washington, Seattle, WA, 1983.

[GL83] A. GREENBERG AND R. LADNER, *Estimating the multiplicity of conflicts in multiple access channels*, 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 323-392.

[GW85] A. GREENBERG AND S. WINOGRAD, *A lower bound on time needed in the worst case to resolve conflicts deterministically in multiple access channel*, J. Assoc. Comput. Mach., to appear.

[Kn65] D. KNUTH, *Additional comments on a problem in concurrent control*, Comm. ACM, 9 (1966), p. 321.

[Ma80] J. L. MASSEY, *Collision resolution algorithms and random communications*, Tech. Report UCLA-Eng-8016, Univ. California, Berkeley, 1980.

[Me73] R. METCALFE, *Steady state analysis of slotted and controlled Aloha systems with blocking*, Proc. 6th Hawaii Conference on System Sciences, 1973, pp. 375-380.

[MB76] R. METCALFE AND D. BOGGS, *Ethernet: distributed packet switching for local computer networks*, Comm. ACM, 19 (1976), pp. 395-404.

[MT80] V. A. MIKHAILOV AND B. S. TSYBAKOV, *Upper bound for the capacity of a random multiple access system*, Problomy Peredaihi Informatsii, 17 (1981), pp. 90-95; translated into English in *Problems of Information Transmission.*

[Pi79] N. PIPPENGER, *Bounds on the performance of protocols on multiple-access broadcast channels*, IBM report RC7742(33525), 1979.

[Ra80] M. RABIN, *N-process synchronization by 4 log N-valued shared variables*, 21st IEEE Symposium on Foundations of Computer Science, 1980, pp. 407-410.

[Ri84] J. RISSANEN, *Universal coding information, prediction and estimation*, IEEE Trans. Inform. Theory, IT-30 (1984), pp. 629-636.

[RV80] J. RAOULT AND J. VUILLEMIN, *Optimal unbounded search strategies*, 7th International Colloquium on Automation and Computing and Programming, 1980, pp. 512-529.

[SH80] J. SHOCH AND J. HUPP, *Measured performance of an ethernet local network*, Comm. ACM, 23 (1980), pp. 711-721.

[St80] Q. STOUT, *Improved prefix encodings of natural numbers*, IEEE Trans. Inform. Theory, IT-26 (1980), pp. 607-610.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
011

EFFICIENT ALGORITHMS FOR GEOMETRIC GRAPH SEARCH PROBLEMS*

HIROSHI IMAI† AND TAKAO ASANO†‡

Abstract. In this paper, we show that many graph search problems can be solved quite efficiently for a geometric intersection graph of horizontal and vertical line segments. We first extract several basic operations for depth first search and breadth first search on a graph. Then we present data structures for the intersection graph in terms of which those operations can be implemented in an efficient manner. The data structures enable us to solve various graph search problems besides depth first search and breadth first search. Specifying the results obtained in this paper for an intersection graph of n horizontal and vertical segments with m pairs of intersecting segments, we obtain algorithms with the following complexity, where $N = \min\{m, n \log n\}$.

(i) Depth first search and breadth first search can be executed in $O(n \log n)$ time and $O(N)$ space.

(ii) The biconnected components can be found in $O(n \log n)$ time and $O(N)$ space.

(iii) A maximum matching and a maximum independent set can be found in $O(\sqrt{n}\, N)$ time and $O(N)$ space when no two horizontal (vertical) segments intersect.

(iv) The connectivity k_G can be found in $O(k_G n^{3/2} N)$ time and $O(N)$ space.

Our algorithms can be applied to various practical problems such as the problem of finding a minimum dissection of a rectilinear region, which arises in the manipulation of VLSI artwork data, and the problem of determining whether there is a Manhattan wiring on a single layer, which arises in the design automation of digital systems.

Key words. computational geometry, segment tree, orthogonal segment intersection search, intersection graph, graph algorithms, depth-first search, linear-time set union algorithm

1. Introduction. In most graph algorithms, graphs are represented by adjacency lists (e.g., see [1]), and the complexity of each algorithm is estimated on the basis of them. However, for graphs with some prescribed properties, it may be possible to design efficient graph algorithms by employing other data structures which make use of those properties. In this paper, we show that this is the case for some geometric intersection graphs, where an intersection graph of objects in the plane is obtained by identifying each object with a vertex, and connecting two vertices by an edge iff their corresponding objects intersect. Specifically, we consider problems for an intersection graph of horizontal and vertical line segments. In Fig. 1.1, an example is shown.

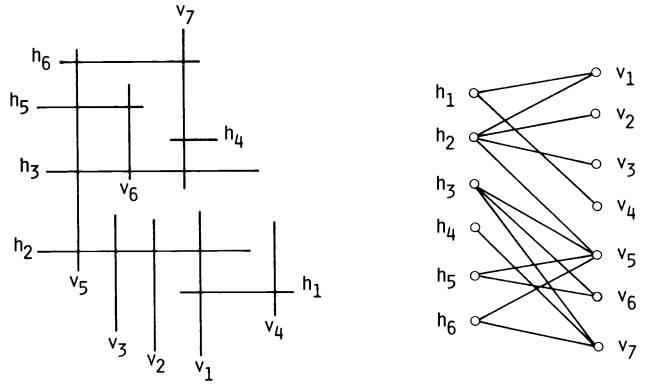
We first extract several basic operations for a graph which are required in executing depth first search and breadth first search. The operations are mainly to find, for a vertex, an arc emanating from it (this operation is called LIST1), and to delete the incoming arcs incident to the vertex (this is called DELETE). These are a bit different from ordinary implementations in deleting arcs coming into a vertex from the graph in place of labeling the vertex. That is, in ordinary implementations for depth first search and breadth first search, a vertex which has been searched is *labeled* (or, *marked*) so that the vertex will never be searched again. However, in our implementations, we delete arcs coming into the vertex from the graph for this purpose.

We then present efficient data structures for the intersection graph so that the basic operations can be executed quickly. Our data structures are based on those for intersection problems of horizontal and vertical line segments in computational geometry, where the segment tree as introduced by Bentley [2] plays an important

* Received by the editors December 8, 1983, and in revised form December 28, 1984.

† Department of Mathematical Engineering and Instrumentation Physics, Faculty of Engineering, University of Tokyo, Tokyo, Japan 113.

‡ Current address. Department of Mechanical Engineering, School of Science and Technology, Sophia University, Tokyo, Japan 102.



(a) *Horizontal and vertical segments.* (b) *Intersection graph.*

FIG. 1.1. *An example.*

role, and further on a linear-time set union algorithm developed by Gabow and Tarjan [11]. For an intersection graph of n horizontal and vertical segments, our data structures enable us to execute, on-line, a sequence of $O(n)$ LIST1's and DELETE's in $O(n \log n)$ time and space.

We can develop efficient algorithms not only for depth first search and breadth first search but also for various other graph problems. Specifying the results obtained in this paper for an intersection graph of n horizontal and vertical segments with m pairs of intersecting segments, we have the following, where $N = \min\{m, n \log n\}$.

(i) Depth first search and breadth first search can be executed in $O(n \log n)$ time and $O(N)$ space.

(ii) The biconnected components can be found in $O(n \log n)$ time and $O(N)$ space.

(iii) A maximum matching and a maximum independent set can be found in $O(\sqrt{n}\, N)$ time and $O(N)$ space when no two horizontal (vertical) segments intersect.

(iv) The connectivity k_G can be found in $O(k_G n^{3/2} N)$ time and $O(N)$ space.

These results can be applied to various practical problems, among which the following two practical problems are taken up in the paper.

(I) The problem of finding a minimum dissection of rectilinear region which consists of l sides and may have holes (Lipski et al. [16], Ohtsuki [20]). This problem arises in manipulation of VLSI artwork data. By applying our maximum matching algorithm, we can solve the problem in $O(l^{3/2} \log l)$ time and $O(l \log l)$ space (in fact, we can solve the problem a bit more efficiently as discussed in § 7.1).

(II) The problem of determining whether or not a set of n point pairs can be wired in Manhattan fashion on a single layer (Masuda et al. [17], Raghavan et al. [21]). We can solve this problem in $O(n \log n)$ time.

2. Extracting basic operations for depth first search and breadth first search. We first consider, as an example, the depth first search of a directed graph $G = (V_G, A_G)$ with vertex set V_G and arc set A_G. Let $n = |V_G|$ and $m = |A_G|$. An undirected graph $G = (V_G, E_G)$ with vertex set V_G and edge set E_G is regarded as a directed graph obtained from G by replacing each edge by two reversely-oriented arcs which connect the same vertices. In Fig. 2.1, we give a procedure SEARCH for finding a depth first spanning forest, which is represented by $p(\ )$, and computing *dfnumber*( ) (Tarjan [24]).

```
procedure SEARCH;
  procedure DFS (u);
    begin
      dfnumber (u) := k; k := k+1;
      while there is an arc (u, v) ∈ A_G with v ∈ W do
        begin
          remove v from W; p(v) := u;
          DFS (v)
        end
    end;
begin
  k := 1; W := V_G;
  while W ≠ ∅ do
    begin
      take an element w out of W; p(w) := nil;
      DFS (w)
    end
end;
```

FIG. 2.1. *The procedure* SEARCH.

In ordinary implementations of the procedure, a vertex removed from W is *labeled* (or, *marked*), and, in DFS (u), we scan each arc (u, v) going out of u and check whether v is labeled or not. If this technique is employed and the graph is represented by adjacency lists, the procedure can be executed in $O(m)$ time and space.

We can implement the procedure in a different way. For this purpose, several procedures are introduced. For $W \subseteq V_G$, we denote by $G(W)$ the graph obtained from G by deleting arcs coming into vertices in $V_G - W$. A data structure representing $G(W)$ for $W \subseteq V_G$ is kept in the course of the algorithm. For $v \in W$, DELETE (v, W) is a procedure which makes $W := W - \{v\}$ and updates the data structure for $G(W)$. For $u \in V_G$, LIST1 (u, W) is a function which returns a vertex v such that (u, v) is an arc in $G(W)$ if such v exists, and returns nil otherwise. Also, for $u \in V_G$, LIST_DEL (u, W) is a function which returns a set $\{v | (u, v)$ is an arc in $G(W)\}$ and executes DELETE (v, W) for each v in the set.

The procedure SEARCH can be executed by DELETE and LIST1. That is, in the main part of the procedure, we execute DELETE (w, W), and, in DFS (u), we execute $v := \text{LIST1}\ (u, W)$ and DELETE (v, W). LIST1 and DELETE are executed $O(n)$ times and the other parts of the procedure take $O(n)$ time. Hence, we obtain the following.

PROPOSITION 2.1. *Suppose that a sequence of* $O(n)$ LIST1*'s and* DELETE*'s can be executed, on-line, in* $g_T(n, m)(=\Omega(n))$ time and $g_S(n, m)(=\Omega(n))$ *space. Then, the depth first search can be executed in* $O(g_T(n, m))$ *time and* $O(g_S(n, m))$ *space.*

Of course, LIST1 and DELETE can be executed by representing G with adjacency lists. In such a case, we trivially have $g_T(n, m)$, $g_S(n, m) = O(m)$. However, if G has some prescribed properties, it may be possible to make $g_T(n, m)$ and $g_S(n, m)$ smaller. In § 3, we show that this is the case for an intersection graph of horizontal and vertical segments.

Similar techniques can be applied to the breadth first search. The problem we consider is to execute a breadth first search on G from a set S of vertices $(S \subseteq V_G)$, which computes, for each $v \in V_G$, the length $d(v)$ of a shortest directed path from S to v in G. Here, the length of a directed path is the number of arcs contained in it. Note that $d(s) = 0$ for $s \in S$, and $d(v) = \infty$ if there is no directed path from a vertex in S to v. In Fig. 2.2, a procedure BFS is given, where Q is a queue consisting of vertices in V_G.

```
procedure BFS;
  begin
    for each s ∈ S do d(s) := 0;
    let Q be a queue consisting of all vertices in S; W := V_G − S;
    while Q ≠ ∅ do
      begin
        take an element u out of Q; V_u := LIST_DEL(u, W);
        for each v ∈ V_u do begin d(v) := d(u)+1; insert v into Q end
      end
  end;
```

FIG. 2.2. *The procedure* BFS (*breadth first search from the set S of vertices*).

Then, we have the following, where it is noted that a sequence of $O(n)$ LIST_DEL's can be executed by a sequence of $O(n)$ LIST1's and DELETE's.

PROPOSITION 2.2. *Suppose that a sequence of* $O(n)$ LIST1*'s and* DELETE*'s can be executed in* $g_T(n, m)(=\Omega(n))$ *time and* $g_S(n, m)(=\Omega(n))$ *space. Then, the breadth first search can be executed in* $O(g_T(n, m))$ *time and* $O(g_S(n, m))$ *space.*

3. The data structures. Let H (resp. V) be a set of horizontal (resp. vertical) line segments in the (x, y)-plane. Let $n_h = |H|$, $n_v = |V|$ and $n = n_h + n_v$. We denote by $G_S(H, V)$ the intersection graph of these horizontal and vertical segments. We say that $G_S(H, V)$ is *geometrically bipartite* (abbreviated as *g-bipartite*) if there are no pair of intersecting horizontal segments and no pair of intersecting vertical segments. $G_S(H, V)$ is an undirected graph, and, if $G_S(H, V)$ is g-bipartite, $G_S(H, V)$ is a bipartite graph.

In this section, we present efficient data structures for $G_S(H, V)$, which are based both on the data structures for a special kind of one-dimensional range search problem and on segment trees, so that LIST1 and DELETE introduced in § 2 can be executed quickly. We first consider the problems for g-bipartite graph $G_S(H, V)$, and then consider the general case.

3.1. A special kind of one-dimensional range search problem. Suppose that, on the x-axis, a set S of points $P_i = (x_i)$ $(i = 1, \cdots, p)$ and a set I of intervals (ranges) $R_k = [l_k, r_k]$ $(k = 1, \cdots, q)$ are given. It is also supposed that x_i $(i = 1, \cdots, p)$ are distinct, and that the values of x_i, l_k and r_k are given in a sorted order by $O((p+q)\log(p+q))$-time preprocessing. Further, for brevity, we provide dummy points P_0 and P_{p+1} in S such that $x_0 \equiv -\infty$ and $x_{p+1} \equiv +\infty$. For these points and intervals, we are concerned with the problem of executing the following procedures DEL and L1 efficiently. For $P_i \in S$, DEL (P_i, S) is a procedure which removes P_i from the set S. For $R_k \in I$, L1(R_k, S) is a function which returns a point P_i in S contained in the interval R_k if such a point exists, and returns nil otherwise. In fact, L1(R_k, S) given below returns a point which is leftmost among points included in R_k.

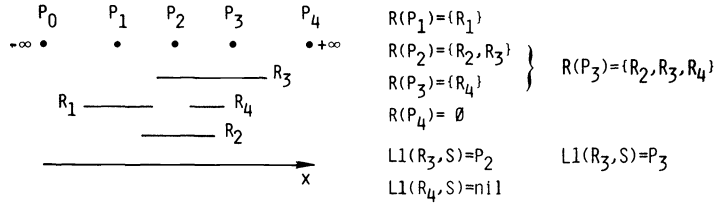
By means of the set union algorithm developed by Gabow and Tarjan [11], we can execute a *sequence* of $O(r)$ L1's and DEL's in $O(p+q+r)$ time as follows. We keep the set S of points P_i by a doubly linked list L in increasing order with respect to their x-coordinates. For $P_i \in S$, let $P_{p(i)}$ be the predecessor element of P_i in the list L. $P_{p(i)}$ is a point that is rightmost among points in S lying in the left side of P_i. Then, define $R(P_i)$ to be a set of all intervals $R_k \in I$ whose left endpoints lie in the interval $(x_{p(i)}, x_i]$ (i.e., $l_k \in (x_{p(i)}, x_i]$). By definition, $R(P_i)$'s $(P_i \in S)$ are disjoint sets. For a pair of consecutive points P_i and P_j in the list L, consider a procedure UNION (P_i, P_j) which makes $R(P_j) := R(P_i) \cup R(P_j)$ and $R(P_i) := \emptyset$. For $R_k \in I$, consider a function FIND (R_k) which returns a point $P_i \in S$ such that $R_k \in R(P_i)$, where P_i is uniquely determined (recall that we consider dummy points P_0 and P_{p+1}). Then, DEL and L1 can be executed as in Fig. 3.1.

```
procedure DEL (P_i, S);
  begin
    let P_s(i) be the successor element of P_i in the list L;
    remove P_i from the set S (i.e., the list L);
    UNION (P_i, P_s(i))
  end;
function L1(R_k, S): an element in S;
  begin
    P_i := FIND (R_k); (recall R_k = [l_k, r_k])
    if x_i > r_k then return nil
             else return P_i
  end;
```

FIG. 3.1. *The procedure* DEL *and the function* L1.

In Fig. 3.2, an example is given. The validity of DEL and L1 is evident. Since we know the *structure* of the UNION's in advance (i.e., the union tree is a path, which is the simplest case; see [11]), and UNION and FIND are executed $O(r)$ times in the sequence, we can execute UNION and FIND in $O(p+q+r)$ time in total, with $O(p+q)$ preprocessing time (Gabow and Tarjan [11]). Hence, we obtain the following.

THEOREM 3.1. *A sequence of* $O(r)$ L1*'s and* DEL*'s for p points and q intervals can be executed in* $O(p+q+r)$ *time, using* $O(p+q)$ *preprocessing time.*



(a) *Points and intervals.* (b) $R(P_i)$ *and* L1. (c) *After* DEL (P_2, S).

FIG. 3.2. *An example.*

3.2. Segmentation. Our data structure below is essentially based on the partitions of segments as induced by segment trees (Bentley [2]), which are now outlined with a slight modification.

We first normalize the y-coordinates of segments in the following way. We shorten vertical segments as much as possible so long as, for each vertical segment, the set of intersecting horizontal segments does not change. Supposing that there are n_y distinct y-coordinates of endpoints of segments, we then normalize the y-coordinates of segments by replacing them by their ranks from 1 to n_y in the set (there, of course, may be ties). Similarly, the x-coordinates of segments are normalized.

For an integer interval $[a, b]$, a segment tree $T(a, b)$ consists of a root r with interval $[a, b]$, and in the case of $b-a \geqq 1$, of a left subtree $T(a, \lfloor (a+b)/2 \rfloor)$ and a right subtree $T(\lfloor (a+b)/2 \rfloor + 1, b)$; in the case of $b-a=0$, the left and right subtrees are empty. In Fig. 3.3, the tree $T(1, 6)$ is depicted, where each node is labeled with its associated interval. We call an interval which equals interval $[i, j]$ associated with some node of $T(1, n_y)$ a *standard interval*, and denote it by $I_{i,j}$. The partition of a segment $[a, b]$ $(1 \leqq a \leqq b \leqq n_y)$ is a collection of standard intervals contained in $[a, b]$ such that

(i) each of $a, a+1, a+2, \cdots, b$ is in exactly one standard interval in the collection, and

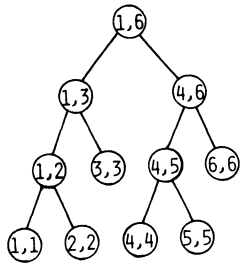


FIG. 3.3. *The segment tree* $T(1, 6)$.

(ii) for each standard interval in the collection, which is associated with node n_T in $T(1, n_y)$, a standard interval associated with a father of n_T in $T(1, n_y)$ is not contained in $[a, b]$.

In Fig. 3.4, the partitions of vertical segments depicted in Fig. 1.1(a) are shown. The partition of a segment consists of $O(\log n_y)$ standard intervals, and can be obtained in $O(\log n_y)$ time.

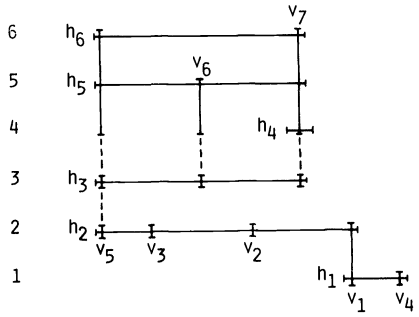


FIG. 3.4. *The partitions of vertical segments given in Fig.* 1.1(a).

3.3. Executing DELETE, LIST1 and LIST_DEL. Since a sequence of $O(n)$ LIST_DEL's can be replaced by a sequence of $O(n)$ LIST1's and DELETE's, we only consider a sequence of $O(n)$ LIST1's and DELETE's in the following. We first consider how to execute LIST1$(h, H \cup V)$ for $h \in H$ and DELETE$(v, H \cup V)$ for $v \in V$ in g-bipartite $G_S(H, V)$. LIST1 for $v \in V$ and DELETE for $h \in H$ can be executed similarly. Consider the partitions of all the vertical segments in V. Then, with each standard interval $I_{i,j}$, we can associate a set $V_I(I_{i,j})$ of vertical segments whose partitions contain the standard interval $I_{i,j}$. (In Fig. 3.4, $V_I(I_{4,6}) = \{v_5, v_7\}$.) For each horizontal segment h, consider a set $I_H(h)$ of standard intervals $I_{i,j}$ such that the y-coordinate of h is contained in the standard interval $I_{i,j}$. (In Fig. 3.4, $I_H(h_2) = \{I_{1,6}, I_{1,3}, I_{1,2}, I_{2,2}\}$.) For a standard interval $I_{i,j}$, consider a set $H_I(I_{i,j})$ of horizontal segments $h \in H$ such that the y-coordinate of h is contained in the standard interval $I_{i,j}$. (In Fig. 3.4, $H_I(I_{4,6}) = \{h_4, h_5, h_6\}$.)

For each standard interval $I_{i,j}$, by projecting the set $V_I(I_{i,j})$ of vertical segments and the set $H_I(I_{i,j})$ of horizontal segments on the x-axis, we can identify each vertical segment with a point and each horizontal segment with an interval on the x-axis. Therefore, we can consider DEL$(v, V_I(I_{i,j}))$ for $v \in V_I(I_{i,j})$ and L1$(h, V_I(I_{i,j}))$ for $h \in H_I(I_{i,j})$ in an obvious way. For each standard interval $I_{i,j}$, we keep the data structure, which is discussed in § 3.1, for the set $V_I(I_{i,j})$ of points and the set $H_I(I_{i,j})$ of intervals in order to execute DEL$(v, V_I(I_{i,j}))$ and L1$(h, V_I(I_{i,j}))$. The data structure can be

constructed in $O(n \log n)$ time, and takes $O(n \log n)$ space in total. Then, DELETE and LIST1 can be executed as in Fig. 3.5.

```
procedure DELETE (v, H ∪ V);
  begin
    for each I_{i,j} such that v ∈ V_I(I_{i,j}) do DEL (v, V_I(I_{i,j}))
  end;
function LIST1 (h, H ∪ V): an element of V;
  begin
    while I_H(h) ≠ ∅ do
      begin
        let I_{i,j} be an element of I_H(h);
        v := L1 (h, V_I(I_{i,j}));
        if v ≠ nil then return v
                   else I_H(h) := I_H(h) − {I_{i,j}}
      end;
    return nil
  end;
```

FIG. 3.5. DELETE *and* LIST1.

Let us evaluate the time complexity to execute a sequence of $O(n)$ LIST1's for $h \in H$ and DELETE's for $v \in V$ given in Fig. 3.5. First, note that, in LIST1, $|I_H(h)| = O(\log n)$ and, in DELETE, $|\{I_{i,j} | v \in V_I(I_{i,j})\}| = O(\log n)$. Suppose that, in the sequence, L1 and DEL concerning $V_I(I_{i,j})$ and $H_I(I_{i,j})$ are executed $n(I_{i,j})$ times for each standard interval $I_{i,j}$. Then, the total complexity required by the whole LIST1's and DELETE's is the complexity for executing a sequence of $O(n(I_{i,j}))$ L1$(h, V_I(I_{i,j}))$'s and DEL$(v, V_I(I_{i,j}))$'s for all standard intervals $I_{i,j}$. For each $I_{i,j}$, the sequence of L1's and DEL's can be executed in $O(n(I_{i,j}) + |V_I(I_{i,j})| + |H_I(I_{i,j})|)$ time by Theorem 3.1. Since $\sum (n(I_{i,j}) + |V_I(I_{i,j})| + |H_I(I_{i,j})|) = O(n \log n)$ where the summation is taken over all standard intervals $I_{i,j}$, the total complexity to execute the whole L1's and DEL's is $O(n \log n)$. Thus, the sequence of $O(n)$ LIST1's for $h \in H$ and DELETE's for $v \in V$ can be executed in $O(n \log n)$ time and $O(n \log n)$ space. By considering partitions of horizontal segments, a sequence of $O(n)$ LIST1's for $v \in V$ and DELETE's for $h \in H$ can be executed similarly in $O(n \log n)$ time and $O(n \log n)$ space.

Next, consider the case $G_S(H, V)$ is not g-bipartite. In this case, we must find pairs of intersecting horizontal segments and of intersecting vertical ones. Here, we consider the problem for horizontal segments only. First, note that two horizontal segments intersect only if their y-coordinates are the same. The set of horizontal segments is partitioned into disjoint subsets so that two segments are in the same subset iff their y-coordinates are the same. Then, we have only to find pairs of intersecting segments in each subset. The problem for each subset is just the one-dimensional interval intersection problem, for which efficient data structures such as a tile tree in McCreight [18] and an interval tree in Edelsbrunner [6], [7] have been developed. Given a set I of p intervals on a line, these data structures report a set I' of all intervals in I that intersect a query interval in $O(\log p + |I'|)$ time, and delete an interval in I from the data structure in $O(\log p)$ time (in fact, in our case, we can delete it in a constant time). The data structures take $O(p)$ space.

Thus, we obtain the following.

PROPOSITION 3.1. *A sequence of* $O(n)$ LIST_DEL's, LIST1's *and* DELETE's *for* $G_S(H, V)$ *can be executed in* $O(n \log n)$ *time and* $O(n \log n)$ *space.*

The above algorithms may be worse than naive algorithms in case, for instance, $m = \Theta(n)$ (m is the number of arcs in $G_S(H, V)$). However, by processing $G_S(H, V)$

in advance, we can design an algorithm, which is never worse than usual ones in any cases, and is quite efficient in almost all cases.

Given the set of horizontal and vertical segments, we first compute the number m of pairs of intersecting segments, which can be done in $O(n \log n)$ time and $O(n)$ space (Bentley and Ottmann [3]). If $m \leqq n \log n$, the intersection graph $G_S(H, V)$ is constructed by enumerating all pairs of intersecting segments, which can be done in $O(n \log n + m)$ time and $O(n + m)$ space [3], and then the problem is solved by ordinary adjacency lists as the data structures. Otherwise, the problem is solved by the data structures given above. Thus, we obtain the following.

THEOREM 3.2. *A sequence of $O(n)$* LIST_DEL's, LIST1's *and* DELETE's *for an intersection graph of n horizontal and vertical segments with m pairs of intersecting segments can be executed in $O(N)$ time and $O(N)$ space with $O(n \log n)$ preprocessing, where $N = \min\{m, n \log n\}$.*

4. Simple graph search problems. In the following §§ 4–6, we consider that, for a graph G with n vertices and m edges, a sequence of $O(n)$ LIST1's and DELETE's can be executed in $g_T(n, m)$ time and $g_S(n, m)$ space ($g_T(n, m)$, $g_S(n, m) = \Omega(n)$), and estimate the complexity of algorithms for various graph problems. In this section, simple graph search problems are investigated. By Theorem 3.2, for an intersection graph $G_S(H, V)$ of n horizontal and vertical segments with m pairs of intersecting segments, $g_T(n, m)$, $g_S(n, m) = O(N)$ when $O(n \log n)$ preprocessing is done in advance, where $N = \min\{m, n \log n\}$.

4.1. Depth first search, breadth first search and the connected components. By Propositions 2.1 and 2.2, depth first search and breadth first search for the graph G can be executed in $O(g_T(n, m))$ time and $O(g_S(n, m))$ space. Hence, we obtain the following.

THEOREM 4.1. *Depth first search and breadth first search for $G_S(H, V)$ can be executed in $O(n \log n)$ time and $O(N)$ space.*

As is well known, the connected components of G can be found by executing depth first search or breadth first search, hence we obtain the following.

COROLLARY 4.1. *The connected components of $G_S(H, V)$ can be found in $O(n \log n)$ time and $O(N)$ space.*

However, concerning the problem of finding the connected components of $G_S(H, V)$, an $O(n \log n)$-time and $O(n)$-space algorithm is known (Edelsbrunner et al. [8], Imai and Asano [14]) which employs the so-called plane-sweep techniques.

4.2. The biconnected components. We first show that the biconnected components of an undirected graph $G = (V_G, E_G)$ with vertex set V_G and edge set E_G can be found in $O(g_T(n, m))$ time and $O(g_S(n, m))$ space (by the biconnected components, we here mean the decomposition of the vertex set obtained by decomposing G into the biconnected components). We suppose that G is connected. By executing depth first search for the graph, a depth first spanning tree represented by $p(\ )$ and the values of *dfnumber* (u) for $u \in V_G$ are found, which takes in $O(g_T(n, m))$ time and $O(g_S(n, m))$ space. Then, the main problem for the biconnected components is to compute $low(u)$ for each vertex u defined as follows (Tarjan [24]).

$$low(u) = \min\{\{dfmin(u)\} \cup \{low(s) | s \in V_G, p(s) = u\}\}, \quad \text{where}$$

$$dfmin(u) = \min\{dfnumber(v) | v = u \text{ or } \{u, v\} \text{ is an edge in } G \text{ with } v \neq p(u)\}.$$

If the values of $dfmin(u)$ for all u can be computed, we can find in $O(n)$ time and space the values of $low(\ )$ by traversing the tree in postorder, and then we can find

the biconnected components in $O(n)$ time and space [24]. So, the problem is to find the values of $dfmin(u)$ for all vertices u. In Fig. 4.1, an algorithm for this problem is given. Note that each edge $\{u, v\}$ of G is considered to be two reversely-oriented arcs (u, v) and (v, u) in the procedure.

```
procedure COMPUTE_DFMIN;
  begin
    W := V_G; let u be a vertex with dfnumber(u) = 1;
    dfmin(u) := 1; DELETE (u, W);
    for i := 1 to n do
      begin
        let v be a vertex with dfnumber(v) = i; U := LIST_DEL(v, W);
        for each u ∈ U do if v = p(u) then dfmin(u) := dfnumber(u)
                                      else dfmin(u) := i
      end
    end;
```

FIG. 4.1. *The procedure* COMPUTE_DFMIN.

Since it can be considered that the values of $dfnumber(u)$ for $u \in V_G$ have been sorted in executing depth first search, we obtain the following.

PROPOSITION 4.1. *The biconnected components of G can be found in $O(g_T(n, m))$ time and $O(g_S(n, m))$ space.*

Then the following is obtained by applying the proposition to graph $G_S(H, V)$.

THEOREM 4.2. *The biconnected components of $G_S(H, V)$ can be found in $O(n \log n)$ time and $O(N)$ space.*

5. Maximum flow algorithms. In this section, we consider the problem of finding a maximum flow in a directed graph $G = (V_G, A_G)$ with unit capacities such that, for each vertex, there is at most one arc that emanates from it or at most one arc that comes into it. We estimate the complexity of Ford and Fulkerson's and Dinic's algorithms for this problem, based on the complexity to execute the basic operations, LIST1 and DELETE, for graphs obtained by modifying the original graph G in the course of the algorithms.

Let S and T be disjoint subsets of V_G, where it is supposed that, for each vertex in S (resp. T), there is the only one arc emanating from (resp. coming into) it. We consider the problem of finding a maximum (integral) flow from S to T in the graph G with unit capacities. This problem is equivalent to that of finding a maximum vertex-disjoint directed paths from S to T in G. Note that the value of a maximum flow is at most $n/2$.

We first describe the basic part of the maximum flow algorithms. Let f be an integral flow on G, that is, for each arc $a \in A_G$, $f(a) = 0$ or 1, and, for each vertex $v \in V_G - (S \cup T)$, $\sum_{a=(u,v)\in A_G} f(a) = \sum_{a=(v,u)\in A_G} f(a)$. Define an auxiliary graph $G(f)$ with vertex set V_G and arc set $\vec{A} = A_0 \cup A_1$ by $A_0 = \{a | a \in A_G,\ f(a) = 0\}$, $A_1 = \{a' | a \in A_G, f(a) = 1, a'$: reorientation of $a\}$. Let $S(f) = \{s | s \in S,$ there is an arc emanating from s in $G(f)\}$ and $T(f) = \{t | t \in T,$ there is an arc coming into t in $G(f)\}$. If, in $G(f)$, there is a directed path P from $S(f)$ to $T(f)$, we can augment the flow by letting $f(a) := 1$ $(a \in P \cap A_0)$, $f(a) := 0$ $(a' \in P \cap A_1, a$: reorientation of $a')$.

In the maximum flow algorithms, we must execute the basic operations, LIST1 and DELETE for auxiliary graphs $G(f)$. Since $G(f)$ can be obtained by slightly modifying the original graph G, those operations for $G(f)$ can be executed by those for G (it should be noted that, for each vertex v, there is at most one arc a emanating from (coming into) v with $f(a) = 1$ in the graph G). Hence, we have the following.

PROPOSITION 5.1. *Suppose that a sequence of* $O(n)$ LIST1*'s and* DELETE*'s for the graph* G *can be executed in* $g_T(n, m)$ *time and* $g_S(n, m)$ *space. Then a sequence of* $O(n)$ LIST1*'s and* DELETE*'s for the auxiliary graph* $G(f)$ *can be executed in* $O(g_T(n, m))$ *time and* $O(g_S(n, m))$ *space.*

5.1. Ford and Fulkerson's algorithm. This algorithm [10] starts with an appropriate integral flow f, which may be zero flow, and iterates to find a directed path from $S(f)$ to $T(f)$ in $G(f)$ and to augment the flow f until there comes to be no directed path from $S(f)$ to $T(f)$ in $G(f)$. Since the value of a maximum flow is at most $n/2$ in this graph G, a maximum flow can be found by solving the problem of finding a directed path in the auxiliary graph at most $n/2$ times. Since the path-finding problem can be solved by executing depth first search or breadth first search, from Proposition 5.1, we obtain the following.

PROPOSITION 5.2. *Ford and Fulkerson's algorithm finds a maximum flow in the graph* G *in* $O(ng_T(n, m))$ *time and* $O(g_S(n, m))$ *space.*

5.2. Dinic's algorithm. Dinic's algorithm [5] (see also Even and Tarjan [9] and Hopcroft and Karp [13]) consists of *phases*, where the number of phases is $O(\sqrt{n})$ for this type of graph G [9] (see also [13]). Each phase consists of two steps. The first step is to determine the layer of each vertex of $G(f)$ by breadth first search. The second step is to find a *maximal* set of vertex-disjoint directed paths in the layered $G(f)$ by depth first search.

The first step is to execute breadth first search on $G(f)$ from $S(f)$ and compute $d(v)$ for each v as in § 2. If $d(t)=\infty$ for each $t \in T(f)$ (i.e., there is no directed path from $S(f)$ to $T(f)$ in $G(f)$), f is a maximum flow. This breadth first search can be done in $O(g_T(n, m))$ time and $O(g_S(n, m))$ space by Propositions 2.2 and 5.1.

The second step is subtler than the first step, and is executed as follows. Let $d^* = \min\{d(v) | v \in T(f)\}$, which is assumed to be finite. Define $T_d(f)$ to be $\{t | t \in T(f), d(t) = d^*\}$. Then, the second step of the phase is to find a *maximal* set of vertex-disjoint directed paths of length d^* in $G(f)$ from $S(f)$ to $T_d(f)$, where a set is maximal with a given property if it is not properly contained in any set that has the property. For this purpose, we need a new graph $G(f, d)$, which is defined to be a directed graph with vertex set V_G and arc set $\{(u, v) | (u, v)$ is an arc in $G(f)$, $d(v) = d(u)+1\}$. A maximal set of such directed paths in $G(f)$ as mentioned above can be found by executing the depth first search on $G(f, d)$ (for details, see [5], [9], [13]). Then, from the arguments in § 2, we see that the problem is to execute the basic operations, LIST1 and DELETE, for the graph $G(f, d)$. Since, concerning the structure of a graph, $G(f, d)$ is quite different from G, we cannot generally obtain a proposition, similar to Proposition 5.1, for this graph $G(f, d)$. Considering the complexity to execute those basic operations for $G(f, d)$ as well as that for the original graph G, we obtain the following.

PROPOSITION 5.3. *Suppose that* $O(n)$ LIST1*'s and* DELETE*'s for the graph* $G(f, d)$ *can be executed in* $h_T(n, m)$ *time and* $h_S(n, m)$ *space. Then, Dinic's algorithm finds a maximum flow in the graph* G *in* $O(\sqrt{n}(g_T(n, m)+h_T(n, m)))$ *time and* $O(g_S(n, m)+h_S(n, m))$ *space.*

That is, it is a little complicated to implement Dinic's algorithm, since it requires data structures for executing a sequence of LIST1's and DELETE's not only for the graph G but also for the graph $G(f, d)$.

5.3. A maximum matching. In this section, we consider the problem of finding a maximum matching of g-bipartite $G_S(H, V)$. $G_S(H, V)$ is considered to be a directed

bipartite graph $(H, V; A_G)$ with left vertex set H, right vertex set V and arc set $A_G \subseteq H \times V$. For this $G_S(H, V)$, construct a directed graph G by adding copies H' and V' of H and V, respectively, to $G_S(H, V)$, and making an arc (h', h) for each $h \in H$ and an arc (v, v') for each $v \in V$. As is well known, the problem of finding a maximum matching of $G_S(H, V)$ is equivalent to that of finding a maximum integral flow from H' to V' in G with unit capacities, hence the arguments in §§ 5.1 and 5.2 can be applied.

Concerning the time and space complexities $g_T(n, m)$ and $g_S(n, m)$, respectively, for executing a sequence of $O(n)$ LIST1's and DELETE's for this G, by using the data structure for $G_S(H, V)$ presented in § 3, we have $g_T(n, m)$, $g_S(n, m) = O(N)$ with $O(n \log n)$ preprocessing. (Thus, by Ford and Fulkerson's algorithm with the data structure in § 3 and from Proposition 5.2, we can find a maximum matching of $G_S(H, V)$ in $O(nN)$ time and $O(N)$ space.)

Concerning the time and space complexities $h_T(n, m)$ and $h_S(n, m)$, respectively, for executing a sequence of $O(n)$ LIST1's and DELETE's for $G(f, d)$ obtained from this G, we can no more use the data structure in § 3 directly, because it cannot handle the condition on d efficiently. However, we can modify it, in the following way, so as to work well even for the graph $G(f, d)$. For each standard interval $I_{i,j}$, let $D(I_{i,j}) = \{d(h) | h \in H_I(I_{i,j})\}$. For each $k \in D(I_{i,j})$, define $H_I(k, I_{i,j})$ and $V_I(k, I_{i,j})$ by

$$H_I(k, I_{i,j}) = \{h | h \in H_I(I_{i,j}), d(h) = k\}, \qquad V_I(k, I_{i,j}) = \{v | v \in V_I(I_{i,j}), d(v) = k+1\}.$$

$H_I(k, I_{i,j})$'s (resp. $V_I(k, I_{i,j})$'s) for all $k \in D(I_{i,j})$ are disjoint sets. Hence, a sequence of $O(n)$ LIST1's for $h \in H$ and DELETE's for $v \in V$ on $G(f, d)$ can be executed in $O(n \log n)$ time and space by providing, for each standard interval $I_{i,j}$ and $k \in D(I_{i,j})$, the data structure given in § 3.1 for $V_I(k, I_{i,j})$ and $H_I(k, I_{i,j})$. We can construct the data structure in $O(n \log n)$ time (in partitioning $H_I(I_{i,j})$ and $V_I(I_{i,j})$ into $H_I(k, I_{i,j})$'s and $V_I(k, I_{i,j})$'s $(k \in D(I_{i,j}))$, we use a technique similar to the so-called bucket sort). Concerning LIST1 for $v \in V$ and DELETE for $h \in H$, since there is at most one arc going out of v in $G(f, d)$ owing to the bipartite structure of $G_S(H, V)$, we can execute each LIST1 and DELETE in a constant time. Then, concerning $h_T(n, m)$ and $h_S(n, m)$, applying the arguments similar to Theorem 3.2, we have $h_T(n, m)$, $h_S(n, m) = O(N)$. Then, the following is obtained from Proposition 5.3.

THEOREM 5.1. *A maximum matching of g-bipartite $G_S(H, V)$ can be found in $O(\sqrt{n}\, N)$ time and $O(N)$ space by Dinic's algorithm with the data structures presented above.*

5.4. The connectivity. For $G_S(H, V)$, consider the directed graph G obtained by splitting each vertex u to u^- and u^+ and making arc (u^+, u^-), and making arcs (u_i^-, u_j^+) and (u_j^-, u_i^+) iff there is an edge $\{u_i, u_j\}$ in $G_S(H, V)$. The connectivity k_G of $G_S(H, V)$ can be found by solving the problem of finding a maximum flow in the graph G repeatedly. Further, in solving the maximum flow problem on G by Dinic's algorithm, we can execute DELETE and LIST1 for G by those for $G_S(H, V)$, and we can apply the data structures developed here to this problem. That is, we can solve the maximum flow problem on this graph G with unit capacities in $O(\sqrt{n}\, N)$ time and $O(N)$ space.

Even and Tarjan's algorithm [9] solves $O(k_G n)$ times the maximum flow problem on G for $G_S(H, V)$. Thus, we can find the connectivity k_G of $G_S(H, V)$ in $O(k_G n^{3/2} N)$ time and $O(N)$ space. Also, by applying Galil's algorithm [12], we can find k_G in $O(k_G \sqrt{n}\, N \max\{k_G, \sqrt{n}\})$ time and $O((k_G^2 + n)n)$ space (note that, since vertex-disjoint directed paths are concerned, the information of each phase in Dinic's algorithm can be kept in $O(n)$ space not by $\Theta(m)$ space).

Stating the former result as a theorem, we have the following.

THEOREM 5.2. *The connectivity k_G of $G_S(H, V)$ can be found in $O(k_G n^{3/2} N)$ time and $O(N)$ space.*

6. An independent set.

6.1. A maximum independent set of g-bipartite $G_S(H, V)$. As is well known, a maximum independent set of a bipartite graph can be found by any bipartite matching algorithm which finds a minimum cover of the graph, hence we obtain the following.

THEOREM 6.1. *A maximum independent set of g-bipartite $G_S(H, V)$ can be found in $O(\sqrt{n}\, N)$ time and $O(N)$ space.*

6.2. Determining the existence of an independent set of maximally possible size. Let $G=(V_G, E_G)$ be a graph with n vertices as in § 4. Let X be a matching of G. For each $u \in V_G$, we define *mate*(u) to be v if there is an edge $\{u, v\} \in X$ and to be nil otherwise. A vertex u with *mate*(u) = nil is called *unmatched*. Apparently, the size of a maximum independent set of G is at most $n-|X|$. Further, if there is an independent set S of G of size $n-|X|$, then, for each edge $\{u, v\} \in X$, exactly one of $\{u, v\}$ is an element of S, and each unmatched vertex is an element of S.

Masuda et al. [17] gave an $O(m)$-time and $O(m)$-space algorithm for determining whether or not there is an independent set of size $n-|X|$. Their algorithm executes the depth first search on G by stretching alternating paths (an alternating path is a path of edges which are alternately in X and not in X). In this section, we give an efficient implementation of their algorithm, which solves the problem for G in $O(g_T(n, m))$ time and $O(g_S(n, m))$ space.

Following their algorithm, we color vertices as follows. A vertex v that is found to be in every independent set (resp. no independent set) of size $n-|X|$ is colored *green* (resp. *red*). A vertex v contained (resp. not contained) in the current candidates for such an independent set is colored *blue* (resp. *orange*). The other vertices v that are unsearched are colored *white*. In Fig. 6.1, a procedure INDEPENDENT_SET for the problem is given.

The outline of the algorithm is as follows. Since unmatched vertices must be in every independent set of size $n-|X|$, we first color these vertices v green. If vertex v is colored green (in such a case, the procedure GREEN (v) is called), a set V_v of vertices adjacent to v must be colored red. Then, for each $w \in V_v$, vertex *mate*(w) must be colored green, and so, coloring *mate*(w) green, we apply this procedure recursively. It should be noted that, after the execution of GREEN (v) for unmatched vertices v, all vertices colored white are matched. Then we take a vertex v colored white as a candidate, and color v blue. Starting with v, we execute the depth first search (the procedure BLUE (v)). If a vertex u adjacent to v is still colored white, we color u orange and vertex $u' =$ *mate*(u) blue, and recursively execute depth first search from u'. If a vertex u adjacent to v is already colored blue, there is a path $u=u_0, u_1, u_2, \cdots, u_{2k-1}, u_{2k}=v$ such that $u_{2i} =$ *mate*(u_{2i-1}), *color* (u_{2i}) = blue and *color*(u_{2i-1}) = orange (this is because BLUE essentially executes the depth first search). If $u=u_0$ is in an independent set S of size $n-|X|$, u_1 cannot be in S, and then u_2 must be in $S, \cdots$, and $u_{2k}=v$ must be in S. However, there has been found an edge connecting u and v, which contradicts the fact that S is an independent set. Thus, in this case, we see that u cannot be in any independent set of size $n-|X|$, and we color u red, and $u' =$ *mate*(u) green, and execute GREEN (u').

In the procedure, we consider operations, LIST_DEL and DELETE, for the set S on G, and operations, LIST1 and DELETE, for the set T on G (we must treat them separately). In the course of the algorithm, $V_G - S$ is a set of vertices colored red, and,

```
procedure INDEPENDENT_SET;
  procedure GREEN (v);
    begin
      V_v := LIST_DEL (v, S); U := ∅;
      for each w ∈ V_v do
        begin
          if color(w) = green then
            begin
              report "There is not an independent set of size n − |X|"; halt
            end;
          u := mate(w); color(w) := red; color(u) := green; U := U ∪ {u};
          if w ∈ T then DELETE (w, T); if u ∈ T then DELETE (u, T)
        end;
      for each u ∈ U do GREEN (u)
    end;
  procedure BLUE (v);
    begin
      u := LIST1 (v, T);
      if u ≠ nil then
        begin
          DELETE (u, T); u' := mate(u);
          if color(u) = blue then
            begin
              color(u') := green; color(u) := red;
              DELETE (u, S); GREEN (u')
            end
          else begin color(u) := orange; color(u') := blue; BLUE (u') end;
          if color(v) = blue then BLUE (v)
        end
    end;
begin
  S := V_G; T := V_G; for each v ∈ V_G do color(v) := white;
  for each unmatched vertex v do begin color(v) := green; DELETE (v, T) end;
  for each unmatched vertex v do GREEN (v);
  while there is a vertex colored white in T do
    begin
      let v be a vertex colored white in T; v' := mate(v); DELETE (v', T);
      color(v') := orange; color(v) := blue; BLUE (v)
    end;
  report "There is an independent set of size n − |X|"
end;
```

FIG. 6.1. *The procedure* INDEPENDENT_SET.

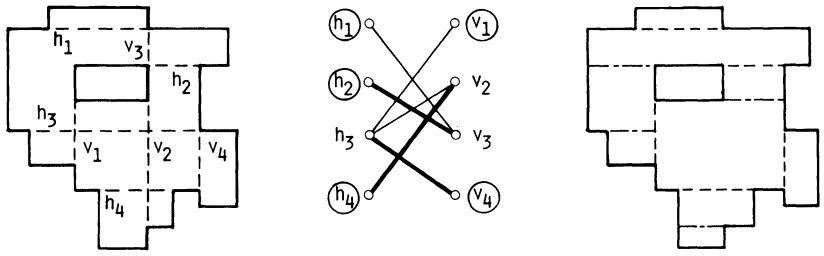
for each $u \in V_G - S$, vertex $mate(u)$ is colored green. T is a set of vertices colored blue or white. If the algorithm reports "There is an independent set of size $n-|X|$," then $\{v|color(v) = \text{green or blue}\}$ is an independent set of size $n-|X|$ at the end of the algorithm.

Next, consider the complexity of the procedure. Since BLUE essentially executes the depth first search, LIST1 and DELETE for T and DELETE for S in BLUE are executed $O(n)$ times. Hence, we can execute all the BLUE's except GREEN's in them in $O(g_T(n, m))$ time and $O(g_S(n, m))$ space. GREEN is executed at most once for each $v \in V_G$. Hence, the total complexity to execute all GREEN's is the complexity to execute a sequence of $O(n)$ LIST_DEL for S and DELETE for T. The other parts of the algorithm obviously take $O(n)$ time and space. Thus, we obtain the following.

PROPOSITION 6.1. *Let G be a graph with n vertices and m edges, to which a matching of size k is given. Then, we can determine whether or not there is an independent set of size $n-k$ in $O(g_T(n, m))$ time and $O(g_S(n, m))$ space.*

7. Applications.

7.1. Minimum dissection of rectilinear region. In manipulation of VLSI artwork data, there arises the problem of dissecting a rectilinear region into a minimum number of nonoverlapping rectangles (Lipski et al. [16], Ohtsuki [20]). Here, rectilinear regions are polygonal regions bounded only by horizontal and vertical edges which may have "holes." In Fig. 7.1, an example is given.



(a) *Rectilinear region and effective chords.*

(b) *Intersection graph of effective chords.*

(c) *A minimum dissection.*

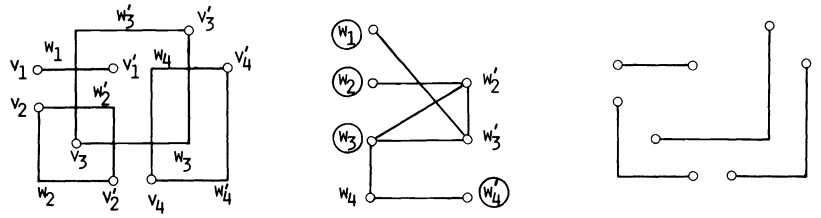
FIG. 7.1. *An example.*

This problem can be solved as follows. Let R be a rectilinear region with l edges. A chord of R connecting two concave vertices whose x- or y-coordinates are the same is called an *effective chord.* Let H (resp. V) be the set of horizontal (resp. vertical) effective chords, where $n=|H|+|V|$ and there are m pairs of intersecting effective chords. An intersection graph $G_S(H, V)$ of those chords is g-bipartite. Let S be a maximum independent set of $G_S(H, V)$. We first dissect the region by chords in S into subregions (note that each subregion has no effective chords). We then dissect each subregion by horizontal chords connecting concave vertices and points on edges. The above algorithm yields a minimum dissection [16], [20].

We now evaluate the complexity of the above algorithm. Given a rectilinear region, we can easily find all effective chords in $O(l \log l+n)$ time and $O(l)$ space by the plane-sweep algorithm. A maximum independent set of $G_S(H, V)$ can be found in $O(\sqrt{n}\, N)$ time and $O(N)$ space by Theorem 6.1 ($N=\min\{m, n \log n\}$). Finally, all the subregions can be dissected by such horizontal chords totally in $O(l \log l)$ time and $O(l)$ space by the plane-sweep algorithm. Thus, a minimum dissection of the rectilinear region can be found in $O(\sqrt{n}\, N+l \log l)$ time and $O(l+N)$ space. This improves the previous time bound $O(n^{5/2}+l \log l)$ in [20], and $O(n^{3/2} \log n \log \log n+l \log l)$ in [15], [16].

7.2. Manhattan wiring problem. We have the following problem in the design automation of digital systems (Masuda et al. [17], Raghavan et al. [21]). On the grid, n pairs of points are given. The Manhattan wiring problem is to connect all the pairs of points by wires along grid lines so that the wires do not intersect one another and no wire has more than one bend. Such a wiring is called a Manhattan wiring [17], [21]. Then the problem we consider is to determine whether there is a Manhattan wiring for given n pairs of points.

There are two rectilinear segments connecting the pair of points and bent at most once (in the case two points have the same x- or y-coordinates, there is the only one), each of which is called an *M-wire* of the pair of points. Then, there is a Manhattan wiring iff there is an independent set of size n in an intersection graph G_M of all the M-wires. In Fig. 7.2, an example is given.



(a) *Four pairs of points and their M-wires.*

(b) *Intersection graph of M-wires.*

(c) *A Manhattan wiring.*

FIG. 7.2. *An example.*

Let n_1 be the number of pairs of points which have the only one M-wire, and $n_2 = n - n_1$. The number of M-wires is $n_1 + 2n_2$. Since G_M has a trivial matching of size n_2, the problem of determining the existence of an independent set of size $(n_1 + 2n_2) - n_2 = n$ can be solved by the algorithm in § 6.2. Apparently, LIST_DEL, LIST1 and DELETE for the graph G_M can be implemented by those for an intersection graph of horizontal and vertical segments of which M-wires are composed. Let m be the number of edges in G_M, and $N = \min\{m, n \log n\}$. Then, by Proposition 6.1, we can solve the problem in $O(n \log n)$ time and $O(N)$ space. Our result is an improvement compared with an $O(n^3)$-time algorithm in [21] and an $O(n \log n + m)$-time algorithm in [17].

8. Concluding remarks. On the intersection problem of horizontal and vertical segments, and rectangles with sides parallel to the axes, efficient data structures have been developed. For the static problem of reporting all the intersections of a family of given n horizontal and vertical segments with an arbitrary horizontal or vertical query segment, an optimal algorithm due to Chazelle [4] is known. His algorithm takes $O(n \log n)$ preprocessing time, $O(n)$ space and $O(\log n + k)$ query time, where k is the number of reported intersections. However, for our purpose, the operation to delete a segment from the family is necessary, so that such a static algorithm does not work. McCreight's priority search tree [19] can be applied to this problem. His data structure is a dynamic one, and takes $O(n)$ space, $O((\log n)^2 + k)$ query time and $O((\log n)^2)$ update (deletion or insertion) time. By his data structure, a sequence of $O(n)$ LIST1's and DELETE's for $G_S(H, V)$ can be executed in $O(n(\log n)^2)$ time and $O(n)$ space. In this paper, by utilizing the special structures of our problems, we have given the data structure which decreases a factor $\log n$ in respect to the time complexity with increasing the same factor in respect to the space complexity.

It is straightforward to generalize the arguments in the paper for intersection graphs of horizontal and vertical segments to the problems for intersection graphs of n rectangles with sides parallel to the axes (and further for graphs of arbitrary boxicity k, where the *boxicity* of a graph G is the smallest k such that G is the intersection graph of hyperrectangles with sides parallel to the axes in k-dimensional space [22]). By employing the general data structure developed by Edelsbrunner [7], a sequence of $O(n)$ LIST_DEL's, LIST1's and DELETE's for the graph of rectangles can be executed in $O(n(\log n)^2)$ time and $O(n \log n)$ space. (It may be possible to execute the sequence in $O(n \log n)$ time and space by taking advantage of special properties of the sequence.) Hence, we can also solve the simple graph search problems for the graph of those rectangles in an efficient manner. For instance, the biconnected components of an intersection graph of n rectangles with sides parallel to the axes can be found in $O(n (\log n)^2)$ time and $O(n \log n)$ space (again, by counting the number of

pairs of intersecting rectangles in advance, and enumerating those intersecting rectangles if necessary, the above complexities can be modified, where the algorithms in Edelsbrunner [6], McCreight [18] and Six and Wood [23] may be used).

In [15], Lipski independently developed an algorithm for the problem of finding a maximum matching of g-bipartite $G_S(H, V)$ with n vertices, and claimed that the time and space complexities are $O(n^{3/2} \log n \log \log n)$ and $O(n \log n)$, respectively. His algorithm is rather complicated and a little worse with respect to the time complexity than our maximum matching algorithm. Furthermore, it seems that the data structures in [15] only would not be enough to execute the bipartite-matching algorithm in [13] correctly.

Finally, we remark that it is a challenging problem to develop an algorithm for executing a sequence of $O(n)$ basic operations, LIST1 and DELETE, for the intersection graph of horizontal and vertical segments in $O(n \log n)$ time, using $O(n)$ space.

Acknowledgments. The authors wish to thank Professor Masao Iri of the University of Tokyo and Professor David Avis of McGill University for their valuable discussions and comments on the paper.

REFERENCES

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

[2] J. L. BENTLEY, *Solutions to Klee's rectangle problems*, unpublished note, Dept. Computer Science, Carnegie-Mellon Univ., Pittsburgh, 1977.

[3] J. L. BENTLEY AND T. A. OTTMANN, *Algorithms for reporting and counting geometric intersections*, IEEE Trans. Comput., C-28 (1979), pp. 643-647.

[4] B. CHAZELLE, *Filtering search: A new approach to query-answering*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 122-132.

[5] E. A. DINIC, *Algorithm for solution of a problem of maximum flow in a network with power estimation*, Soviet Math. Dokl., 11, 5 (1970), pp. 1277-1280.

[6] H. EDELSBRUNNER, *A time- and space-optimal solution for the planar all intersecting rectangles problem*, Report 50, Institut für Informationsverarbeitung, Technische Universität Graz, 1980.

[7] ———, *Dynamic data structures for orthogonal intersection queries*, Report 59, Institut für Informationsverarbeitung, Technische Universität Graz, 1980.

[8] H. EDELSBRUNNER, J. VAN LEEUWEN, TH. OTTMANN AND D. WOOD, *Connected components of orthogonal geometric objects*, Report 72, Institut für Informationsverarbeitung, Technische Universität Graz, 1981.

[9] S. EVEN AND R. E. TARJAN, *Network flow and testing graph connectivity*, this Journal, 4 (1975), pp. 507-518.

[10] L. R. FORD, JR. AND D. R. FULKERSON, *Flows in Networks*, Princeton Univ. Press, Princeton, NJ, 1962.

[11] H. N. GABOW AND R. E. TARJAN, *A linear-time algorithm for a special case of disjoint set union*, Proc. 15th Annual ACM Symposium on Theory of Computing, Boston, 1983, pp. 246-251.

[12] Z. GALIL, *Finding the vertex connectivity of graphs*, this Journal, 9 (1980), pp. 197-199.

[13] J. E. HOPCROFT AND R. M. KARP, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, this Journal, 2 (1973), pp. 225-231.

[14] H. IMAI AND T. ASANO, *Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane*, J. Algorithms, 4 (1983), pp. 310-323.

[15] W. LIPSKI, JR., *Finding a Manhattan path and related problems*, Networks, 13 (1983), pp. 399-409.

[16] W. LIPSKI, JR., E. LODR, F. LUCCIO, C. MUGNAI AND L. PAGLI, *On two dimensional data organization* II, Fund. Inform., 2 (1979), pp. 245-260.

[17] S. MASUDA, S. KIMURA, T. KASHIWABARA AND T. FUJISAWA, *On the Manhattan wiring problem*, Papers of the Technical Group on Circuits and Systems, CAS 83-20, Institute of Electronics and Communication Engineers of Japan, 1983. (In Japanese.)

[18] E. M. MCCREIGHT, *Efficient algorithms for enumerating intersecting intervals and rectangles*, CSL-80-9, Xerox Palo Alto Research Center, Palo Alto, CA, 1980.

[19] ———, *Priority search trees*, CSL-81-5, Xerox Palo Alto Research Center, 1982; this Journal, 14 (1985), pp. 257-276.

[20] T. OHTSUKI, *Minimum dissection of rectilinear regions*, Proc. 1982 IEEE International Symposium on Circuits and Systems, Rome, 1982, pp. 1210–1213.

[21] R. RAGHAVAN, J. COHOON AND S. SAHNI, *Manhattan and rectilinear wiring*, Technical Report 81-5, Computer Science Dept., Univ. Minnesota, Minneapolis, 1981.

[22] F. ROBERTS, *Graph Theory and Its Applications to Problems of Society*, CBMS Regional Conference Series in Applied Mathematics, 29, Society for Industrial and Applied Mathematics, Philadelphia, 1978.

[23] H. W. SIX AND D. WOOD, *Counting and reporting intersections of d-ranges*, IEEE Trans. Comput., C-31, 3 (1982), pp. 181–187.

[24] R. E. TARJAN, *Depth-first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146–160.

[25] V. K. VAISHNAVI AND D. WOOD, *Rectilinear line segment intersection, layered segment trees, and dynamization*, J. Algorithms, 3 (1982), pp. 160–176.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
012

PLANAR MULTICOMMODITY FLOWS, MAXIMUM MATCHINGS AND NEGATIVE CYCLES*

KAZUHIKO MATSUMOTO†‡, TAKAO NISHIZEKI† AND NOBUJI SAITO‡

Abstract. This paper shows that the multicommodity flow problem on a class of planar undirected graphs can be reduced to another famous combinatorial problem, the weighted matching problem. Assume that in a given planar graph G all the sources can be joined to the corresponding sinks without destroying the planarity. Then we show that the feasibility of multicommodity flows can be tested simply by solving, once, the weighted matching problem on a certain graph constructed from G, and that the multicommodity flows of given demands can be found by solving the matching problem $O(n)$ times if G has n vertices. Efficient algorithms are also given for detecting negative and minimum cycles in planar undirected graphs.

Key words. algorithm, cut condition, feasibility, max flow-min cut theorem, minimum cycle, maximum matching, multicommodity flows, negative cycle, network, planar graph, planar separator theorem

1. Introduction. The network flow problem and its variants have been extensively studied. The most basic theorem of flow theory is the max flow-min cut theorem of Ford and Fulkerson which holds for single commodity and two-commodity flows [6], [12]. There are efficient algorithms for finding a maximum single-commodity flow; the $O(|E||V| \log |V|)$ time algorithm of Sleater and Tarjan is the theoretically best known one for sparse graphs, where V is the set of vertices and E the set of edges in a graph [21].

In the multicommodity flow problem, one would like to (1) test the feasibility, that is, decide whether a given graph G has multicommodity flows, each from a source to a sink and of a specified demand, and (2) then actually find them if G has. In contrast with the case of the single- or two-commodity flow problem, no efficient algorithm is currently known for the multicommodity flow problem on general graphs, although it can be applied to many practical problems such as the control of communication or traffic networks and the routing of VLSI. In fact the two-commodity *integral* flow problem is indeed NP-hard [5]. Several algorithms have been reported only for restricted classes of (planar) graphs [3], [17], [19], [22]. For example we have shown that if all sources and sinks lie on the boundary of the outer face of a given undirected planar graph then the multicommodity flows can be found by applying, $O(|V|)$ times, a shortest path algorithm to the dual of a given planar graph [17].

This paper deals with the multicommodity flow problem on another class of planar undirected graphs. Join all the sources of a given planar graph G to the corresponding sinks. Denote the resulting graph by G_a. We deal with the class of planar graphs G having planar G_a. We show that the multicommodity flow problem on such planar graphs can be reduced to a famous combinatorial problem, the maximum weighted matching problem, which is known to be polynomial-time solvable but looks apparently unrelated to our problem. More precisely, we reduce testing the feasibility of multicommodity flows in G to detecting a negative cycle in the dual G_a^* of G_a, whereas we reduce finding the multicommodity flows in G to finding, $O(n)$ times, a minimum cycle passing through a specified vertex in G_a^* or its variants. We then reduce detecting the negative and minimum cycles in a graph to finding a maximum weight matching

* Received by the editors October 10, 1983, and in revised form July 10, 1984.

† Department of Electrical Communications, Faculty of Engineering, Tohoku University, Sendai 980, Japan.

‡ Current address: System Development Laboratory, Hitachi Ltd., Ohzenji, Asou-ku, Kawasaki 215, Japan.

in a certain graph constructed from the graph. Thus the feasibility can be tested by solving, once, the weighted matching problem, and the multicommodity flows of given demands can be found by solving the matching problem $O(n)$ times. Furthermore we give an $O(n^{3/2}\log n)$ algorithm which finds a maximum matching in the graphs appeared in our reductions and hence detects the negative and minimum cycles in any planar undirected graph. The algorithm is based on both Lipton and Tarjan's planar separator theorem [16] and Galil, Micali and Gabow's matching algorithm [9]. Consequently the feasibility can be tested in $O(n^{3/2}\log n)$ time and the multicommodity flows can be found in $O(n^{5/2}\log n)$ time. Throughout this paper n denotes the number of vertices in a graph.

If $k=1$ in our problem, that is, a planar graph G has exactly one pair of source and sink, both of which are on the same face-boundary, then the well-known algorithm of "top-most" augmenting path can find efficiently a single-commodity flow in G [11], [13]. However the method cannot be adapted directly to our problem. Recently Seymour has shown that the max flow-min cut theorem holds true for the case of multicommodity flows in the same planar graphs as ours [20]. Our algorithm for testing feasibility uses his result. However our algorithm for finding multicommodity flows is completely different from his proof although its correctness depends on his theorem.

This paper is organized as follows. In § 2 we give $O(n^{3/2}\log n)$ algorithms for detecting the negative and minimum cycles in planar graphs. In § 3 we give an algorithm for testing feasibility of multicommodity flows. In § 4 we first present algorithm MULTIFLOW for finding multicommodity flows and then verify the correctness. In § 5 we first give the details of the implementation and then analyze the complexity. Section 6 is a conclusion.

2. Negative and minimum cycles in planar graphs. In this paper we denote by $G=(V, E)$ a finite *undirected* simple graph with vertex set V and edge set E, and denote by n the number of vertices of G. Assume that a real-valued weight is assigned to each edge. A *cycle* means the so-called simple nontrivial cycle, in which there are at least two vertices and a same vertex or edge does not appear more than once. Thus a single vertex is not a cycle. A *circuit* means a union of edge-disjoint cycles. The *weight* of a subset of E (or a subgraph) is the sum of the edge weights. A cycle is said to be *negative* if the weight is negative. A cycle is said to be *minimum* if the weight is minimum. In this section we first show that a negative cycle in any planar graph G can be detected in $O(n^{3/2}\log n)$ or $O(kn\log n+k^3)$ time if G has k negative edges. We then show that a minimum cycle passing through a specified vertex can be detected in $O(n^{3/2}\log n)$ time if G has no negative cycle. For a *directed* graph one can easily detect a negative cycle by applying shortest path algorithms [14]. However it is not the case for undirected graphs, because a false "negative cycle" may be detected that passes through a single negative undirected edge twice, once in each direction [23].

2.1. Negative cycle. A *matching* $M\subset E$ of a graph G is a set of pairwise nonadjacent edges of G. If $|M|=n/2$, then M is called a *complete matching* of G. In this section we give two algorithms for detecting a negative cycle in G. The first one reduces detecting a negative cycle in G to finding a maximum weight matching in a certain graph rather directly constructed from G. Replace each vertex $v\in V$ of G by a "star" [24] depicted in Fig. 1. Let $G'=(V', E')$ be the resulting graph. (See Fig. 2.) Define the edge weights of G' as follows: the surrogate of each edge $e\in E$ has the same weight as e; and all new edges in stars have zero weights. Similar constructions have appeared in [7], [14], [24]. We have the following lemma.

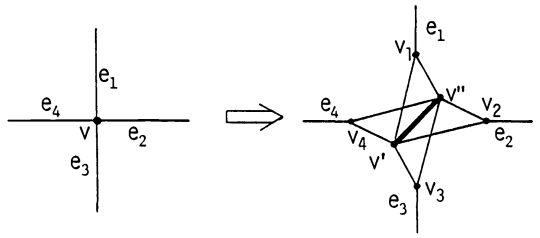
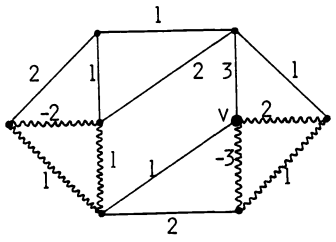


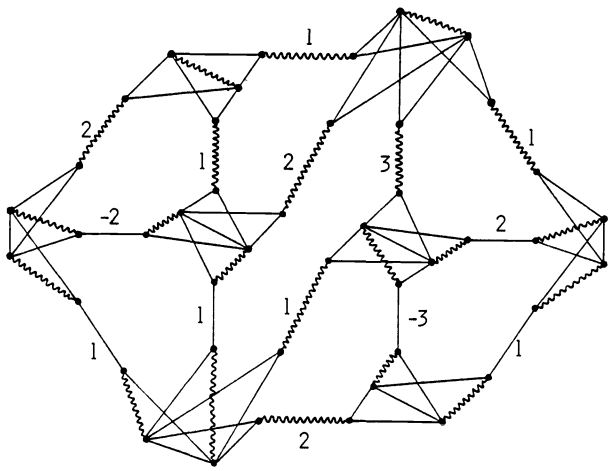
FIG. 1. *The star substituting a vertex of degree four.*

LEMMA 1. *Let $M \subset E'$ be a maximum complete matching of $G' = (V', E')$. Then $G = (V, E)$ has a negative cycle if and only if the weight of $E - M$ is negative.*

Proof. Let $M \subset E'$ be any complete matching of G'. If M contains edge (v', v'') in a star substituting a vertex v of G, then M must contain all the edges incident to v in G and hence v has degree 0 in the subgraph of G induced by $E - M$. (Edge (v', v'') is drawn by a thick line in Fig. 1.) On the other hand, if M does not contain (v', v''), then v has degree 2 in the subgraph. Thus $E - M$ is a vertex-disjoint union of cycles in G. Furthermore one can easily observe from the construction of G' that the converse is also true: if $C \in E$ is a vertex-disjoint union of cycles in G then there exists a complete matching M of G' such that $C = E - M$ (see Fig. 2). Clearly the weight of $E - M$ is minimum if and only if the weight of M is maximum. The claim follows immediately from these facts. Q.E.D.



(a) G



(b) G'

FIG. 2. *Graphs G and G'. (The edges in M of G' and the edges in E − M of G are drawn by wavy lines.)*

If G'' is a graph same as G' except for each weight being added the same large constant, then a maximum weight matching of G'' is a maximum weight complete matching of G' [14]. A maximum weight matching can be found in $O(|E||V|\log|V|)$ time for a general graph having $|V|$ vertices and $|E|$ edges by Galil, Micali and Gabow's algorithm [9]. (See [8] for the exposition of their algorithm and [2] for an alternative algorithm.) Since $|V'| = O(|E|)$ and $|E'| = O(|E|)$, a negative cycle in G can be detected in $O(|E|^2 \log n)$ time. (Using Gabow's recent result [7], this bound can be improved to $O(n|E|\log n)$.) If G is planar, then $|E| = O(n)$ by a corollary of Euler's formula [10]. Therefore a negative cycle can be detected in $O(n^2 \log n)$ time for a planar graph.

The bound $O(n^2 \log n)$ above can be further improved to $O(n^{3/2} \log n)$ as follows. Lipton and Tarjan gave the following planar separator theorem [16].

LEMMA 2 (planar separator theorem). *Let G be any planar graph with nonnegative vertex costs summing to no more than one. Then the vertices of G can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, neither A nor B has total vertex cost exceeding $2/3$, and C contains no more than $2(2n)^{1/2}$ vertices. Furthermore A, B, C can be found in $O(n)$ time.*

In this paper we say that a graph G has a *good separator* if there exist two positive constants $c_1 (<1)$ and c_2 satisfying:

> The vertices of G can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, neither A nor B contains more than $c_1 n$ vertices, and C contains no more than $c_2 n^{1/2}$ vertices.

In the special case of equal-cost vertices, Lemma 2 becomes.

LEMMA 3. *Any planar graph has a good separator with $c_1 = 2/3$ and $c_2 = 2(2)^{1/2}$.*

Lipton and Tarjan [15, pp. 624-625] gave an $O(n^{3/2} \log n)$ algorithm for finding a maximum weight matching in a planar graph, using a "divide and conquer" method based on Lemma 3. In our case G' is not always planar although G is planar. Therefore their algorithm cannot be directly applied to our case, but their techniques are adaptable to our problem. It may be assumed without loss of generality that every vertex of G has degree at least three. We first transform the planar graph G into a planar cubic graph G_3 with every vertex having degree three. A well-known transformation in graph theory [10, p. 132] may be used to generate G_3 from G. Consider a plane embedding of G. For each vertex v, where $w_0, \cdots, w_{d-1}$ is a cyclic ordering of the vertices adjacent to v in the plane embedding and $d (\geqq 3)$ is the degree of v, replace v with new vertices $v_0, \cdots, v_{d-1}$. Add edges $\{(v_i, v_{(i+1) \bmod d}): i = 0, \cdots, d-1\}$, each of weight 0, and replace the edges $\{(w_i, v): i = 0, \cdots, d-1\}$ with $\{(w_i, v_i): i = 0, \cdots, d-1\}$, of corresponding weights. Let G_3 be the resulting graph. (See Fig. 3(a).) From a corollary of Euler's formula [10], the number of vertices in G_3 will be less than $6n$. Clearly zero cycles newly appear in G_3. However one can observe the following lemma.

LEMMA 4. *G contains a negative cycle if and only if G_3 does.*

Thus a negative cycle in G can be detected by finding a maximum weight complete matching in the graph G_3', constructed from G_3 by replacing each vertex with a star. We furthermore have the following lemma.

LEMMA 5. *The class of graphs G_3' constructed from planar cubic graphs G_3 have good separators.*

Proof. Let n and n' denote the numbers of vertices in G_3 and G_3', respectively. By Lemma 3 the vertices of the planar graph G_3 can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, neither A nor B contains more than $(2/3)n$ vertices and C contains no more than $2(2n)^{1/2}$ vertices. Since each vertex of G_3 is replaced by a star having five vertices, G_3' has $n' = 5n$ vertices. The partition A, B, C naturally induces a partition of the vertices of G_3' into three sets A',

B', C', such that no edge joins a vertex in A' with a vertex in B', neither A' nor B' contains more than $(2/3)n'$ vertices, and C' contains $5 \cdot 2(2n)^{1/2} = 2(10n')^{1/2}$ vertices. Thus the class of graphs G_3' constructed from planar graphs G_3 have good separators. Q.E.D.

Similarly, we can show that the subgraphs of G_3' induced by A' and B', their subgraphs partitioned by their separators and so on, all have good separators. The following recursive algorithm finds a maximum matching in G_3'' which is the same as G_3' except for each weight being added the same large constant.

Step 1. If G_3'' contains a few vertices, say at most l vertices, find a maximum weight matching G_3'' by the algorithm in [2] or [9].

Step 2. Otherwise, apply Lemma 5 to G_3''. Let A, B, C, be the resulting vertex partition and let G_A, G_B be the subgraphs of G_3'' induced by the vertex sets A, B, respectively. Apply the algorithm recursively to find maximum weight matching M_A in G_A, M_B in G_B. Let $M = M_A \cup M_B$, $S = A \cup B$.

Step 3. Add C one vertex at a time to S. Each time a vertex is added to S, replace M by a maximum weight matching in G_S, the subgraph of G_3'' induced by the vertex set S. (This can be done in $O(n \log n)$ time per vertex in C [15, p. 625][1].) Stop when $S = A \cup B$.

Similarly as in [15] we can analyze the running time of the algorithm above. If $T(n)$ is the running time of the algorithm on graph G_3'' having n vertices, then

$$T(l) = a \quad \text{if } n \geqq l,$$

$$T(n) \leqq \max\{T(n_1) + T(n_2) + bn^{3/2} \log n : n_1 + n_2 \leqq n,\ n_1, n_2 \leqq cn\} \quad \text{if } n > l,$$

where a, b, $c(<1)$ and l are suitable positive constants, since $|C| = O(n^{1/2})$. An inductive proof shows that $T(n) = O(n^{3/2} \log n)$. Thus a negative cycle can be detected in $O(n^{3/2} \log n)$ time.

We now give the second algorithm for detecting a negative cycle, which is a modification of Tobin's algorithm for detecting a *nonpositive* cycle [23]. Denote by G^- the subgraph of G induced by the negative edges. Assume that G^- consists of J connected components. Denote by G_{ab} the graph same as G except for the weights being replaced by the absolute values. Let $K_0 = (V_0, E_0)$ be the complete graph on the vertices having odd degrees in G^-; the weight of edge (u, v) is the length of the shortest path between vertices u and v in G_{ab}. Let K_{0j}, $1 \leqq j \leqq J$, be the subgraph of K_0 induced by the vertices in the jth component of G^-. We have the following lemma.

LEMMA 6. *There exists no negative cycle in an undirected graph G if and only if*

(a) *G^- is a forest, that is, it has no cycle*;

(b) *For any two vertices u and v contained in the jth component of G^-, $1 \leqq j \leqq J$, there exists in G_{ab} a shortest path joining u and v all the edges of which have negative weights in G; and*

(c) *K_0 contains a minimum complete matching which is a union of (minimum) complete matchings of K_{0j}, $1 \leqq j \leqq J$.*

Proof. Omitted since one can verify the claim almost similarly as in [23]. Q.E.D.

Assume that G contains k negative edges, then G^- contains at most $2k$ vertices. Obviously, one can check condition (a) above in $O(|E|)$ time. Condition (b) is checked as follows: compare the two distances between u and v; one in G_{ab}; the other in the subgraph of G_{ab} induced by the edges which have negative weights in G. Thus, using

[1] The details of this claim are not found in H. Gabow's technical paper (*An efficient implementation of Edmonds' algorithm for maximum weight matching on graphs*, Univ. of Colorado, CU-CS-075-75, 1975) cited by [15]. However the claim is later proved partly in [9] and completely in [2].

the Dijkstra's algorithm [1], [4], one can check condition (b) for all u, v in $O(k|E|\log n)$ time. Clearly K_0 can be constructed in $O(k|E|\log n)$ time. A minimum complete matching can be found in $O(|V|^3)$ time in a general graph having $|V|$ vertices [14]. Therefore minimum complete matchings in K_0 and K_{0j}, $1 \leqq j \leqq J$, can be found in $O(k^3)$ time in total. Thus one can check condition (c) in $O(k^3)$ time. Hence the second algorithm can detect a negative cycle in a general graph G in $O(k|E|\log n + k^3)$ time. In the case of planar graphs this algorithm has an advantage if $k = o(n^{1/2})$.

We now have the following theorem.

THEOREM 1. *A negative cycle in a planar undirected graph G can be detected in $O(n^{3/2}\log n)$ or $O(kn\log n + k^3)$ time if G has k negative edges.*

2.2. Minimum cycle. Assume that graph G has no negative cycle and a vertex v is specified in G. In this section we show how to find a minimum cycle Z passing through v in G. Replace every vertex of G except v by a star, and replace v by a star not having edge (v', v''). Let G'_v be the resulting graph. Then one can easily obtain the following lemma.

LEMMA 7. *Assume that $M \subset E'$ is a complete matching of $G'_v = (V', E')$. Then M is maximum if and only if $E - M$ is a vertex-disjoint union of a minimum cycle Z passing through v and possibly some zero cycles in $G = (V, E)$.*

Proof. Since G'_v does not contain edge (v', v''), M does not contain (v', v''). Therefore vertex v has degree 2 and each of the others has degree 2 or 0 in the subgraph of G induced by $E - M$. That is, $E - M$ is a vertex-disjoint union of cycles of G including a cycle passing through v. The converse is also true. Now the claim is immediate. Q.E.D.

Thus, applying Galil, Micali and Gabow's matching algorithm to graph G'_v, one can find Z in $O(|E|^2\log n)$ time for a general graph G and in $O(n^2\log n)$ time for a planar graph G.

Using the planar separator theorem (Lemma 2), we can improve the bound $O(n^2\log n)$ above to $O(n^{3/2}\log n)$, similarly as the case of the negative cycle detection. However, the transformation of the case is not applicable to this case since a new zero cycle appears in place of v in the planar cubic graph G_3, which may be detected as a false minimum cycle passing through a surrogate of v. Therefore we replace each vertex of G except v by a zero cycle, but leave v as it is. Denote the resulting graph by G_{3v}. (See Fig. 3(b).) Clearly G_{3v} is planar, and all the vertices except v have degree three. Noting that G has no negative cycle, one can observe the following lemma.

LEMMA 8. *The weight of a minimum cycle in G passing through v is equal to that of G_{3v}.*

Replace each vertex of G_{3v} except v by a star and replace v by a star not having (v', v''). Let G'_{3v} be the resulting graph. Then we have the following lemma.

LEMMA 9. *The class of graphs G'_{3v}, constructed from planar graphs G'_{3v}, have good separators.*

Proof. Let n be the number of vertices of G'_{3v}, and let d be the degree of v. Then G'_{3v} have $n' = d + 2 + 5(n-1)$ vertices. Apply Lemma 2 to the planar graph G_{3v} with assigning cost $(d+2)/n'$ to vertex v and cost $5/n'$ to each of the other vertices. Then the vertices of G_{3v} can be partitioned into three sets A, B, C, such that no edge joins a vertex in A with a vertex in B, neither A nor B has cost more than $2/3$ and C contains no more than $2(2n)^{1/2}$ vertices. This partition naturally induces a partition of the vertices of G'_{3v} into three sets A', B', C', such that no edge joins a vertex in A' with a vertex in B', neither A' nor B' contains more than $(2/3)n'$ vertices. Thus it suffices to show that $|C'| = O(n'^{1/2})$.

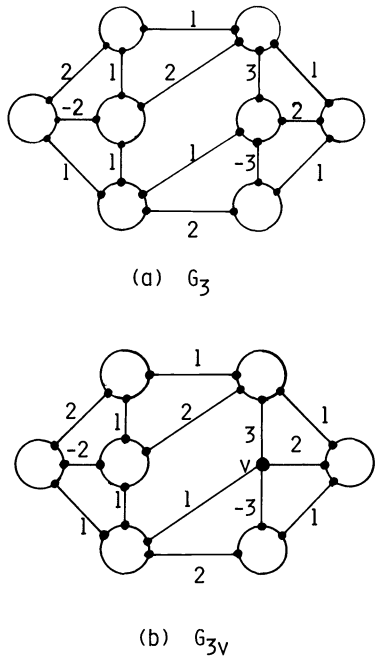


FIG. 3. (a) *Planar cubic graph* G_3 *and* (b) *planar graph* G_{3v}, *both constructed from planar graph* G *in Fig.* 2(a). (*The edges with no number have weight* 0.)

Consider first the case $v \notin C$. Then C' contains no more than $5 \cdot 2(2n)^{1/2} \leqq 2(10n')^{1/2}$ vertices. Thus A', B', C' is a desired partition of G'_{3v} in this case.

Consider next the case $v \in C$. Then $|C'| = d+2+5(|C|-1)$. Thus $|C'|$ is not necessarily $O(n'^{1/2})$ if d is large. However we can construct a desired partition A'', B'', C'' as follows. Assume that each edge e_l incident to v in G_{3v} is replaced by the surrogate edge (w_l, v_l) in G'_{3v}, where $l = 1, \cdots, d$ and assume that v_l is a vertex in the star of v. Let A'' be A' plus all the vertices v_l such that $w_l \in A'$, let B'' be B' plus all the vertices v_l such that $w_l \in B'$, and let C'' be the remaining vertices of G'_{3v}. Clearly no edge joins a vertex in A'' with a vertex in B''. Since $|A''| \leqq |A'| + d$, $|A'| \leqq (2/3)n'$ and $d \leqq n'/6$, we have $|A''| \leqq (5/6)n'$. Similarly we have $|B''| \leqq (5/6)n'$. Let $r(\leqq d)$ be the number of edges of G_{3v} joining v and vertices in C, then clearly $r \leqq |C|$. Therefore

$$|C''| = r+2+5(|C|-1) \leqq 6|C| = O(n'^{1/2}).$$

Thus we have shown that G'_{3v} has a good separator in this case, too. Q.E.D.

Similarly one can easily prove that the subgraphs of G'_{3v} induced by A' and B' (or A'' and B''), their subgraphs and so on, all have good separators. Using a "divide and conquer" method based on the separator above, we can find a maximum complete matching of G'_{3v} in $O(n^{3/2} \log n)$ time similarly as in § 2.1. Note that G'_{3v} has $O(n)$ vertices if the given planar graph G has n vertices. Thus we have:

THEOREM 2. *If there exists no negative cycle in a planar undirected graph* G, *then a minimum cycle passing through a specified vertex in* G *can be found in* $O(n^{3/2} \log n)$ *time.*

3. Feasibility of multicommodity flows. A (*flow*) *network* $N = (G, P, c)$ is a triplet, where:

(i) $G = (V, E)$ is a finite undirected simple graph.

(ii) P is the set of *source-sink pairs* (s_i, t_i), where source s_i and sink t_i are distinguished vertices in G.

(iii) $c: E \to R^+$ is the *capacity* function. (R (or R^+) denotes the set of (positive) real numbers.)

Each source-sink pair (s_i, t_i) of N is given a positive *demand* $d_i > 0$. Although G is undirected, we *orient* the edges of G arbitrarily so that the sign of a value of a flow function can indicate the real direction of the flow in an edge. A set of functions $\{f_1, f_2, \cdots, f_k\}$ with each $f_i: E \to R$ is *k-commodity flows* of demands $d_1, d_2, \cdots, d_k$ if it satisfies:

(a) the *capacity rule*: for each $e \in E$

$$\sum_{i=1}^{k} |f_i(e)| \leqq c(e),$$

where $|f_i(e)|$ denotes the absolute value of a real number $f_i(e)$;

(b) the *conservation rule*: each f_i satisfies

$$\mathrm{IN}(f_i, v) = \mathrm{OUT}(f_i, v) \quad \text{for each } v \in V - \{s_i, t_i\},$$

and

$$\mathrm{OUT}(f_i, s_i) - \mathrm{IN}(f_i, s_i) = \mathrm{IN}(f_i, t_i) - \mathrm{OUT}(f_i, t_i) = d_i,$$

where $\mathrm{IN}(f_i, v)$ is the total amount of flow f_i entering v, that is, $\mathrm{IN}(f_i, v) = \sum f_i(e) - \sum f_i(e')$, the first sum being over all the edges e oriented to enter v and with $f_i(e) > 0$, and the second over all the edges e' oriented to emanate from v and with $f_i(e') < 0$; and $\mathrm{OUT}(f_i, v)$ is the total amount of f_i emanating from v.

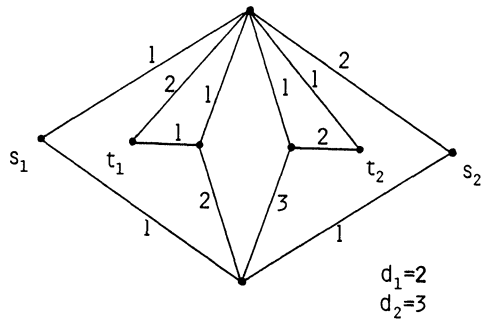
Denote by G_a the graph obtained from a given planar graph $G = (V, E)$ by adding a new edge $e_{ai} = (s_i, t_i)$ to G for each i, $1 \leqq i \leqq k$, as depicted in Fig. 4. We call e_{ai} a *demand* or *negative* edge. (This terminology will be justified below.) In this paper we assume that G is connected and G_a is planar, and hence $k = O(n)$.

Let $X \subset V$. Then $E(X)$ denotes the set of edges with one end in X and the other in $V - X$. $E(X)$ is called a *cut* of G. Especially $E(X)$ is called a *cutset* of G if $G - E(X)$ has exactly two connected components. We denote by $c(X)$ the sum of capacities of all the edges in $E(X)$ and call it a *capacity* of cut $E(X)$. We also denote by $d(X)$ the sum of the demands of all source-sink pairs with a source or sink in X and the other in $V - X$. We say that a network N satisfies the *cut condition* for the given demands if $c(X) \geqq d(X)$ for every $X \subset V$. It is known that N satisfies the cut condition if and only if $c(X) \geqq d(X)$ for every *cutset* $E(X)$ of G, where G is not necessarily planar [18]. Seymour [20] has shown that network N with planar G_a has multicommodity flows of given demands if and only if N satisfies the cut condition. Thus we shall show how to test the cut condition for a planar network.

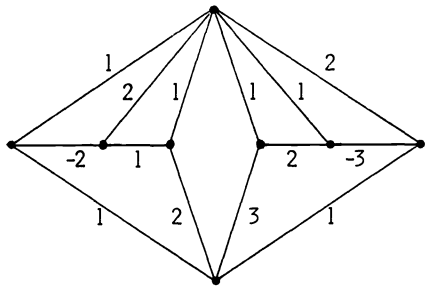
Define a new capacity function $c_a: E_a \to R$ for $G_a = (V, E_a)$ as follows:

$$c_a(e) = \begin{cases} c(e) & \text{if } e \in E, \\ -d_i & \text{if } e = e_{ai}. \end{cases}$$

Thus only e_{ai}, $1 \leqq i \leqq k$, have negative capacities in G_a. Also define $E_a(X)$ and $c_a(X)$ in a way similar to $E(X)$ and $c(X)$, respectively. Then clearly $c_a(X) = c(X) - d(X)$. Consider the dual G_a^* of the planar graph G_a. (Precisely G_a and G_a^* are not "graphs", but "multigraphs", that is, they may have multiple edges. However this does not affect the arguments in what follows.) The graph in Fig. 2(a) is indeed the dual of G_a in Fig. 4(b). We denote by e^* the edge of G_a^* corresponding to an edge e of G_a. We interpret c_a as a weight function of G_a^*. If $E(X)$ is a cutset of G, then $E_a(X)$ is a



(a) G



(b) G_a

FIG. 4. *Planar graphs* (a) *G and* (b) G_a.

cutset of G_a and hence corresponds to a cycle of G_a^*. Thus the Seymour's result [20] implies that network N has multicommodity flows of given demands if and only if G_a^* has no negative cycle. Hence the feasibility can be tested simply by checking the existence of negative cycles in G_a^*. Therefore by Theorem 1 we have:

THEOREM 3. *The feasibility of the k-commodity flows in a planar undirected graph can be tested in* $O(n^{3/2} \log n)$ *or* $O(kn \log n + k^3)$ *time.*

4. Algorithm MULTIFLOW. In this section we first give an algorithm MULTIFLOW which finds multicommodity flows in planar networks satisfying the cut condition. Then we verify the correctness.

4.1. Algorithm. Each demand edge e_{ai}, $1 \leqq i \leqq k$, adjoins two faces F_{i1} and F_{i2} of the planar graph G_a. Let Q_{i1} be the path joining s_i and t_i on the boundary of F_{i1} without passing through e_{ai}. Similarly define Q_{i2} with respect to F_{i2}. MULTIFLOW is outlined as follows. Choose an appropriate path Q_{ij} among $2k$ ones; push appropriate units D of flow f_i through Q_{ij}; reduce by D the demand d_i and the capacities of edges in Q_{ij}; and delete saturated edges if any. Repeat this operation until k-commodity flows of given demands are obtained.

Of course, D can exceed neither d_i nor $c(e)$ for every $e \in Q_{ij}$. However we wish to choose D as large as the resulting network does not violate the cut condition. (It is the crucial part of the algorithm.) Thus we decide D as follows:

$$D = \min \{-c_a(e_{ai}), \min \{c_a(e): e \in Q_{ij}\}, c_a(Q_{ij})/2\}. \tag{1}$$

Here $c_a(Q_{ij})$ is the minimum capacity of cutsets of G_a containing exactly two edges of Q_{ij}. That is, if $c_a(e, e')$ is defined for two edges e and e' of Q_{ij} as follows:

(2) $c_a(e, e') = \min\{c_a(X)$: $E_a(X)$ is a cutset of G_a and $E_a(X) \cap Q_{ij} = \{e, e'\}\}$,

then

$$c_a(Q_{ij}) = \min\{c_a(e, e'): e, e' \in Q_{ij}\}.$$

We define $c_a(Q_{ij}) = \infty$ if $|Q_{ij}| = 1$, i.e., Q_{ij} is a single edge joining s_i and t_i.

We are now ready to present MULTIFLOW.

```
procedure MULTIFLOW;
begin
  for each e ∈ E and i(1 ≦ i ≦ k) do f_i(e) := 0; {initialization}
  while G_a has a demand edge {there remains an unsatisfied flow} do
    begin
      choose path Q_ij with positive D;
        {1 ≦ i ≦ k. j = 1, 2. Q_ij contains no demand edges.}
      {push D units of flow f_i through Q_ij}
      for each e ∈ Q_ij do
        begin
          f_i(e) := f_i(e) ± D;
          {the sign ± depends on both the orientation of
          edge e and the direction of path Q_ij}
          c_a(e) := c_a(e) − D; {residual capacity}
          if c_a(e) = 0 {e is saturated}
            then G_a := G_a − e {delete e}
        end
      c_a(e_ai) := c_a(e_ai) + D; {decrease d_i = −c(e_ai) by D}
      if c_a(e_ai) = 0 {flow f_i has been satisfied}
        then G_a := G_a − e_ai {delete demand edge e_ai}
    end
end
```

4.2. Correctness. The following two lemmas must hold if algorithm MULTIFLOW above correctly finds multicommodity flows.

LEMMA 10. *If the original network $N = (G, P, c)$ satisfies the cut condition, then the new network $N' = (G', P', c')$ also satisfies the cut condition when D units of flow f_i have been pushed through Q_{ij}.*

Proof. We may assume that neither e_{ai} nor edges of path Q_{ij} are deleted and hence $G'_a = G_a$. (The proof for the other case is almost similar.) Let $c'_a(e)$ be the capacity of edge e in G'_a. Then $c'_a(e) = c_a(e) - D$ (>0) for every edge e on Q_{ij}, and $c'_a(e_{ai}) = c'_a(e_{ai}) + D$ (<0). Let $E_a(X)$ be any cutset of G'_a. Since $Q_{ij} \cup \{e_{ai}\}$ is a boundary of a face, we have

$$|E_a(X) \cap (Q_{ij} \cup \{e_{ai}\})| = 0 \text{ or } 2.$$

Therefore

$$c'_a(X) = \begin{cases} c_a(X) - 2D & \text{if } |E_a(X) \cap Q_{ij}| = 2, \\ c_a(X) & \text{otherwise.} \end{cases}$$

Since D satisfies (1), clearly $D \leqq c_a(X)/2$ if $|E_a(X) \cap Q_{ij}| = 2$. Thus we have $c'_a(X) \geqq 0$. Q.E.D.

LEMMA 11. *If network $N=(G, P, c)$ satisfies the cut condition, then G_a has at least one Q_{ij} with positive D among $2k$ paths.*

Proof. Assume to the contrary that there is no Q_{ij} with positive D, that is, any units of flow can be pushed through none of $2k$ paths. Then each Q_{ij} satisfies one of the following:

(a) Q_{ij} has a negative edge, that is, a demand edge;

(b) Q_{ij} has two "blocking" edges e and e' with $c_a(e, e')=0$.

Since N satisfies the cut condition, Seymour's theorem implies that N has a set of k-commodity flows satisfying given demands. For each Q_{ij} of $2k$ paths we define a route (path) R_i of flow f_i as follows. As shown in Fig. 5, let R_i be a path of G_a joining s_i and t_i such that positive units of flow f_i pass through R_i and the region bounded by $R_i \cup Q_{ij}$ contains a minimum number of faces in the interior. The assumption implies that the region contains at least one face.

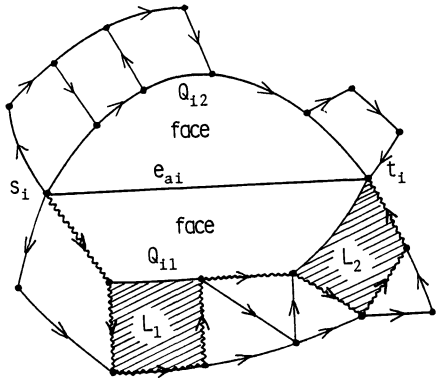


FIG. 5. *Illustration of route R_i of flow f_i. (R_i is drawn by wavy lines. An arrow of an edge indicates the direction of f_i through the edge. No flow f_i passes through edges having no arrows.)*

Among $2k$ paths and all possible sets of k-commodity flows, we now choose a path Q_{ij} and a set of k-commodity flows for which region $R_i \cup Q_{ij}$ contains a minimum number of faces. If region $R_i \cup Q_{ij}$ consists of nonempty connected components L_1, $L_2, \cdots, L_h$, then choose any one of them, say L_b, $1 \leqq b \leqq h$. (In the case of Fig. 5 $h=2$.) The planarity of G_a implies that if a source (or sink) lies in the proper interior of L_b then the corresponding sink (source) lies in the interior of L_b. Clearly Q_{ij} and the boundary of L_b share a common edge through which flow f_r other than f_i passes. Then source-sink pair (s_r, t_r) lie in L_b: otherwise, by interchanging routes of f_r and f_i as shown in Fig. 6, one can construct another set of k-commodity flows for which region

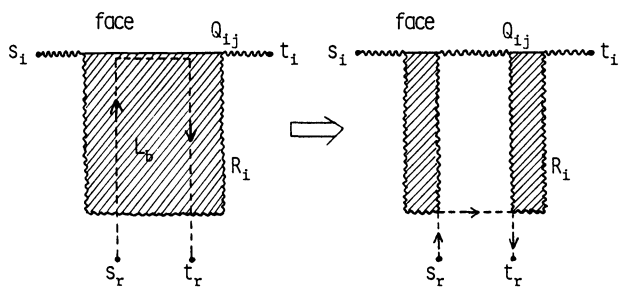


FIG. 6. *Alternation of a route of flow f_i. (A route of f_r is drawn by dashed lines.)*

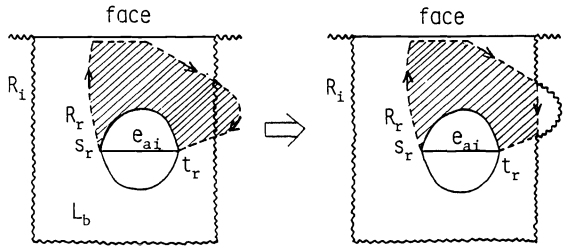


FIG. 7. *Alteration of route R_r of flow f_r.* (*R_r is drawn by dashed lines.*)

$R_i \cup Q_{ij}$ contains fewer faces, a contradiction. Furthermore we may assume that route R_r of flow f_r is contained in L_b: otherwise, i.e., if R_r intersects with R_i, then one can construct another set of k-commodity flows for which R_r is contained in L_b, as shown in Fig. 7. However, in this case region $R_r \cup Q_{rj'}$ ($j' = 1$ or 2) contains fewer faces than L_b, contrary to the assumption. Q.E.D.

These two lemmas imply that MULTIFLOW correctly finds multicommodity flows satisfying given demands. However, to this point, it is not clear that MULTIFLOW terminates in polynomial-time or even finitely.

5. Refinement and complexity. In this section we first refine MULTIFLOW and then analyze the complexity.

5.1. Refinement. Suppose that the part of MULTIFLOW choosing Q_{ij} with positive D is modified slightly as follows: choosing Q_{ij}, one by one, in cyclic order, check whether D of Q_{ij} is positive; and select the first one with positive D. Then Lemma 11 implies that we can find Q_{ij} with positive D among the first $2k$ paths. Furthermore it can be proved that the sign of D is checked $O(kn)$ times in total during one execution of MULTIFLOW, and hence MULTIFLOW runs in polynomial time.

However a sophisticated method can improve the bound $O(kn)$ above into $O(n)$. Consider each of the three terms in the right-hand side of (1). The first term $-c_a(e_{ai}) = d_i$ is always positive. The second term $\min\{c_a(e): e \in Q_{ij}\}$ may be negative. The second is positive if and only if path Q_{ij} contains no negative (i.e. demand) edge. Denote by u_{ij} the vertex of G_a^* corresponding to the face of G_a bounded by $Q_{ij} \cup e_{ai}$. Then the dual edge e_{ai}^* of G_a^* has two ends u_{i1} and u_{i2}. The negative edges of G_a^* always induce a forest F because G_a^* contains no negative cycle. (See Fig. 8.) A path Q_{ij} of G contains

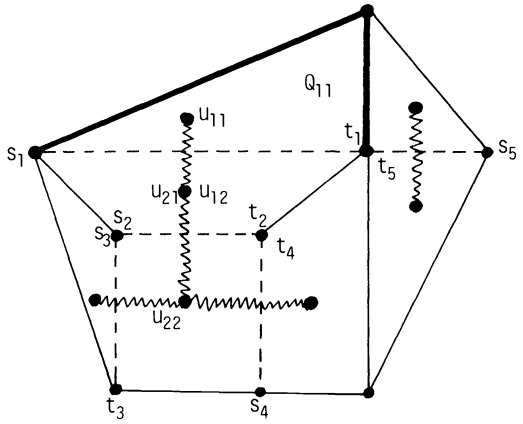


FIG. 8. *Planar graph G_a and the forest F induced by the negative edges in G_a^*.* (*The demand edges are drawn by dashed lines, F by wavy lines, and Q_{11} by thick lines.*)

no negative edge if and only if vertex u_{ij} is a *leaf* in F, that is, u_{ij} has degree one. Thus it suffices to evaluate the value D only for Q_{ij} such that u_{ij} is a leaf in F. Finally the third term $c_a(Q_{ij})/2$ of (1) is always nonnegative by Lemma 10. Each $c_a(X)$ in the right-hand side of (2) corresponds to a cycle in G_a^* passing through vertex u_{ij} but not edge e_{ai}^*, that is, a cycle in $G_a^* - e_{ai}^*$ passing through u_{ij}. (For a graph G and an edge e, $G-e$ denotes the graph obtained from G by deleting e.) Thus we have the following lemma.

LEMMA 12. *Assume that u_{ij} is a leaf in forest F. Then D of Q_{ij} is positive if and only if there exists, in $G_a^* - e_{ai}^*$, no zero cycle passing through u_{ij}.*

In Lemma 11 and its proof we implicitly assumed that G_a is connected. However the lemma can be generalized to the case in which G_a is not necessarily connected. It should be noted: if G_a is disconnected, then a face boundary is not necessarily a cycle but is a circuit (i.e. a union of edge-disjoint cycles), and hence Q_{ij} is not necessarily a path but is a union of a single path and cycles. One can prove the generalized result almost similarly as the proof of Lemma 11 except for choosing an innermost connected component having a negative edge. Combining the result and Lemma 12, we immediately have the following lemma.

LEMMA 13. *Forest F has a leaf u_{ij} such that D of Q_{ij} is positive, that is, all the cycles passing through u_{ij} are positive in $G_a^* - e_{ai}^*$.*

Consider now what changes occur in graphs G_a and G_a^* when $D(\geqq 0)$ units of flow f_i are pushed through path Q_{ij}. All the saturated edges of Q_{ij} are deleted in new G_a. The negative edge e_{ai} is also deleted if flow f_i is satisfied. The deletion of an edge in a primal graph corresponds to the "*contraction*" of the dual edge in the dual graph, in which the dual edge is deleted after its two ends are identified into a single vertex. Therefore all the saturated dual edges, which must be incident to u_{ij}, are contracted in new G_a^*, and the negative edge e_{ai}^* is also contracted in G_a^* if flow f_i is satisfied. Thus two or more trees in forest F may be merged into a single tree, and hence leaves may become "*interior vertices*" (i.e. vertices having degree two or more) in new F. To the contrast a tree is never split into two or more trees. Furthermore one can observe the following lemma.

LEMMA 14. *When D units of flow f_i are pushed through Q_{ij}, at most one interior vertex becomes a leaf in F. Furthermore this case occurs only if e_{ai}^* is contracted.*

For example see Fig. 8. At present u_{11} is a leaf and u_{21} is interior in F. If d_1 units of f_1 are pushed through Q_{11}, then only u_{21} can newly become a leaf in F.

Once it is known that $G_a^* - e_{ai}^*$ contains a zero cycle passing through leaf u_{ij} of F, it is not necessary to check the sign of D for Q_{ij} as long as u_{ij} continues to be a leaf in F. This is guaranteed by the following lemma.

LEMMA 15. *Let u_{ij} be a leaf in F, and assume that there exists a zero cycle Z passing through u_{ij} in $G_a^* - e_{ai}^*$. Then there remains such a zero cycle in $G_a^* - e_{ai}^*$ as long as u_{ij} continues to be a leaf in F.*

Proof. It is clear from the proof of Lemma 10 that for any $X \subset V$ $c_a(X)$ does not increase during the execution of MULTIFLOW. Therefore the weight of any cycle (or circuit) in G_a^* does not increase. Hence the weight of Z remains zero. However cycle Z may become a circuit.

Suppose first that Z is contracted into a circuit consisting of two or more cycles. Then all of them must be zero cycles since there exists no negative cycle in G_a^* and the weights of these cycles in Z total zero. Thus in this case there remains a zero cycle passing through u_{ij} in $G_a^* - e_{ai}^*$. Note that the zero cycle is possibly different from Z.

Suppose next that Z is contracted into a single vertex in G_a^*, that is, all the edges of the cut corresponding to Z are deleted in G_a. Consider the situation of G_a^* just

before the pushing flow is done that makes Z a single vertex. Then there must exist a zero cycle Z' passing through u_{ij} in $G_a^* - e_{ai}^*$. Since u_{ij} continues to be a leaf in F, Z' must contain at least three edges including exactly one negative edge e_{al}^* not incident to u_{ij}. Clearly, the pushing flow must be done for path Q_{l1} or Q_{l2}, say Q_{l1}. Then all the edges contracted by the pushing flow are incident to u_{l1} in G_a^*. However exactly two of them including e_{al}^* are contained in cycle Z'. Therefore at least one positive edge remains in Z', that is, Z' cannot become a single vertex after the pushing flow, contrary to the supposition. Hence this case does not occur.

Thus we have shown that there remains a zero cycle passing through u_{ij} in $G_a^* - e_{ai}^*$ as long as u_{ij} continues to be a leaf in F. Q.E.D.

If the existence of a zero cycle is known for a leaf u_{ij}, we will not check a zero cycle for the same u_{ij} until it becomes once interior and then a leaf again. Thus MULTIFLOW is refined as follows.

```
procedure MULTIFLOW;
begin
  for each e ∈ E and i(1 ≦ i ≦ k) do f_i(e) := 0;
  let u_ij be an arbitrary leaf of F;
  while G_a has a demand edge do
    if either D ≦ 0 or e_ai is deleted then
      let u_ij be another leaf of F which has never been
      chosen during the period it has continued to be a leaf
    else {D > 0. Push flow f_i through Q_ij}
      begin
        for each e ∈ Q_ij do
          begin
            f_i(e) := f_i(e) ± D;
            c_a(e) := c_a(e) − D;
            if c_a(e) = 0 then G_a := G_a − e
          end
        c_a(e_ai) := c_a(e_ai) + D;
        if c_a(e_ai) = 0 then G_a := G_a − e_ai
      end
end
```

Using Lemmas 13 and 15, one can easily verify the correctness of the refined MULTIFLOW. Furthermore we have the following lemma.

LEMMA 16. *The sign of D is checked for at most $O(n)$ paths in total during one execution of* MULTIFLOW, *that is, the while loop of* MULTIFLOW *is repated $O(n)$ times.*

Proof. When the sign of D is checked for Q_{ij} and subsequently $D(\geqq 0)$ units of flow f_i have been pushed through Q_{ij}, one of the following must occur:

(a) an edge of Q_{ij} is saturated and hence deleted;
(b) flow f_i is satisfied, and hence demand edge e_{ai} is deleted; or
(c) the value D of Q_{ij} is known zero or newly becomes zero. (In this case $G_a^* - e_{ai}^*$ has a zero cycle passing through u_{ij}.)

Clearly (a) occurs at most $3n$ times, and (b) at most k times. We claim that (c) occurs at most $3k$ times. If so, this lemma follows immediately since $k = O(n)$.

A vertex u_{ij} of F continues to be a leaf during several time periods; each of the periods consists of consecutive passes of the *while* loop in MULTIFLOW. Once MULTIFLOW chooses a leaf u_{ij}, MULTIFLOW continues to choose the same u_{ij} until

either (i) case (a) occurs and u_{ij} becomes interior, (ii) case (b) occurs, or (iii) case (c) occurs. If case (c) occurs, then MULTIFLOW will never choose the same u_{ij} during the current period. Thus (c) occurs at most once for the vertex u_{ij} in each of the periods. Initially F has at most $2k$ leaves. Interior vertices may become leaves, and leaves may become once interior and then leaves again. However the number of such vertices is at most k in total since the deletion of a negative edge can make at most one interior vertex a leaf by Lemma 14. Consequently the sum of the numbers of time periods, over all vertices of F, is at most $2k+k=3k$. Hence (c) occurs at most $3k$ times during one execution of MULTIFLOW. Q.E.D.

Remark. One can slightly strengthen the claim of Lemma 16 as follows. The number of times that (c) occurs but neither (a) nor (b) does is at most the number of leaves in initial F. Consequently the **while** loop is repeated at most that number plus the number of edges of G_a.

5.2. Time and space. By Lemma 16 the **while** loop in algorithm MULTIFLOW is repeated $O(n)$ times. The bookkeeping operations required for maintaining flow functions, capacities and leaves of F are all done in $O(n)$ time per pass of the loop. Thus the most time consuming operation in the loop is the computation of D. The problem is to compute $c_a(Q_{ij})$. Clearly $c_a(Q_{ij})$ is the weight of a minimum cycle Z in planar graph $G_a^* - e_{ai}^*$ passing through vertex u_{ij} if Q_{ij} contains at least two edges, and is infinite otherwise. Therefore by Theorem 2 the value D for Q_{ij} can be computed in $O(n^{3/2}\log n)$ time. Thus MULTIFLOW finds multicommodity flows of given demands in $O(n^{5/2}\log n)$ time.

All the graphs appearing in the algorithm are represented by adjacency lists, which clearly uses $O(|E|)$ space. Each flow function is represented by an array of length $|E|$. Therefore the k-commodity flows can be represented in $O(kn)$ space. Furthermore the algorithms for the maximum weighted matching in [2, 9] use $O(|E|)$ space. Thus MULTIFLOW uses $O(kn)$ space.

Thus we have the following theorem.

THEOREM 4. *If G_a is a planar graph of n vertices, then* MULTIFLOW *finds k-commodity flows satisfying given demands in $O(n^{5/2}\log n)$ time, using $O(kn)$ space.*

6. Conclusion. In this paper we first gave algorithms which find both a negative cycle and a minimum cycle passing through a specified vertex in a planar undirected graph in $O(n^{3/2}\log n)$. Then, using the algorithm we showed that the feasibility of multicommodity flows can be tested in the same time for any planar network having planar G_a. Finally we gave algorithm MULTIFLOW which finds multicommodity flows in $O(n^{5/2}\log n)$ time.

It is interesting that MULTIFLOW finds multicommodity flows whose values are half-integers if the capacities and demands are all integers. In particular, if the weights of all edges incident with each vertex of G_a total to an even number, then MULTIFLOW finds integral multicommodity flows.

Another interesting fact implied by Lemma 16 is that any multicommodity flows found by MULTIFLOW can be decomposed into $O(n)$ "path flows".

In the implementation of algorithm MULTIFLOW, although a maximum matching is known for a graph, we compute a maximum matching "from scratch" in a new graph for each time. Ball and Derigs [2] and Weber [25] have given efficient methods to update a maximum complete matching when a few edge weights are altered. We conjecture that the bound on MULTIFLOW can be improved, say to $O(n^2\log n)$, if one uses their updating methods and the techniques in [2], [9]. The detail is left to the ambitious reader.

Acknowledgments. We would like to thank Hiroshi Imai for suggesting the possibility of the improvement mentioned in the Conclusion and providing a relevant reference, and the referees for their comments which improved the presentation of the paper.

REFERENCES

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

[2] M. O. BALL AND U. DERIGS, *An analysis of alternative strategies for implementing matching algorithms*, Networks, 13 (1983), pp. 517-549.

[3] H. DIAZ AND G. DE GHELLINCK, *Multicommodity maximum flow in planar networks (The D-algorithm approach)*, CORE discussion paper No. 7212, Center for Operations Research and Econometrics, Louvain-la-Neuve, Belgium, 1972.

[4] E. W. DIJKSTRA, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269-271.

[5] S. EVEN, A. ITAI AND A. SHAMIR, *On the complexity of timetable and multicommodity flow problems*, this Journal, 5 (1976), pp. 691-703.

[6] L. R. FORD AND D. R. FULKERSON, *Maximum flow through a network*, Canad. J. Math., 8 (1956), pp. 399-404.

[7] H. N. GABOW, *An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems*, Proc. 15th Annual ACM Symposium on Theory of Computing, Boston, 1983, pp. 448-456.

[8] Z. GALIL, *Efficient algorithms for finding maximal matching in graphs*, Tech. Rept., Dept. Computer Science, Columbia Univ., New York, 1983.

[9] Z. GALIL, S. MICALI AND H. GABOW, *Priority queues with variable priority and an $O(EV \log V)$ algorithm for finding a maximal weighted matching in general graphs*, 23rd Annual Symposium on Foundations of Computer Science, Chicago, 1982, pp. 255-261.

[10] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1972.

[11] R. HASSIN, *Maximum flow in (s, t) planar networks*, Inform. Proc. Lett., 13 (1981), p. 107.

[12] T. C. HU, *Integer Programming and Network Flows*, Addison-Wesley, Reading, MA, 1969.

[13] A. ITAI AND Y. SHILOACH, *Maximum flows in planar networks*, this Journal, 8 (1979), pp. 135-150.

[14] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[15] R. J. LIPTON AND R. E. TARJAN, *Applications of a planar separator theorem*, this Journal, 9 (1980), pp. 615-627.

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

[17] K. MATSUMOTO, T. NISHIZEKI AND N. SAITO, *An efficient algorithm for finding multicommodity flows in planar networks*, this Journal, 14 (1985), pp. 289-301.

[18] H. OKAMURA AND P. D. SEYMOUR, *Multicommodity flows in planar graphs*, J. Combin. Theory B, 31 (1981), pp. 75-81.

[19] M. SAKAROVITCH, *The multicommodity flow problem*, Doctoral thesis, Operations Research Center, Univ. California, Berkeley, 1966.

[20] P. D. SEYMOUR, *On odd cuts and planar multicommodity flows*, Proc. London Math. Soc., (3) 42 (1981), pp. 178-192.

[21] D. D. SLEATOR AND R. E. TARJAN, *A data structure for dynamic trees*, J. Comput. System Sci., 26 (1983), pp. 362-390.

[22] D. T. TANG, *Bi-path networks and multicommodity flows*, IEEE Trans. Circuit Theory, CT-11 (1964), pp. 468-474.

[23] R. L. TOBIN, *Minimal complete matchings and negative cycles*, Networks, 5 (1975), pp. 371-387.

[24] W. T. TUTTE, *A short proof of the factor theorem for finite graphs*, Canad. J. Math., 6 (1954), pp. 347-352.

[25] G. M. WEBER, *Sensitivity analysis of optimal matchings*, Networks, 11 (1981), pp. 41-56.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
013

RELATIVIZATIONS OF UNAMBIGUOUS AND RANDOM POLYNOMIAL TIME CLASSES*

JOHN GESKE† AND JOACHIM GROLLMANN‡

Abstract. We study relationships between **P**, **NP**, and the unambiguous and random time classes, **U**, and **R**. Questions concerning these relationships are motivated by complexity issues in public-key cryptosystems. We prove that there exists a recursive oracle A such that $\mathbf{P}^A \neq \mathbf{U}^A \neq \mathbf{NP}^A$, and such that the first inequality is "strong," i.e., there exists a $\mathbf{P}^A$-immune set in $\mathbf{U}^A$. Further, we construct a recursive oracle B such that $\mathbf{U}^B$ contains an $\mathbf{R}^B$-immune set. As a corollary, we obtain $\mathbf{P}^B \neq \mathbf{R}^B \neq \mathbf{NP}^B$ and both inequalities are strong. By use of the techniques employed in the proof that $\mathbf{P}^A \neq \mathbf{U}^A \neq \mathbf{NP}^A$, we are also able to solve an open problem raised by Book, Long and Selman [*Quantitative relativizations of complexity classes*, SIAM J. Comput. 13 (1984), pp 461–487].

Key words. relativized complexity classes, oracles, random computations, unique solutions, immunity, polynomial time

1. Introduction. Valiant [10] introduced the notion of an *unambiguous* Turing machine—a nondeterministic Turing machine that has at most one accepting computation for any input. Let $\mathbf{U} \subseteq \mathbf{NP}$ be the collection of languages accepted by unambiguous Turing machines in polynomial time. $\mathbf{U}^X$ is the relativization of this class with respect to some oracle X. Rackoff [7] showed that there is a recursive oracle A such that $\mathbf{P}^A \neq \mathbf{U}^A = \mathbf{NP}^A$ and that there is a recursive oracle B such that $\mathbf{P}^B = \mathbf{U}^B \neq \mathbf{NP}^B$. A natural question that arises is: Does there exist an oracle C such that $\mathbf{P}^C \neq \mathbf{U}^C \neq \mathbf{NP}^C$? We answer this in the affirmative.

The proof of the result for **U** involves a combinatorial argument for which we have developed a pebbling game. This technique is of interest in itself, and a natural generalization of this game is used in solving an open problem of Book, Long and Selman [4].

The question of whether $\mathbf{U} = \mathbf{NP}$ is closely related to the question of whether there exist NP-hard public-key cryptosystems (PKCS). Even Selman and Yacobi [5] have shown that if promise problems associated with such systems do not exist then $\mathbf{U} \neq \mathbf{NP}$. Since the promise problem for a PKCS and sets in **U** are very similar, and any algorithm solving the cracking problem of the PKCS should need more than polynomial time for all sufficiently large codes, we, in addition, want $\mathbf{P}^X$-immune sets to exist in $\mathbf{U}^X$. Recall that for an arbitrary class of languages **C**, an infinite set A is **C**-*immune* if no infinite subset of A is in **C**. Given arbitrary classes **X** and **Y** such that $\mathbf{X} \neq \mathbf{Y}$, we say that the inequality is *strong* if there exists an **X**-immune set in **Y**.

The class $\mathbf{R} \subseteq \mathbf{NP}$, the common class of problems having efficient randomized algorithms, was defined by Adleman and Manders [1]. A set A belongs to **R** if and only if there exists a nondeterministic polynomial time-bounded Turing machine M such that $A = L(M)$ and for each $x \in A$, M accepts x with probability at least $\frac{1}{2}$. Rackoff showed, analogous to the results for **U**, that there is a recursive set D such that $\mathbf{P}^D \neq \mathbf{R}^D = \mathbf{NP}^D$, and there is a recursive set E such that $\mathbf{P}^E = \mathbf{R}^E \neq \mathbf{NP}^E$. Again the question arises: Is there a recursive oracle F such that $\mathbf{P}^F \neq \mathbf{R}^F \neq \mathbf{NP}^F$? Such an oracle

* Received by the editors November 17, 1983 and in revised form November 20, 1984. Funding for this research was provided by the National Science Foundation under grants MCS81-20263 and DCR84-02033.

† Computation Center, Iowa State University, Ames, Iowa 50011.

‡ Lehrstuhl Informatik I, Universität Dortmund, 4600 Dortmund 50, West Germany. This research was performed while this author visited the Computer Science Department, Iowa State University, Ames, Iowa 50011.

is provided by Sipser's construction [9] of a recursive set X such that $\mathbf{R}^X$ contains no complete set.

A secure cryptosystem should not be susceptible to cryptanalytic attack by efficient randomized algorithms. It should not even be "crackable" by an efficient randomized algorithm infinitely often. Hence, we would like to know whether a language in $\mathbf{U}$ exists which is $\mathbf{R}$-immune. In fact, we will show that both inequalities in $\mathbf{P}^F \neq \mathbf{R}^F \neq \mathbf{U}^F$ are strong for some oracle F. Therefore, for this oracle, $\mathbf{P}^F \neq \mathbf{R}^F \neq \mathbf{NP}^F$ and both inequalities are strong.

Rackoff's results and ours, taken together, indicate that it will be hard to prove $\mathbf{P} \neq \mathbf{U} \neq \mathbf{NP}$ and $\mathbf{P} \neq \mathbf{R} \neq \mathbf{NP}$. Intuitively, one believes $\mathbf{P} \neq \mathbf{U} \neq \mathbf{NP}$ and $\mathbf{P} \neq \mathbf{R} \neq \mathbf{NP}$. Since existence of an oracle X such that $\mathbf{P}^X \neq \mathbf{U}^X \neq \mathbf{NP}^X$ is a necessary condition for $\mathbf{P} \neq \mathbf{U} \neq \mathbf{NP}$, the separation results obtained here support our intuition about the nonrelativized world. The same can be said for the $\mathbf{P} \neq \mathbf{R} \neq \mathbf{NP}$ question.

We conclude this short introduction with some remarks about the notation used. The models of computation that will be used will be the oracle Turing machines as described by Baker, Gill and Solovay [2]. We assume the tape alphabet of all machines is $\Sigma = \{0, 1\}$, and all languages are subsets of Σ^*. We will use the notation $\Sigma^{\leq n}$ (respectively, Σ^n) for the set of all strings in Σ^* of length at most n (respectively, exactly n). We fix enumerations $\{P_i\}_{i \in \mathbf{N}}(\{NP_i\}_{i \in \mathbf{N}})$ of polynomial time-bounded deterministic (nondeterministic) oracle Turing machines. We assume that $p_i(n) = n^i + i$ is a strict upper bound on the length of any computation by P_i (or NP_i) with oracle X. P_i^X and NP_i^X denote query machines using oracle X.

In an inductive construction of a set X, $X(i)$ denotes the finite set of strings placed into X prior to stage i. Let $\langle \cdot, \cdot \rangle$ denote a fixed polynomial time computable pairing function with polynomial time computable inverses.

2. Main results. Let $\mathbf{U} \subseteq \mathbf{NP}$ be the collection of languages accepted by unambiguous Turing machines in polynomial time. A characterization of $\mathbf{U}$ is that L belongs to $\mathbf{U}$ if and only if there are some polynomial time computable predicate $P(x, y)$ and constant k such that $L = \{x$: there exists a y such that $|y| = |x|^k$ and $P(x, y)\} = \{x$: there is a unique y such that $|y| = |x|^k$ and $P(x, y)\}$. $\mathbf{R} \subseteq \mathbf{NP}$ is the collection of languages L such that for some polynomial time computable predicate $P(x, y)$ and a constant k, $L = \{x$: there exists a y such that $|y| = |x|^k$ and $P(x, y)\} = \{x$: there exist at least $2^{|x|^k - 1}$ values of y such that $|y| = |x|^k$ and $P(x, y)\}$.

We say that a nondeterministic Turing machine M is *unambiguous on* $\Sigma^{\leq n}$ if on each accepted input of length n or less, M has a unique accepting path. Conversely, a nondeterministic Turing machine M is *ambiguous on* $\Sigma^{\leq n}$ if on some accepted input of length n or less, M has more than one accepting path.

Before we state and prove our theorems we must first introduce some terminology and state a combinatorial lemma. A *board* is an $m \times m$ matrix over $\{0, 1\}$. A *square* of the board is an element, b_{ij}, of the matrix, where $i, j \leq m$. There are m^2 squares for every $m \times m$ board.

We describe a very simple pebbling game. Given a pebble, we may place it on any square of the board that does not already have a pebble on it. We denote a pebble on a square by a "1". The board is *covered* if the following two conditions are met for all $i \leq m$ and all $j \leq m$:

(1) $b_{ii} = 1$.

(2) $b_{ij} = 0 \rightarrow b_{ji} = 1$.

The object of the game is to cover the board.

LEMMA 2.1. *At least* $\lceil m^2/2 \rceil$ *pebbles are necessary to cover an* $m \times m$ *board.*

The proof is trivial.

DEFINITION 2.1. For any oracle X,

$L_0(X) = \{0^n$: there exists a $y \in X$ such that $|y| = n$ and n is odd$\}$,

$L_1(X) = \{0^n$: there is a $y \in X$ such that $|y| = n$ and n is even$\}$.

THEOREM 2.1. *There is a recursive oracle* A *such that* $\mathbf{P}^A \neq \mathbf{U}^A \neq \mathbf{NP}^A$.

Proof. For every oracle X, $\mathbf{P}^X \subseteq \mathbf{U}^X \subseteq \mathbf{NP}^X$. Therefore, it is sufficient to construct an oracle A containing at most one string of length n, for each odd n, such that $L_0(A) \notin \mathbf{P}^A$ and $L_1(A) \notin \mathbf{U}^A$. (Note that $L_0(A) \in \mathbf{U}^A$ and $L_1(A) \in \mathbf{NP}^A$.) We build A in stages. Initially $m = 0$ and $A = \varnothing$.

Stage i. If i is odd we look at the least element j not already examined in the enumeration of polynomial time-bounded deterministic oracle Turing machines. Once a machine P_j is examined it will never be examined again. Machine P_j has polynomial time-bound $p_j(n)$. Pick an odd integer n, $n > m$, so large that $p_j(n) < 2^n$. Run P_j with oracle $A(i)$ on input $x_j = 0^n$. If $P_j^{A(i)}$ accepts 0^n, add nothing to $A(i)$ at this stage. Otherwise, if $P_j^{A(i)}$ rejects 0^n, add to A the least string of length n not queried during the computation of $P_j^{A(i)}$ on input 0^n. (We know such a string exists.) We have thus added a single string of odd length to A. Finally, set $m = 2^n$ and go to the next stage.

If i is even we look at the least element k not already examined in the enumeration of polynomial time-bounded nondeterministic oracle Turing machines. Again, once machine NP_k is examined it will never be examined again. Machine NP_k has polynomial time-bound $p_k(n)$. Pick an even integer n, $n > m$, so large that $p_k(n) < 2^{n-1}$. If $NP_k^{A(i)}$ is ambiguous on $\Sigma^{\leq n}$ we add nothing to $A(i)$ at this stage, set $m = 2^n$, and go to the next stage.

If $NP_k^{A(i)}$ is unambiguous on $\Sigma^{\leq n}$, run $NP_k^{A(i)}$ on input $x_k = 0^n$. If $NP_k^{A(i)}$ accepts 0^n, add nothing to $A(i)$ at this stage. If $NP_k^{A(i)}$ rejects 0^n then add one or more strings of length n to $A(i)$ such that NP_k either still rejects 0^n, or accepts 0^n ambiguously. We will show that such strings can always be found. Now we set $m = 2^n$ and go to the next stage.

Claim. A nonempty subset $X \subseteq \Sigma^n$ exists such that $NP_k^{A(i) \cup X}$ either rejects 0^n or accepts 0^n ambiguously.

Proof of claim. We can, by exhaustive search, determine whether there exists a nonempty $X \subseteq \Sigma^n$ such that $NP_k^{A(i) \cup X}$ rejects 0^n. If we find such an X, then we are finished.

If such a subset cannot be found, then it must be the case that for all nonempty subsets X of Σ^n, $NP_k^{A(i) \cup X}$ accepts 0^n. In particular, $NP_k^{A(i)}$ does not accept 0^n, but for each string $x \in \Sigma^n$, $NP_k^{A(i) \cup \{x\}}$ does accept 0^n. An accepting path of $NP_k^{A(i) \cup \{x\}}$ on input 0^n is called a *critical path for* x. Note that every string x of length n has a critical path. If for some $x \in \Sigma^n$ there exists more than one critical path, then $NP_k^{A(i) \cup \{x\}}$ is ambiguous on 0^n. In this case take $X = \{x\}$ to settle our claim. Hence, we assume that there exists exactly one critical path for each string x. Therefore, there exist at most 2^n critical paths.

Denote cr (x) as the set of queries of length n made in the critical path for x. Clearly $x \in$ cr (x). Any change in an answer to a query in cr (x) may affect the resulting computation. If we place a string $y \in \Sigma^n$ into the oracle and $y \notin$ cr (x), then the addition of y to the oracle is oblivious on the critical path for x. Therefore, if we can find strings $x, y \in \Sigma^n$ such that $x \notin$ cr (y) and $y \notin$ cr (x), then by placing both x and y into $A(i)$ there will exist two distinct computation paths that accept 0^n. We show that such strings can be found by reducing this problem to the board covering game described earlier.

Fix an ordering, $x_1, x_2, x_3, \cdots$, of the strings of Σ^n. The success, or failure, in finding strings $x_i \notin \mathrm{cr}(x_j)$ and $x_j \notin \mathrm{cr}(x_i)$ is equivalent to determining whether a $2^n \times 2^n$ board can be covered with a given number of pebbles. The "pebbles" in this game are the queries of length n made in each critical path of a string in Σ^n, i.e., $b_{ij} = 1 \Leftrightarrow x_j \in \mathrm{cr}(x_i)$. If the board is covered, then for each $x_j \notin \mathrm{cr}(x_i)$ $(b_{ij} = 0)$ we have $x_i \in \mathrm{cr}(x_j)$ $(b_{ji} = 1)$. If the board is not covered, either for some i, $x_i \notin \mathrm{cr}(x_i)(b_{ii} = 0)$, which can never happen, or for some i and j, $x_j \notin \mathrm{cr}(x_i)$ and $x_i \notin \mathrm{cr}(x_j)$ $(b_{ij} = 0$ and $b_{ji} = 0)$. If this is the case, then we have found suitable strings.

Each critical path is of depth at most $p_k(n)$. Since there exist at most 2^n critical paths, there are at most $p_k(n)2^n$ pebbles, and since

$$p_k(n)2^n < 2^{n-1}2^n = 2^{2n-1},$$

it follows from Lemma 2.1 that we cannot cover the board. Therefore, there must be strings x_i and x_j such that $x_j \notin \mathrm{cr}(x_i)$ and $x_i \notin \mathrm{cr}(x_j)$, and this proves our claim.

To complete the proof of the theorem we need only to show $L_0(A) \notin \mathbf{P}^A$ and $L_1(A) \notin \mathbf{U}^A$. $L_0(A) \notin \mathbf{P}^A$ by the usual argument (cf. [2]). Suppose $L_1(A) \in \mathbf{U}^A$, then $L_1(A)$ is accepted by some unambiguous NP_i. At some stage k, and for some integer n, NP_i is run with oracle $A(k)$. By assumption $NP_i^{A(k)}$ is unambiguous on $\Sigma^{\leq n}$. But $0^n \in L_1(A)$ if and only if 0^n is rejected by $NP_i^{A(k)}$ or 0^n is accepted ambiguously. Since we assumed NP_i is unambiguous, we have a contradiction; $L_1(A) \notin \mathbf{U}^A$. □

THEOREM 2.2. *There exist a recursive oracle B and a language $L(B)$ such that $L(B)$ is $\mathbf{P}^B$-immune and $L(B) \in \mathbf{U}^B$.*

Proof. The construction is basically the same as the one given in the proof of the Immunity theorem in [8]. However, our theorem does not follow from the Immunity theorem, and so we give a straightforward proof here.

We build the oracle B in stages. $T(i)$ is a finite set of indices at stage i. Initially $T(i) = B(i) = \varnothing$ and $m = 0$.

Stage i. Let $T'(i+1) = T(i) \cup \{i\}$. Choose an integer $n > m$ such that $2^{n-1} \geqq \sum_{j \in T'(i+1)} (n^j + j)$. Check whether there exists an index $j \in T'(i+1)$ such that $0^n \in L(P_j^{B(i)})$. If an index exists, take the smallest such index j, define $T(i+1) = T'(i+1) - \{j\}$, $m = 2^n$, add nothing to B at this stage and go to the next stage. If such an index does not exist, choose a string of length n that is not queried by any of the machines $P_j^{B(i)}$, $j \in T'(i+1)$, on input 0^n. (We have chosen a sufficiently large n so that such a string does exist.) Add this string to B, define $T(i+1) = T'(i+1)$, $m = 2^n$ and go to the next stage. Note for each n, B contains at most one string of length n.

Let $L(B) = \{0^n$: there exists a $y \in B$ and $|y| = n\}$. Clearly $L(B) \in \mathbf{U}^B$. We have to show that $L(B)$ is $\mathbf{P}^B$-immune. First, $\|L(B)\| = \infty$. Suppose $L(B)$ were finite; then B would also be finite. Therefore, after some stage i_0, we must always have, in stage $i > i_0$, the case that $0^n \in L(P_j^B)$ for a given n and some $j \in T'(i+1)$. Therefore, j is removed from $T'(i+1)$. This means that, from stage i_0 on, all $T(i)$ have a constant length. Therefore, only a finite number of sets $L(P_j^B)$ (those whose indices are never removed from any $T(i)$) do not contain some element 0^n for some n. But, there are infinitely many j with $L(P_j^B) = \varnothing$, and so we have a contradiction, and $L(B)$ must be infinite.

Now we show that no infinite subset S of $L(B)$ is equal to some $L(P_j^B)$. Suppose $S = L(P_j^B)$, $S \subseteq L(B)$ and S infinite. If j is removed from $T'(i+1)$ at some stage i, then $0^n \in L(P_j^{B(i)})$ and $0^n \in L(P_j^B)$. Therefore, $0^n \in S$ and $0^n \in L(B)$. But if $0^n \in L(P_j^{B(i)})$ at stage i, we add no strings of length n to B, therefore, $0^n \notin L(B)$. This is a contradiction, so j must stay in $T(k)$ for all stages $k > j$. Therefore, for all but a finite number of 0^n chosen, $0^n \notin L(P_j^B)$. But in $L(B)$ we only have elements x, where $x = 0^n$ for some chosen

n, and by assumption, $L(P_j^B) \subseteq L(B)$. Therefore, $L(P_j^B)$ must be finite. This is also a contradiction. □

COROLLARY 2.1. *There exist a recursive oracle C and a language $L(C)$ such that $\mathbf{P}^C \neq \mathbf{U}^C \neq \mathbf{NP}^C$, $L(C)$ is $\mathbf{P}^C$-immune, and $L(C) \in \mathbf{U}^C$.*

Proof. For the proof we only need to replace the odd stages in the proof of Theorem 2.1 with the corresponding stages in the proof of Theorem 2.2, *mutatis mutandis.* □

Let $\{Pr_j\}_{j \in \mathbf{N}}$ be a recursive enumeration of polynomial time-bounded oracle Turing machines that compute predicates of two variables. We define for any oracle X, each $k \in \mathbf{N}$, and for each Pr_j^X, $j \in \mathbf{N}$, the following language:

$$L(Pr_j^X, k) = \{x: \text{there exists a } y \text{ such that } |y| = |x|^k \text{ and } Pr_j^X(x, y)\}.$$

(Pr_j^X, k) is *random* if and only if $L(Pr_j^X, k) = \{x$: there exist $2^{|x|^k - 1}$ distinct $y \in X$ such that $|y| = |x|^k$ and $Pr_j^X(x, y)\}$. Note that $L \in \mathbf{R}^X$ if and only if there exists a j and k such that (Pr_j^X, k) is random and $L = L(Pr_j^X, k)$. We assume that $(|x| + |y|)^j + j$ is the time bound of Pr_j on input (x, y). Thus, on input (x, y), where $|y| = |x|^k$, Pr_j runs for at most $|x|^{kj} + j$ steps.

THEOREM 2.3. *There are a recursive oracle D and a language $L(D)$ such that $L(D) \in \mathbf{U}^D$ and $L(D) \notin \mathbf{R}^D$.*

Proof. Let $L(X) = \{0^n$: there exists a $y \in X$ such that $|y| = n\}$. To show that $L(X) \notin \mathbf{R}^X$, it is sufficient to show:

$$\forall j,\ k \in \mathbf{N}(Pr_j^X, k) \text{ random} \rightarrow L(X) \neq L(Pr_j^X, k).$$

We build the oracle D in stages. Initially $D = \varnothing$ and $m = 0$.

Stage i. We look at $(Pr_j^{D(i)}, k)$ where $\langle j, k \rangle = i$. Choose an integer $n > m$ such that $2^{n-1} > n^{kj} + j$. Determine whether $(Pr_j^{D(i)}, k)$ is random on $\Sigma^{\leq n}$. If it is not random, set $m = 2^n$, add nothing to D at this stage and go to the next stage. If it is random, and $0^n \in L(Pr_j^{D(i)}, k)$, set $m = 2^n$, add nothing to D at this stage and go to the next stage.

If $0^n \notin L(Pr_j^{D(i)}, k)$, we must find a string x, $|x| = n$, such that 0^n is not accepted by $(Pr_j^{D(i) \cup \{x\}}, k)$ with probability $\geq \frac{1}{2}$. Such a string exists. Add x to the oracle, set $m = 2^n$ and go to the next stage.

Claim. If $0^n \notin L(Pr_j^{D(i)}, k)$, a string $x \in \Sigma^n$ exists such that 0^n is not accepted by $(Pr_j^{D(i) \cup \{x\}}, k)$ with probability $\geq \frac{1}{2}$.

Proof of claim. For each y, $|y| = n^k$, a (deterministic) path of $Pr_j^{D(i)}$ on $(0^n, y)$ leads to a reject state. There are 2^{n^k} such paths; along each path there may be oracle queries of the form "$x \in D$?," where $|x| = n$, that are answered "no." If adding any x to the oracle causes the machine to accept 0^n with probability $\geq \frac{1}{2}$, then there are $2^{n^k - 1}$ paths such that the machine queries the oracle about x on these paths; the corresponding changes in the responses causes the machine to accept. We say these are *critical queries* for x. For the 2^n strings of length n we have at least $2^n 2^{n^k - 1}$ critical queries. The length of any path is limited by $n^{kj} + j$. The total number of possible critical queries is $(n^{kj} + j)2^{n^k} < 2^n 2^{n^k - 1}$. A string x, $|x| = n$, such that 0^n is not accepted by $(Pr_j^{D(i) \cup \{x\}}, k)$ with probability $\geq \frac{1}{2}$ must exist.

From the construction it follows that $L(D) \notin \mathbf{R}^D$. $L(D) \in \mathbf{U}^D$ since for every length n there is at most one string of length n in D. □

COROLLARY 2.2. *There exists a recursive oracle E such that $\mathbf{P}^E \neq \mathbf{R}^E \neq \mathbf{U}^E$, $\mathbf{R}^E$ contains a $\mathbf{P}^E$-immune set and $\mathbf{U}^E$ contains an $\mathbf{R}^E$-immune set.*

Proof. Let $L_0(X)$ and $L_1(X)$ be as defined in Definition 2.1. We build E in stages. At the odd stages we "slowly diagonalize" $L_0(E)$ out of $\mathbf{P}^E$. That is, we proceed as in the proof of Theorem 2.2, but instead of adding one string of length n to the oracle,

add 2^{n-1} strings to the oracle. Hence, $L_0(E) \in \mathbf{R}^E$. In the even stages, we slow down the construction in the proof of Theorem 2.3 so that $L_1(E)$ is $\mathbf{R}^E$-immune. □

Balcázar and Russo [3] contain a variety of relativization results of probabilistic complexity classes, and some of these results overlap with Corollary 2.2. They independently prove the existence of a recursive oracle E_0 such that $\mathbf{R}^{E_0}$ contains a $\mathbf{P}^{E_0}$-immune set, and they prove the existence of a recursive oracle E_1 such that $\mathbf{NP}^{E_1} \cap \text{co-}\mathbf{R}^{E_1}$ contains an $\mathbf{R}^{E_1}$-immune set.

The following corollary shows that each of the classes **P**, **R**, **U** and **NP** can be made distinct. Its proof is a simple extension of Corollary 2.2. Rather than diagonalize in two stages, we diagonalize in three stages, and in the third stage diagonalize **NP** out of **U** via the technique used in Theorem 2.1.

COROLLARY 2.3. *There exists a recursive oracle F such that $\mathbf{P}^F \neq \mathbf{R}^F \neq \mathbf{U}^F \neq \mathbf{NP}^F$, $\mathbf{R}^F$ contains a $\mathbf{P}^E$-immune set and $\mathbf{U}^F$ contains an $\mathbf{R}^F$-immune set.*

We raise the following open questions:

(1) Is there an oracle X such that $\mathbf{NP}^X$ contains $\mathbf{U}^X$-immune sets?

(2) Are there an oracle X and a language $L(X)$ such that $L(X) \in \mathbf{R}^X$ and $L(X) \notin \mathbf{U}^X$?

Now we will apply our combinatorial technique to solve an open problem raised by Book, Long and Selman [4] which studies properties of restricted forms of relativizations of **NP**.

DEFINITION 2.2. Let M be an oracle machine. For any set X and any input string x of M, let $\text{QA}(M, X, x) = \{q$: in some accepting computation of M relative to X on input x, the oracle is queried about $q\}$.

DEFINITION 2.3. Let X be a set. $\mathbf{NP.ACC.DEP}^X$ is the class of languages L such that $L \in \mathbf{NP}^X$ is witnessed by a machine M such that for some polynomial q, and for all x, $\|\text{QA}(M, X, x)\| \leqq q(|x|)$.

Obviously for all sets X we have $\mathbf{U}^X \subseteq \mathbf{NP.ACC.DEP}^X \subseteq \mathbf{NP}^X$.

THEOREM 2.4. *There exists a recursive oracle F such that $\mathbf{P}^F \neq \mathbf{U}^F \neq \mathbf{NP.ACC.DEP}^F$.*

Theorem 2.4 follows from a simple modification of the proof of Theorem 2.1. Namely, we may ensure that in the even stages of the proof we always add at most two strings of any even length to the oracle. Hence $L_1(F) \in \mathbf{NP.ACC.DEP}^F$.

Book, Long and Selman raised and left open the question of whether there exists an oracle X such that $\mathbf{NP.ACC.DEP}^X \subsetneq \mathbf{NP}^X$. This is so, and to prove this fact we first describe a generalization of the pebbling game defined earlier.

A *board* is a c-dimensional m-element matrix (i.e. m entries in each dimension) over $\{0, 1\}$, where c and m are positive integers. A *square* of the board is an element, $b_{\langle i_1, \cdots, i_c \rangle}$, of the matrix, where $i_1, \cdots, i_c \leqq m$. A c-dimensional m-element board contains m^c squares. Let $I = \{\langle i_1, \cdots, i_c \rangle : i_1, \cdots, i_c \leqq m\}$ be the set of all ordered c-tuples. A c-tuple is denoted by $\mathbf{i}$. Let $i_j \in \mathbf{i}$ if and only if $\mathbf{i} = \langle i_1, \cdots, i_j, \cdots, i_c \rangle$, i.e., i_j is a component of the c-tuple $\mathbf{i}$. We say that $\mathbf{i}$ is *pairwise disjoint* if for all $i_j, i_k \in \mathbf{i}$, $j \neq k$ implies $i_j \neq i_k$.

The pebbling game is played as described earlier. Namely, given a pebble, denoted by a "1", we may place it on any square not already covered by a pebble. The object of the game is to cover the board with a given number of pebbles. The board is *covered* if the following conditions hold:

(1) $b_{\mathbf{i}} = 1$ for all $\mathbf{i}$ that are not pairwise disjoint.

(2) For all pairwise disjoint $\mathbf{i}$ there is a permutation π such that $b_{\pi(\mathbf{i})} = 1$.

LEMMA 2.2. $\binom{m}{c} + m^c - m!/(m-c)!$ *pebbles are needed to cover a c-dimensional m-element board.*

Proof. Let $\Delta = \{\mathbf{i}: \mathbf{i}$ is not pairwise disjoint$\}$. If the board is covered, then for each $\mathbf{i} \in \Delta$, $b_{\mathbf{i}} = 1$. Also, if the board is covered, then for each $\mathbf{i} \in I - \Delta$, $b_{\pi(\mathbf{i})} = 1$ for some

permutation π. There are $c!$ permutations for each $\mathbf{i}$, and $\|I-\Delta\| = m!/(m-c)!$. Therefore, the total number of pebbles needed to cover the board is

$$\frac{m!}{(m-c)!c!} + m^c - \frac{m!}{(m-c)!}$$

which proves our claim. □

THEOREM 2.5. *There exists a recursive oracle G such that* $\mathbf{NP.ACC.DEP}^G \neq \mathbf{NP}^G$.

Proof. Let $L(G) = \{0^n$: there exists a y, $y \in G$ and $|y| = n\}$. G is constructed in stages such that $L(NP_i^G) \neq L(G)$ for all $L(NP_i^G) \in \mathbf{NP.ACC.DEP}^G$. Initially $m = 0$ and $G = \varnothing$.

Stage $i = \langle j, k\rangle$. We examine $NP_j^{G(i)}$ (with polynomial time bound $p_j(n)$) and polynomial $p_k(n) = n^k + k$. Choose an integer n, $n > m$, so large that $p_k(n) < 2^{n/3} - 1$ and $p_j(n) < 2^n$. At this stage we will ensure that either $L(NP_j^G) \neq L(G)$ or $\|\mathrm{QA}(NP_j, G, 0^n)\|$ is not bounded by $p_k(n)$. Run $NP_j^{G(i)}$ on $\Sigma^{\leqq n}$. If $NP_j^{G(i)}$ accepts any string x, $|x| \leqq n$, and $\|\mathrm{QA}(NP_j, G(i), x)\| > p_k(|x|)$, then we add nothing to $G(i)$ at this stage, set $m = 2^n$, and go to the next stage.

If on each accepted string x, $|x| \leqq n$, $\|\mathrm{QA}(NP_j, G(i), x)\| \leqq p_k(|x|)$, then run $NP_j^{G(i)}$ on input 0^n. If $NP_j^{G(i)}$ accepts 0^n, then we add nothing to $G(i)$, set $m = 2^n$ and go the next stage.

If $NP_j^{G(i)}$ rejects 0^n, then we add $X \subseteq \Sigma^n$ to $G(i)$ such that either $NP_j^{G(i) \cup X}$ still rejects 0^n or $\|\mathrm{QA}(NP_j, G(i) \cup X, 0^n)\| > p_k(n)$. We will show that such strings can always be found. Now set $m = 2^n$ and go to the next stage.

Claim. A nonempty subset $X \subseteq \Sigma^n$ exists such that $NP_j^{G(i) \cup X}$ either rejects 0^n or accepts 0^n but $\|\mathrm{QA}(NP_j, G(i) \cup X, 0^n)\| > p_k(n)$.

Proof of claim. We can, by exhaustive search, determine whether there exists $X \subseteq \Sigma^n$ such that $NP_j^{G(i) \cup X}$ rejects 0^n. If we find such an X, then we are finished.

If such a subset cannot be found, then it must be the case that for all nonempty subsets X of Σ^n, $NP_j^{G(i) \cup X}$ accepts 0^n. In particular, $NP_k^{G(i)}$ does not accept 0^n, but for each string $x \in \Sigma^n$, $NP_k^{G(i) \cup \{x\}}$ does accept 0^n. $\mathrm{QA}(\mathrm{NP}_j, G(i) \cup \{x\}, 0^n)$ is the set of queries made on the accepting paths of NP_j with the oracle $G(i) \cup \{x\}$. In this context we denote this set $\mathrm{QA}(x)$. To every string x of length n there exists a nonempty set $\mathrm{QA}(x)$. Obviously, there exist no more than 2^n such sets. If $\|\mathrm{QA}(x)\| > p_k(n)$ for some $x \in \Sigma^n$, then take $X = \{x\}$, and the claim is proved. Hence, we can assume without loss of generality that $\|\mathrm{QA}(x)\| \leqq p_k(n)$ for each $x \in \Sigma^n$.

Clearly $x \in \mathrm{QA}(x)$. Any change in an answer to a query in $\mathrm{QA}(x)$ may affect the resulting computation. If $y \notin \mathrm{QA}(x)$ and y is placed in the oracle, then the addition of y to the oracle is oblivious on accepting computation paths of $NP_j^{G(i) \cup \{x\}}$. Therefore, if we can find a set of strings X in Σ^n such that $\|X\| = p_k(n) + 1$ and for all $x, y \in X$, $x \neq y$ implies $x \notin \mathrm{QA}(y)$, then $\|\mathrm{QA}(NP_j, G(i) \cup X, 0^n)\| > p_k(n)$. We show that such strings can be found by reducing this problem to the generalized board covering game.

Fix an ordering, $x_1, x_2, x_3, \cdots$, of the strings of Σ^n. Let us see that the success, or failure in finding a suitable X is equivalent to determining whether a $(p_k(n)+1)$-dimensional 2^n-element board can be covered with a given number of pebbles. We will pebble the board in the following way: We place a pebble on every $\mathbf{i} \in I$ which is not pairwise disjoint, because $x_r \in \mathrm{QA}(x_r)$ for each $x_r \in \Sigma^n$. Now let us assume that $x_r \in \mathrm{QA}(x_s)$, $r \neq s$. Then, for each maximal subset S of I whose elements contain r and s, are pairwise disjoint and identical up to permutation, we choose one of the elements of S, say $\mathbf{i}$, and set $b_{\mathbf{i}} = 1$.

If the board is covered, then for each $\mathbf{i} \in I$ there exists a permutation π such that $b_{\pi(\mathbf{i})} = 1$. But if $b_{\pi(\mathbf{i})} = 1$, then it must be the case that for some $r, s \in \mathbf{i}$, $x_r \in \mathrm{QA}(x_s)$.

Therefore, if the board is covered, then a subset X, as described above, does not exist. Conversely, if the board is not covered, then there exists a pairwise disjoint $\mathbf{i}$ such that $b_{\mathbf{i}}=0$ and for all permutations π, $b_{\pi(\mathbf{i})}=0$. For such an $\mathbf{i}$ to exist it must be the case that for all $r, s\in\mathbf{i}$, $r\neq s$, we have $x_r\notin \mathrm{QA}(x_s)$. The set $X=\{x_r: r\in\mathbf{i}\}$ is a suitable set.

Now we count the pebbles on the board. There are

$$(2^n)^{p_k(n)+1}-\frac{2^n!}{(2^n-p_k(n)-1)!}$$

pebbles on the board for those elements in I that are not pairwise disjoint. Furthermore, since there are at most 2^n sets $\mathrm{QA}(x)$, and since for every x, $\|\mathrm{QA}(x)\|\leqq p_k(n)$, it follows that there exist at most $p_k(n)2^n$ queries $x_r\in\mathrm{QA}(x_s)$, $r\neq s$. For each such query, the number of pebbles placed on the board is the number of different subsets S of I as given above. There are

$$\binom{2^n-2}{p_k(n)-1}$$

such subsets. Hence there are at most

$$(2^n)^{p_k(n)+1}-2^n!/(2^n-p_k(n)-1)!+p_k(n)2^n\binom{2^n-2}{p_k(n)-1}$$

pebbles on the board.

By Lemma 2.2, it is sufficient to show that this number is smaller than

$$\binom{2^n}{p_k(n)+1}+(2^n)^{p_k(n)+1}-\frac{2^n!}{(2^n-p_k(n)-1)!}.$$

But this is equivalent to

$$p_k^2(n)(p_k(n)+1)<2^n-1.$$

Since we have chosen $p_k(n)<2^{n/3}-1$, this inequality is fulfilled. So it follows that the board cannot be covered. There must be a suitable subset $X\subseteq\Sigma^n$ such that $x_r\notin\mathrm{QA}(x_s)$ for all $x_s, x_r\in X$, $r\neq s$, and this proves the claim.

It follows from the usual argument that $L(G)\notin\mathbf{NP.ACC.DEP}^G$ and $L(G)\in\mathbf{NP}^G$. □

COROLLARY 2.4. *There is a recursive oracle H such that* $\mathbf{P}^H\neq\mathbf{U}^H\neq\mathbf{NP.ACC.DEP}^H\neq\mathbf{NP}^H$, *and the first inequality is strong.*

We do not know whether Corollary 2.4 can be strengthened so that all inequalities are strong.

3. Concluding remarks. Here we wish to make some remarks about the existence of one-way functions in a relativized setting. Recall that a one-way function is a 1-1, honest function that is computable in polynomial time but whose inverse is not polynomial time computable. One-way functions are known to play a critical role in complexity issues surrounding public-key cryptosystems. It is observed in Grollmann and Selman [6] that one-way functions exist if and only if $\mathbf{P}\neq\mathbf{U}$, so the results of Rackoff, as well as those herein, show that there do exist relativized worlds in which one-way functions exist.

It is also of interest to know whether there exist one-way functions with range belonging to **P**. Indeed, Grollmann and Selman show that this existence question is equivalent to whether $\mathbf{P}\neq\mathbf{U}\cap\text{co-}\mathbf{U}$. Using the techniques developed in this paper, it

is not hard to prove the existence of a recursive oracle A such that $\mathbf{P}^A \neq \mathbf{U}^A \cap \text{co-}\mathbf{U}^A \neq \mathbf{NP}^A$, and therefore, relative to oracle A, there exist one-way functions with easy to recognize range.

Acknowledgments. We would like to thank Tim Long for pointing out several errors in earlier versions of this paper. In addition, we wish to thank Alan Selman for his guidance and many helpful conversations.

REFERENCES

[1] L. ADLEMAN AND K. MANDERS, *Reducibility, randomness, and intractibility*, Proc. 9th ACM Symposium on Theory of Computing, 1977, pp. 151–153.

[2] T. BAKER, J. GILL AND R. SOLOVAY, *Relativization of the* P = ? NP *question*, this Journal, 4 (1975), pp. 431–442.

[3] J. L. BALCÁZAR AND D. RUSSO, *Immunity and simplicity in relativizations of probabilistic complexity classes*, manuscript, 1984.

[4] R. BOOK, T. LONG AND A. SELMAN, *Quantitative relativizations of complexity classes*, this Journal, 13 (1984), pp. 461–487.

[5] S. EVEN, A. SELMAN AND Y. YACOBI, *The complexity of promise problems with applications to public-key cryptography*, Inform. and Control, 61 (1984), pp. 159–173.

[6] J. GROLLMANN AND A. SELMAN, *Complexity measures of public-key cryptosystems*, Proc. 25th IEEE Symposum on Foundations of Computer Science, 1984, pp. 495–503.

[7] C. RACKOFF, *Relativized questions involving probabilistic algorithms*, J. Assoc. Comput. Mach., 29 (1982), pp. 261–268.

[8] U. SCHÖNING AND R. BOOK, *Immunity, nondeterminism, and relativization*, this Journal, 13 (1984), pp. 329–337.

[9] M. SIPSER, *On relativization and the existence of complete sets*, Automata, Languages, and Programming, Lecture Notes in Computer Science 140, Springer-Verlag, Berlin 1982, pp. 523–531.

[10] L. VALIANT, *Relative complexity of checking and evaluating*, Inform. Proc., 5 (1976), pp. 20–23.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
014

PROBABILITIES RELATED TO FATHER-SON DISTANCES IN BINARY SEARCH TREES*

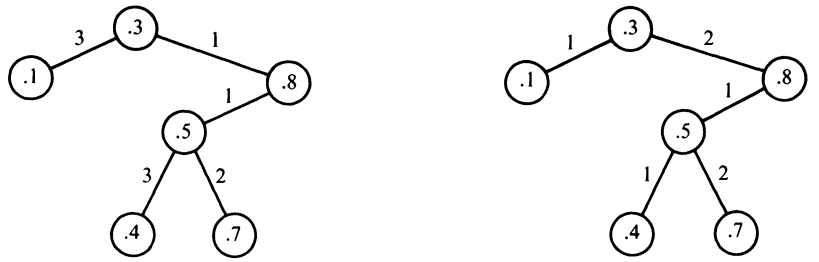
CARL E. LANGENHOP† AND WILLIAM E. WRIGHT‡

Abstract. We present some properties of a random variable called father-son distance, which is defined on binary search trees formed naturally from an insertion sequence $X_1, \cdots, X_n$ of independent identically-distributed continuous random variables. If X_i is the key in a node in such a tree and X_j is the key in a son of that node, then the father-son distance between the nodes is defined to be $j-i$. We obtain a general formula for the probability that a path from the root with specified successive father-son distances will be contained in a randomly generated tree.

Key words. binary search tree, internode distance, father-son distance, independent identically-distributed continuous random variables, order statistics, path length, inter-record times

1. Introduction. In this paper we derive formulas for the probabilities of certain events associated with the process of building a binary search tree by the insertion of a sequence of keys drawn at random from some ordered population such as the real numbers. A description of the insertion algorithm is given by Lynch [3] and by Knuth [2, p. 424]. If X_i denotes the value of the ith key to be inserted in the tree, then for $n \geqq 1$ we call $X_1, X_2, \cdots, X_n$ the insertion sequence. The first key X_1 is placed in the root node. Then each succeeding key is placed in a new node determined by that key's natural order relative to the keys already placed. (We assume $X_i \neq X_j$ if $i \neq j$.) This is done by searching the tree in normal fashion and then making the new node a left or right son, as appropriate, of the last node in the search path. No balancing is performed on the tree.

Given n different key values the tree that is constructed depends on the sequence in which these keys are inserted. The usual assumption, see Knuth [2, p. 427], is to assume all sequences are equally likely so the tree resulting from any specific sequence is assigned the probability $1/n!$. However, two different sequences can result in the same tree structure with the nodes containing precisely the same keys. This phenomenon is illustrated in Fig. 1 by the trees generated from the insertion sequences .3, .8, .5, .1, .7, .4 and .3, .1, .8, .5, .4, .7.



(a) *Insertion sequence*: .3, .8, .5, .1, .7, .4.

(b) *Insertion sequence*: .3, .1, .8, .5, .4, .7.

FIG. 1. *Randomly generated binary search trees with father-son distances.*

We introduce the notion of a father-son distance function defined on the branches of a binary search tree. Suppose for such a tree that the key X_i is located in node v and the key X_j is located in a son v' of this node. Then we must have $j > i$ and we

* Received by the editors August 31, 1982, and in revised form December 10, 1984.

† Department of Mathematics, Southern Illinois University, Carbondale, Illinois 62901.
‡ Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901.

define the father-son distance of the branch from v to v', denoted $d(v, v')$, by the relation

$$d(v, v') = j - i. \tag{1.1}$$

In Fig. 1 the numbers associated with each branch of the trees shown are the father-son distances determined by the respective insertion sequences. It is clear that if one is given a tree and the father-son distances for each branch, then one can recover the insertion sequence of the key values appearing in the nodes regardless of what these key values are. In fact, the structure of a binary search tree merely specifies the numerical or linear order of the key values in the nodes of the tree while the associated father-son distances specify the order in which those keys were inserted.

Disregarding the values of the keys we shall refer to the binary trees of n nodes with associated father-son distances as *distanced binary search trees with n nodes* or, more briefly as distanced trees. We denote by S_n the collection of all such distanced trees. Clearly there are $n!$ elements in the set S_n, a distinct element being generated by each of the $n!$ permutations of a given set of n keys used as the insertion sequence.

Ordinarily we may suppose that the nodes of a binary search tree are stored sequentially in a computer memory as the tree is generated by the input of a sequence of keys. In this case the distance function d gives the number of cells in memory from a father node to a son. Comparison searching for a particular key in an already generated binary search tree requires moving along a unique path from the root node to the node containing that key or to the corresponding null link when that key is not yet in the tree. For each node in the path except the last one (or corresponding null link), it is necessary to select the right or left branch and then move to the selected right or left son. In some applications the "cost" of moving from a father to a son may depend on the corresponding father-son distance. Thus probability distributions associated with the father-son distance function d are of potential utility.

Aside from such applications, the father-son distances in certain cases, but with different terminology, have already been recognized as of some interest in statistics. The cases considered correspond to paths through the extreme right nodes (or paths through the extreme left nodes) of a binary search tree. In the statistical terminology the keys in the extreme nodes are "record values"; each is larger (or smaller) than any preceding it. The father-son distances between these nodes are the "inter-record times". (See Chandler [1] or Siddiqui and Biondini [4].) We shall say more about this application later.

It is our purpose to derive formulas for the probabilities of certain subsets in S_n. A subset of the type considered is denoted $Q(e_1, \cdots, e_k; h_1, \cdots, h_k)$ to indicate the set of all trees in S_n having the father-son distances $h_1, \cdots, h_k$ along the path $e_1, \cdots, e_k$ from the root to a node k levels below the root. Here $e_1, \cdots, e_k$ is a sequence of zeros and ones specifying the path, with $e_j = 0$ denoting that the jth branch of the path is a left branch and $e_j = 1$ denoting that it is a right branch, $1 \leqq j \leqq k$. The h_j are positive integers specifying the father-son distances along the successive branches in the path. Regardless of their values, the keys in the successive nodes along this path, beginning with the root, are X_1, X_{1+h_1}, $X_{1+h_1+h_2}$, etc. Observe that for such trees in S_n we must have $1 + h_1 + \cdots + h_k \leqq n$. In case $1 + h_1 + \cdots + h_k > n$ we must interpret $Q(e_1, \cdots, e_k; h_1, \cdots, h_k)$ as the empty subset of S_n.

For both trees in Fig. 1 the path from the root to the node containing .7 as key is specified by $(e_1, e_2, e_3) = (1, 0, 1)$. However, the tree in Fig. 1(a) belongs to the set $Q(1, 0, 1; 1, 1, 2)$ while that in Fig. 1(b) belongs to $Q(1, 0, 1; 2, 1, 2)$.

As another example consider the set $Q(1, 1; 3, 5)$ in S_n with $n \geqq 9$. We can describe this set using the statistical terminology. It consists of all trees with new upper "record

values" at times 4 and 9. Each such tree is produced by an insertion sequence $X_1, X_2, \cdots, X_n$ satisfying the properties:

$$X_1 < X_4 < X_9 \qquad X_i < X_1 \quad \text{for } i = 2, 3, \qquad X_i < X_4 \quad \text{for } i = 5, 6, 7, 8.$$

The value X_1 is a "record" by convention and X_4 and X_9 are subsequent new "records"; the "inter-record times" are $4-1=3$ and $9-4=5$.

Suppose now that $1+h_1+\cdots+h_k \leqq s$ and $s+h_{k+1} \leqq n$ and suppose T is a tree in the subset $Q(e_1, \cdots, e_k;\ h_1, \cdots, h_k)$ of S_s. If the tree T has been generated by the insertion sequence of keys $X_1, \cdots, X_s$, then it will be converted to some tree T' in S_n by the subsequent insertion of keys $X_{s+1}, \cdots, X_n$. Whether the resulting tree T' is then in the subset $Q(e_1, \cdots, e_{k+1};\ h_1, \cdots, h_{k+1})$ on S_n is determined by how the values of the keys $X_{s+1}, \cdots, X_n$ are ordered relative to those of the keys $X_1, \cdots, X_s$ already in T. For example, we have seen that the tree T in Fig. 1(a) is in the subset $Q(1, 0, 1;\ 1, 1, 2)$ of S_6. In order for it to be converted into a tree in the subset $Q(1, 0, 1, 0;\ 1, 1, 2, 2)$ of S_n with $n \geqq 8$ the key X_7 must satisfy either $X_7 < X_3$ $(=.5)$ or $X_7 > X_5$ $(=.7)$ and then X_8 must satisfy $X_3 < X_8 < X_5$.

In order to determine the probabilities of the sets $Q(e_1, \cdots, e_k;\ h_1, \cdots, h_k)$ we have found it expedient to adopt a model in which the actual keys are observations of a sequence of independent random variables with a common uniform distribution on the interval $[0, 1]$. As in the example just mentioned, one may then consider conditions on $X_{s+1}, \cdots, X_n$ such that a tree generated by observations of $X_1, \cdots, X_s$ and in the subset $Q(e_1, \cdots, e_k;\ h_1, \cdots, h_k)$ of S_s may be converted into a tree in the subset $Q(e_1, \cdots, e_k, e_{k+1};\ h_1, \cdots, h_k, h_{k+1})$ in S_n. In our approach the particular observed values of the keys are of no concern and only the distanced tree structure is attended to. Nevertheless, our method associates the probability $1/n!$ with each tree in S_n, just as when one fixes the n keys ahead of time and uses the $n!$ permutations of these as insertion sequences to generate the $n!$ elements in S_n.

2. Probabilities of distanced paths. For each positive integer s let $Z(s) = \{1, 2, \cdots, s\}$ and let $X_1, X_2, \cdots$ be independent random variables uniformly distributed on $[0, 1]$. Let u be a permutation on $Z(s)$ and consider the event

$$0 < X_{u(1)} < \cdots < X_{u(s)} < 1. \tag{2.1}$$

That is, if $x_1, \cdots, x_s$ are observed values of $X_1, \cdots, X_s$, then (2.1) occurs if and only if

$$0 < x_{u(1)} < \cdots < x_{u(s)} < 1. \tag{2.1a}$$

If the numbers $x_1, \cdots, x_s$ are used as insertion sequence, a tree in S_s will be determined which we denote by $T(s, u)$. Conversely, given a tree in S_s we may place s numbers from $[0, 1]$ as keys in the nodes consistent with the tree structure. The father-son distances then determine the insertion sequence $x_1, \cdots, x_s$ of these numbers to generate the tree. When they are ordered in size as in (2.1a), a unique permutation u on $Z(s)$ is thereby specified. For this u the numbers $x_1, \cdots, x_s$ represent observations for which the event (2.1) occurs. Thus, obtaining the tree from observations of the variables $X_1, \cdots, X_s$ corresponds to the event described in (2.1). The probability of this event is given by

$$\int_0^1 \int_0^{x_{u(s)}} \cdots \int_0^{x_{u(2)}} dx_{u(1)} \cdots dx_{u(s)} = \frac{1}{s!}.$$

Hence, as we claimed earlier, our model for generating binary search trees gives

$$P(T(s, u)) = 1/s! \tag{2.2}$$

as the probability of obtaining any given tree $T(s, u)$ in S_s. In fact, one would get the same result so long as the random variables $X_1, X_2, \cdots$ are independent and have the same continuous distribution, i.e. $F(x) = P(X_i \leqq x)$ is a continuous function of the real variable x.

Along with the independent random variables $X_1, \cdots, X_s$ it is helpful to consider the corresponding order statistics which we denote by X_r^s, $1 \leqq r \leqq s$. Here X_1^s is the minimum of $\{X_1, \cdots, X_s\}$, X_2^s denotes the next smallest of $\{X_1, \cdots, X_s\}$, etc., with X_s^s denoting the maximum of $\{X_1, \cdots, X_s\}$. It will be convenient notationally to introduce constant random variables $X_0^s = 0$ and $X_{s+1}^s = 1$ for each $s \geqq 1$. With probability one we have

$$X_0^s < X_1^s < \cdots < X_{s+1}^s \tag{2.3}$$

since the probability is zero that any two of $X_1, \cdots, X_s$ are equal or that one of them is equal to zero or one. We note that the event (2.1) corresponding to the tree $T(s, u)$ can be described by the relations $X_r^s = X_{u(r)}$, $1 \leqq r \leqq s$. That is,

$$[T(s, u)] = [X_r^s = X_{u(r)},\ 1 \leqq r \leqq s] \tag{2.4}$$

where we use brackets, [ ], to denote the event described therein.

The relations (2.3) indicate that the random variables $X_1, \cdots, X_s$ divide the interval $(0, 1)$ into successive random intervals (X_r^s, X_{r+1}^s), $0 \leqq r \leqq s$. Moreover, with probability one, X_{s+1} must fall into one of these random intervals; otherwise $X_{s+1} = 0, 1$ or X_i for some i, $1 \leqq i \leqq s$, and these events have probability zero. Relative to binary search trees in S_n with $n \geqq s$, observations of the random intervals (X_r^s, X_{r+1}^s), $0 \leqq r \leqq s$, correspond to the null links in the subtree generated by the observations of the first s members of the sequence $X_1, \cdots, X_n$. If for this tree we have $X_r^s = X_i$ and $X_{r+1}^s = X_j$ for some r in the range $1 \leqq r \leqq s-1$, then $(X_r^s, X_{r+1}^s) = (X_i, X_j)$ corresponds to a null right link of the node with key X_i when $i > j$ and to a null left link of the node with key X_j when $i < j$. If the tree is such that $X_1^s = X_j$, then $(X_0^s, X_1^s) = (0, X_j)$ corresponds to a null left link of the node with key X_j and if $X_s^s = X_i$, then $(X_s^s, X_{s+1}^s) = (X_i, 1)$ corresponds to a null right link of the node with key X_i. In Fig. 2 the correspondence between null links and the observed intervals (X_r^s, X_{r+1}^s) is illustrated for the case in Fig. 1(a) where the observed sequence $X_1, \cdots, X_6$ is .3, .8, .5, .1, .7, .4.

If $X_1, \cdots, X_s$ generate the tree $T(s, u)$ and the event $X_r^s < X_{s+1} < X_{r+1}^s$ occurs for a specific r, $0 \leqq r \leqq s$, then a unique tree $T(s+1, u')$ in S_{s+1} is determined. This tree is

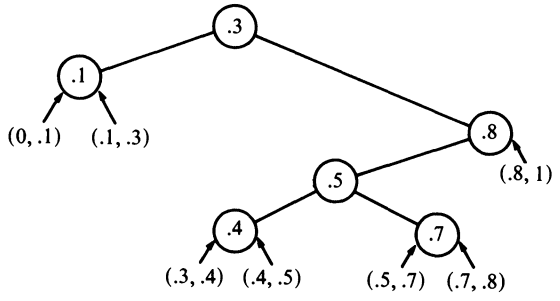


FIG. 2. *Correspondence between intervals* (X_r^s, X_{r+1}^s) *and null links.*

described by

$$X_\sigma^{s+1} = X_{u'(\sigma)}, \qquad 1 \leqq \sigma \leqq s+1,$$

where, by virtue of (2.4), the permutation u' on $Z(s+1)$ is given by

$$\begin{aligned} u'(\sigma) &= u(\sigma), \qquad 1 \leqq \sigma \leqq r, \\ u'(r+1) &= s+1, \\ u'(\sigma) &= u(\sigma-1), \qquad r+2 \leqq \sigma \leqq s+1. \end{aligned} \tag{2.5}$$

Indeed, if $I = (X_r^s, X_{r+1}^s)$, then we have

$$[T(s, u)] \cap [X_{s+1} \in I] = [T(s+1, u')]. \tag{2.6}$$

Clearly $T(s+1, u')$ is produced only in this way for if one deletes from $T(s+1, u')$ the node with the key X_{s+1}, then $T(s+1, u)$ remains since r and u are determined uniquely from (2.5).

Since $P(T(s, u)) = 1/s!$ for all positive integers s (and all permutations u on $Z(s)$), it follows from (2.6) that if $I = (X_r^s, X_{r+1}^s)$ for some r, $0 \leqq r \leqq s$, then

$$P(X_{s+1} \in I | T(s, u)) = \frac{1/(s+1)!}{1/s!} = \frac{1}{s+1}. \tag{2.7}$$

Note that this is independent of u as well as the value of r. Later we shall need to use the obvious fact

$$P(X_1 \in (0, 1)) = 1 \tag{2.8}$$

and to recognize the event $X_1 \in (0, 1)$ as the one tree in S_1 consisting merely of the root node. Denoting this tree by $T(1, 1)$, we may write

$$[X_1 \in (0, 1)] = [T(1, 1)]. \tag{2.9}$$

We wish to extend the relation (2.7). If we denote by $\mathscr{L}_s$ the set of ordered pairs (X_r^s, X_{r+1}^s), $0 \leqq r \leqq s$, corresponding to the successive null links in the trees in S_s, then $I \in \mathscr{L}_s$ is a key assumption leading to (2.7). Consider now ordered pairs $I = (A, B)$ in which A and B can be any of the random variables $X_1, \cdots, X_s$, $X_0^s = 0$, or $X_{s+1}^s = 1$. Some of these will be of the form (X_r^s, X_{r+1}^s) for some trees $T(s, u)$ and not for some others. For example, $I = (X_2, X_1)$ corresponds to a null link in the tree generated by X_1, X_2, X_3 in the case $X_2 < X_1 < X_3$ or $X_3 < X_2 < X_1$ but not when $X_1 < X_2 < X_3$ nor when $X_2 < X_3 < X_1$, etc. Some ordered pairs $I = (A, B)$ cannot be of the form (X_r^s, X_{r+1}^s) for any tree; the pairs $(1, X_2)$, (X_2, X_2), and $(X_1, 0)$ are examples of this.

In any case we shall let J_s denote the set of all ordered pairs $I = (A, B)$ in which A and B are from the set of random variables $X_1, \cdots, X_s$, 0, 1. Given $I = (A, B) \in J_s$, then $[I \in \mathscr{L}_s]$ is an event consisting of all trees $T(s, u)$ for which (A, B) corresponds to a null link. In particular, if $A = X_i$ and $B = X_j$ with $1 \leqq i, j \leqq s$ and $i \neq j$, then $I \in \mathscr{L}_s$ if and only if $X_i = X_r^s$ and $X_j = X_{r+1}^s$ for some r, $1 \leqq r \leqq s-1$. Using (2.4), we see that $T(s, u)$ is in the event $[I \in \mathscr{L}_s]$ if and only if $u(r) = i$ and $u(r+1) = j$ for some r, $1 \leqq r \leqq s-1$. Similar identifications can be made in the case $I = (0, X_j)$ with $1 \leqq j \leqq s$ or in the case $I = (X_i, 1)$ with $1 \leqq i \leqq s$. For all other cases the event $[I \in \mathscr{L}_s]$ is empty.

For $I = (A, B) \in J_s$ we shall use the notation $\mathscr{N}_s(I) = \mathscr{N}_s(A, B)$ for the class of all subevents of $[I \in \mathscr{L}_s]$. That is, $M \in \mathscr{N}_s(I)$ if and only if

$$M = \bigcup_{u \in R} [T(s, u)] \tag{2.10}$$

for some set R of permutations u on $Z(s)$ such that for some r, $0 \leqq r \leqq s$, we have

$$(A, B) = (X_r^s, X_{r+1}^s) \tag{2.11}$$

by virtue of the relations $X_\sigma^s = X_{u(\sigma)}$, $1 \leqq \sigma \leqq s$, along with $X_0^s = 0$ and $X_{s+1}^s = 1$.

If $I = (A, B) \in J_s$, then by $[X_{s+1} \in I]$ we mean the event that $A < X_{s+1} < B$. Our extension of (2.7) is the following:

LEMMA 2.1. *For any $s \geqq 1$, if $I \in J_s$ and $M \in \mathcal{N}_s(I)$, then*

$$P(X_{s+1} \in I | M) = \frac{1}{s+1} \tag{2.12}$$

and

$$P(X_{s+1} \notin I | M) = \frac{s}{s+1}. \tag{2.13}$$

Proof. Using (2.10), we have

$$P(M \cap [X_{s+1} \in I]) = \sum_{u \in R} P([T(s, u)] \cap [X_{s+1} \in I]). \tag{2.14}$$

Now for each $u \in R$ we have $I \in \mathcal{L}_s$. Hence we may use (2.7) in (2.14) to get

$$P(M \cap [X_{s+1} \in I]) = \sum_{u \in R} P([T(s, u)]) \cdot \frac{1}{s+1} = \frac{1}{s+1} P(M).$$

Relation (2.12) now follows and (2.13) is an immediate consequence of (2.12). □

Remark. The assumption $M \in \mathcal{N}_s(I)$ is essential here. For example, suppose $I = (X_1, X_2)$ and $T(3, u)$ is generated where $X_1 < X_3 < X_2$. Then $T(3, u) \notin [I \in \mathcal{L}_3]$. In fact, $[T(3, u)] \cap [X_4 \in I]$ is the event that X_4 enters either the null link (X_1, X_3) or the null link (X_3, X_2) for $T(3, u)$. The corresponding term in the right side of (2.14) would thus be double what it would be if $T(3, u)$ were in $[I \in \mathcal{L}_3]$. On the other hand, if $I = (X_2, X_1)$ and $T(3, u)$ is the same as before, then I does not correspond to any nontrivial union of null links in $T(3, u)$. In this case, $[T(3, u)] \cap [X_4 \in I]$ has probability zero since $[X_4 \in I]$ requires $X_2 < X_4 < X_1$ which is incompatible with the relations $X_1 < X_3 < X_2$ characterizing the tree $T(3, u)$.

In (2.12) and (2.13) the events $M \cap [X_{s+1} \in I]$ and $M \cap [X_{s+1} \notin I]$ are important. For these we have the following:

LEMMA 2.2. *Suppose $s \geqq 1$ and $I = (A, B) \in J_s$. If $M \in \mathcal{N}_s(I)$, then*

$$M \cap [X_{s+1} \in I] \in \mathcal{N}_{s+1}(A, X_{s+1}) \cap \mathcal{N}_{s+1}(X_{s+1}, B) \tag{2.15}$$

and, if we ignore an event of probability zero,

$$M \cap [X_{s+1} \notin I] \in \mathcal{N}_{s+1}(I). \tag{2.16}$$

Proof. Suppose $T(s, u)$ is represented in M through (2.10). Then $M \in \mathcal{N}_s(I)$ implies that $I = (A, B)$ is a null link in this tree. This tree together with $[X_{s+1} \in I]$ yield a tree for which both (A, X_{s+1}) and (X_{s+1}, B) are null links. This is true for every tree in the event M and (2.15) is thus seen to be valid. Similarly, if $T(s, u)$ is represented in M and $X_{s+1} \notin I$, then the key X_{s+1} must be inserted into some other null link of $T(s, u)$ or it must equal $X_1, \cdots, X_s, 0$, or 1. These latter possibilities all have probability zero. Hence, except for an event of probability zero, $[T(s, u)]$ and $[X_{s+1} \notin I]$ yield one or more trees in S_{s+1} for which I remains a null link, i.e., for which $I \in \mathcal{L}_{s+1}$. Since this is the case for each $T(s, u)$ represented in M, the assertion (2.16) is seen to be valid. □

The results in Lemmas 2.1 and 2.2 enable us to obtain the probabilities of the events $Q(e_1, \cdots, e_k; h_1, \cdots, h_k)$. Consider first the events $Q(e_1; h_1)$ when $n \geqq 1+h_1$. If $e_1=0$, we define $I_1=(0, X_1)$ and if $e_2=1$, we define $I_1=(X_1, 1)$. We then have $I_1=(X_0^1, X_1^1)$ when $e_1=0$ and $I_1=(X_1^1, X_2^1)$ when $e_1=1$. If we take $M_1=[T(1,1)]=[X_1 \in (0,1)]$ as in (2.9), then in either case we have $M_1 \in \mathcal{N}_1(I_1)$. Also in either case we may write

$$Q(e_1; h_1) = M_1 \cap \left(\bigcap_{j=2}^{h_1} [X_j \notin I_1] \right) \cap [X_{h_1+1} \in I_1]. \tag{2.17}$$

Applying (2.16) inductively, we conclude that

$$M_p = \left[M_1 \cap \left(\bigcap_{j=2}^{p} [X_j \notin I_1] \right) \right] \in \mathcal{N}_p(I_1) \tag{2.18}$$

for $1 \leqq p \leqq h_1$. But for $2 \leqq p \leqq h_1$ we have $M_p = M_{p-1} \cap [X_p \notin I_1]$ so, using (2.13), we get

$$P(M_p) = P(M_{p-1}) \frac{p-1}{p}. \tag{2.19}$$

But $P(M_1)=1$ by (2.8) so (2.19) implies

$$P(M_{h_1}) = 1 \cdot \frac{2-1}{2} \cdot \frac{3-1}{3} \cdots \frac{h_1-1}{h_1} = \frac{1}{h_1}. \tag{2.20}$$

By (2.18) we can write (2.17) as

$$Q(e_1; h_1) = M_{h_1} \cap [X_{h_1+1} \in I_1]. \tag{2.21}$$

Since $M_{h_1} \in \mathcal{N}_{h_1}(I_1)$ we may apply (2.12) to get

$$P(Q(e_1; h_1)) = \frac{1}{h_1} \cdot \frac{1}{h_1+1} \tag{2.22}$$

in light of (2.20).

It may be noted that although $M_1=[T(1,1)]$ consists of just one tree, the events M_p defined in (2.18) consist of more than one tree when $p \geqq 3$. For example, in the case $e_1=1$ for which $I_1=(X_1, 1)$ the set M_2 consists of the one tree determined by the relation $X_2 < X_1$ but M_3 consists of two trees, namely, the one generated by the conditions $X_3 < X_2 < X_1$ and the other generated by the conditions $X_2 < X_3 < X_1$. Thus Lemma 2.1 which extends the basic relation (2.7) is a key ingredient in our analysis.

In (2.21) we have $I_1=(A, B)$ with $A=0$, $B=X_1$ for the case $e_1=0$ and $A=X_1$, $B=1$ for the case $e_1=1$. Moreover, $M_{h_1} \in \mathcal{N}_{h_1}(I_1)$ so by (2.15) it follows that $Q(e_1; h_1) \in \mathcal{N}_{h_1+1}(I_2)$ whether we take $I_2=(A, X_{h_1+1})$ or $I_2=(X_{h_1+1}, B)$. For $Q(e_1, e_2; h_1, h_2)$ we make the first choice in the case $e_2=0$ and the second choice in the case $e_2=1$. We can then define $Q(e_1, e_2; h_1, h_2)$ by proceeding from $Q(e_1; h_1)$ similarly to the way $Q(e_1; h_1)$ is given in (2.17) proceeding from $M_1=[T(1,1)]$.

We give an inductive definition for the events $Q(e_1, \cdots, e_k; h_1, \cdots, h_k)$. For this the notations

$$s(k) = h_1 + \cdots + h_k, \qquad k \geqq 1, \tag{2.23}$$

and

$$M_{s(k)+1} = Q(e_1, \cdots, e_k; h_1, \cdots, h_k), \tag{2.24}$$

are convenient. Assume that $I_k = (A_k, B_k) \in J_{s(k)}$ and that

$$M_{s(k)+1} \in \mathcal{N}_{s(k)+1}(A_k, X_{s(k)+1}) \cap \mathcal{N}_{s(k)+1}(X_{s(k)+1}, B_k). \tag{2.25}$$

If $e_{k+1} = 0$, we take $I_{k+1} = (A_k, X_{s(k)+1})$ and if $e_{k+1} = 1$, we take $I_{k+1} = (X_{s(k)+1}, B_k)$. In either case (2.25) implies

$$M_{s(k)+1} \in \mathcal{N}_{s(k)+1}(I_{k+1}). \tag{2.26}$$

Now define

$$M_p = M_{s(k)+1} \cap \left(\bigcap_{j=s(k)+2}^{p} [X_j \notin I_{k+1}] \right) \tag{2.27}$$

for $s(k)+2 \leqq p \leqq s(k+1) = s(k) + h_{k+1}$. Then (2.26) and (2.27) imply $M_p \in \mathcal{N}_p(I_{k+1})$ for such p by virtue of (2.16) in Lemma 2.2. Finally then, we have

$$Q(e_1, \cdots, e_{k+1};\ h_1, \cdots, h_{k+1}) = M_{s(k+1)} \cap [X_{s(k+1)+1} \in I_{k+1}]. \tag{2.28}$$

Since $M_{s(k+1)} \in \mathcal{N}_{s(k+1)}(I_{k+1})$, we see by (2.15) and (2.24) that (2.25) holds with k replaced by $k+1$. We chose $I_1 \in J_1$ and verified (2.25) for the case $k = 1$, so (2.25) and hence also (2.26) hold for all $k \geqq 1$.

We may now prove our main result.

THEOREM 2.3. *Let $e_1, \cdots, e_k$ be any sequence of zeros and ones and let $h_1, \cdots, h_k$ be any sequence of positive integers. If $s(k)+1 \leqq n$, then in S_n we have*

$$\begin{aligned} P(Q(e_1, \cdots, e_k;\ h_1, \cdots, h_k)) &= \frac{1}{s(k)+1} \prod_{i=1}^{k} \frac{1}{s(i)} \\ &= \frac{1}{h_1(h_1+h_2)\cdots(h_1+\cdots+h_k)(h_1+\cdots+h_k+1)}. \end{aligned} \tag{2.29}$$

Proof. For $k = 1$ this reduces to (2.22) which was established above. Assume then that (2.29) holds for some $k \geqq 1$. Now for M_p as in (2.27) we saw above that $M_p \in \mathcal{N}_p(I_{k+1})$ for $s(k)+1 \leqq p \leqq s(k+1)$. It follows then from (2.27) and (2.13) in Lemma 2.1 that (2.19) holds for $s(k)+2 \leqq p \leqq s(k+1)$. Hence, by virtue of the inductive hypothesis (2.29) we get

$$\begin{aligned} P(M_{s(k+1)}) &= P(M_{s(k)+1}) \prod_{p=s(k)+2}^{s(k+1)} \frac{p-1}{p} \\ &= \left(\frac{1}{s(k)+1} \prod_{i=1}^{k} \frac{1}{s(i)} \right) \frac{s(k)+1}{s(k+1)} = \prod_{i=1}^{k+1} \frac{1}{s(i)}. \end{aligned} \tag{2.30}$$

Finally by (2.28), (2.30) and (2.12) in Lemma 2.1 we get

$$P(Q(e_1, \cdots, e_{k+1};\ h_1, \cdots, h_{k+1})) = \left(\prod_{i=1}^{k+1} \frac{1}{s(i)} \right) \frac{1}{s(k+1)+1}$$

which is (2.29) with k replaced by $k+1$. □

Remark. We mentioned earlier that there has been some interest in statistical theory in the events corresponding to our sets $Q(e_1, \cdots, e_k;\ h_1, \cdots, h_k)$ in which each $e_i = 1$ (or each $e_i = 0$). The probabilities for these cases are known and, of course, they agree with our result in (2.29). See, for example, K. N. Chandler [1 eq. (4)] where $u_j - 1$ there is our $s(j-1)$ and where $n-1$ there is our k. Other cases, i.e. when there are both zeros and ones among the e_i's, evidently have not been considered previously in statistical theory. A significant feature of our result in Theorem 2.3 is that the

probability of $Q(e_1, \cdots, e_k; h_1, \cdots, h_k)$ does not depend on the path $e_1, \cdots, e_k$ but only on the father-son distances $h_1, \cdots, h_k$ along this path. These distances are the analogues of the "inter-record times" for the extreme paths. Thus known properties of the probability distributions of "inter-record times" can be applied directly to father-son distances along any other path of potential interest.

3. Averages related to the father-son distance. Here we consider the computation of some averages using the probabilities in (2.29). The term "average" is often used in connection with some feature of binary search trees. One should be careful, however, to distinguish between a random variable which is an average defined on each tree in S_n and the expectation of that random variable over all trees in S_n.

Consider, for example, the "average path length of a tree" in S_n. Let us define $\lambda(\nu) = k$ if node ν in a tree T in S_n is k levels below the root. That is, $\lambda(\nu)$ is the "path length" to node ν in the tree T with the convention that $\lambda(\nu) = 0$ if ν is the root node. If the nodes in T are indexed $\nu_1, \cdots, \nu_n$, then $\Lambda(T) = \sum_{i=1}^{n} \lambda(\nu_i)/n$ is the average length for T and Λ is a random variable on S_n. The expected value of Λ is what is usually meant by "the average path length of a tree" in S_n. This is a known quantity and denoted by C_n in Knuth [2, p. 427]. We may write it as

$$C_n = \frac{1}{n!} \sum_{T \in S_n} \Lambda(T)$$

since $P(T) = 1/n!$ for each T in S_n. It is the case (see Knuth [2, p. 427]) that

$$C_n = 2(1 + 1/n)H_n - 3 \tag{3.1}$$

where

$$H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} k^{-1}. \tag{3.2}$$

The numbers C_n are related to numbers C'_n called the average number of comparisons made in an unsuccessful search for a key in a tree in S_n. The relation is (see Knuth [2, p. 427])

$$C_n = (1 + 1/n)C'_n - 1. \tag{3.3}$$

Let us indicate how C'_n can be computed using our formula (2.29).

To this end let k and s denote integers such that $1 \leqq k \leqq s \leqq n-1$ and let $\Gamma(s, k)$ denote the set of all trees in S_n having a path of k links with total father-son distance s. Now for each tree T in S_n there is one and only one path whose total father-son distance is s, namely, the path to the node containing the key X_{s+1} if T is generated by the insertion sequence $X_1, \cdots, X_n$. This path will have k links for some integer k satisfying $1 \leqq k \leqq s$. Note that the sets $\Gamma(s, k)$ for $1 \leqq k \leqq s$ are disjoint and their union is S_n. Accordingly there is a random variable on S_n which we denote by L_s which is defined by $L_s(T) = k$ if $T \in \Gamma(s, k)$.

If $T \in \Gamma(s, k)$ is generated by the insertion sequence $X_1, \cdots, X_n$ then in building the tree k comparisons with the keys $X_1, \cdots, X_s$ are required to place the key X_{s+1} in the tree. Thus $L_s(T)$ is the number of comparisons made in an unsuccessful search for the key X_{s+1} in the subtree of s keys $X_1, \cdots, X_s$. Thus

$$C'_s = \operatorname{Exp}(L_s) = \sum_{k=1}^{s} kP(\Gamma(s, k)). \tag{3.4}$$

Now $\Gamma(s, k)$ consists of all events $Q(e_1, \cdots, e_k; h_1, \cdots, h_k)$ such that

$$s(k) = h_1 + \cdots + h_k = s. \tag{3.5}$$

There are 2^k different choices for the path $e_1, \cdots, e_k$ so, using (2.29), we may write

$$P(\Gamma(s, k)) = \frac{2^k}{s+1} \sum \prod_{i=1}^{k} \frac{1}{s(k)} \tag{3.6}$$

where the summation is over all sequences $h_1, \cdots, h_k$ of k positive integers satisfying (3.5). The substitution of (3.6) into (3.4) then gives an explicit formula for the numbers C'_s. We have verified directly that this agrees with the result described in Knuth [2, Prob. 6, p. 448]. The verification is somewhat involved so we do not give the details here since this does not seem to be an efficient procedure for computing the C'_s.

In the remainder of this section we indicate how the formula (2.29) may be used in some other situations of potential interest. First, for $n \geqq 2$ let $K_n(e_1)$ denote the set of all trees T in S_n for which the root has an e_1-son. That is, $T \in K_n(e_1)$ if and only if $T \in Q(e_1; h_1)$ for some h_1, $1 \leqq h_1 \leqq n-1$. The events $Q(e_1; h_1)$ are disjoint for different h_1 so, using (2.29), we get

$$P(K_n(e_1)) = \sum_{h_1=1}^{n-1} \frac{1}{h_1(1+h_1)} = \sum_{h_1=1}^{n-1} \left(\frac{1}{h_1} - \frac{1}{h_1+1} \right) = \frac{n-1}{n}. \tag{3.7}$$

On $K_n(e_1)$ we can define a random variable D_1 by the condition $D_1(T) = h_1$ if $T \in Q(e_1; h_1)$. Using (2.29) and (3.7), we can compute the conditional expectation of the distance to the e_1-son of the root given that there is such a son. The result is

$$\operatorname{Exp}[D_1 | K_n(e_1)] = \sum_{h_1=1}^{n-1} h_1 \frac{1}{h_1(1+h_1)} \frac{n}{n-1} = \frac{n}{n-1}[H_n - 1] \tag{3.8}$$

with H_n as given in (3.2). In the statistical terminology this can be described as the average time to set a new record (considering the first observation X_1 as a record) given that a new record is set within the time interval $n-1$. Analogous but more involved subpopulations of trees and random variables dealing with a specific path $e_1, \cdots, e_k$ by time n can be handled similarly.

As a final example of the use of the formula (2.29) consider for $k \geqq 1$ the set $Q^* = Q(e_1, \cdots, e_k; h_1, \cdots, h_k)$ for a fixed path $e_1, \cdots, e_k$ and fixed father-son distances $h_1, \cdots, h_k$. Let $s = s(k) = h_1 + \cdots + h_k$. Now consider a tree in S_n generated by the insertion sequence $X_1, \cdots, X_n$. If $n \geqq 1 + s$, then the last node in the path $e_1, \cdots, e_k$ for the tree contains the key X_{1+s}. If $n > 1 + s$, this path in T may be extended in the e ($= 0$ or 1) direction with a father-son distance of h on the extending link provided $1 \leqq h \leqq n - 1 - s$. The quantity h may be interepreted as the waiting time for extending the given path $e_1, \cdots, e_k$ in the e direction from the last node of this path. For convenience let $Q^*(e; h) = Q(e_1, \cdots, e_k, e; h_1, \cdots, h_k, h)$. Then for $n > 1 + s$ we define this waiting time by $W_n^*(T) = h$ if $T \in S_n \cap Q^*(e, h)$. Since $Q^*(e; h) \subset Q^*$, then for $n > 1 + s$ we get from (2.29) that

$$P(W_n^* = h) = P(Q^*(e; h) | Q^*) = \frac{1+s}{(s+h)(1+s+h)}, \qquad 1 \leqq h \leqq n-1-s. \tag{3.9}$$

The events $Q^*(e; h)$ are disjoint for different values of h. Summing (3.9), we get

$$\sum_{h=1}^{n-1-s} P(W_n^* = h) = 1 - \frac{1+s}{n} \tag{3.10}$$

for the probability that the path of Q^* is extended in the direction e by the time the tree contains n nodes (the time the key X_n is put in the tree). Thus there is the probability $(1+s)/n$ that the path will not be extended in the e direction by the time the tree has n nodes.

If we choose not to limit the length of the insertion sequence $X_1, X_2, \cdots$, we may consider the class $Q^* = Q(e_1, \cdots, e_k;\ h_1, \cdots, h_k)$ of all binary search trees T with infinitely many nodes which contain the path $e_1, \cdots, e_k$ from the root with father-son distance $h_1, \cdots, h_k$. With $Q^*(e;\ h)$ related to Q^* as before, we define the waiting time W^* by $W^*(T) = h$ if $T \in Q^*(e;\ h)$. But regardless of h we may take any $n \geqq 1+s+h$ and consider the subtree T_n of T consisting of the first n nodes (including the root) of T. For any such n we have $T_n \in Q^*(e;\ h)$ when $T \in Q^*(e;\ h)$ and, using (2.29), we get

$$P(W^* = h) = \frac{1+s}{(s+h)(1+s+h)}, \qquad h \geqq 1 \tag{3.11}$$

as in (3.9). Now, however we allow any $h \geqq 1$ and we note that

$$\sum_{h=1}^{\infty} P(W^* = h) = 1.$$

Thus (3.11) is the probability distribution for the wating time W^* which takes on integer values $h \geqq 1$. Note that

$$\sum_{h=1}^{\infty} hP(W^* = h) = \infty;$$

that is, the waiting time has infinite expectation. For the special case when all $e_j = 1$, $1 \leqq j \leqq k$, this corresponds to the known fact that the "inter-record times" have infinite expectation. The feature of interest here is that the distribution of W^* given in (3.11) is independent of the path $e_1, \cdots, e_k$ and the direction of the extension. It is also independent of the particular father-son distances $h_1, \cdots, h_k$ so long as $\sum_{j=1}^{k} h_j = s$.

REFERENCES

[1] K. N. CHANDLER, *The distribution and frequency of record values*, J. Roy. Statist. Soc. Ser. B, 14 (1952), pp. 220-228.

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

[3] W. C. LYNCH, *More combinatorial properties of certain trees*, Comput. J., 7 (1965), pp. 229-302.

[4] M. M. SIDDIQUI AND R. W. BIONDINI, *The joint distribution of record values and inter-record times*, Ann. Probab., 3 (1975), pp. 1012-1013.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
015

NEGATION IS POWERLESS FOR BOOLEAN SLICE FUNCTIONS*

L. G. VALIANT†

Abstract. It is shown that for any slice function of n variables the monotone circuit complexity exceeds the circuit complexity over a universal basis by at most a multiplicative constant factor and an additive term of order $O(n(\log n)^2)$.

Key words. Boolean circuits, circuit complexity, monotone circuits

1. Introduction. The question of determining how much economy the universal basis $\{\wedge, \vee, \neg\}$ provides over the monotone basis $\{\wedge, \vee\}$ has been a long standing open problem in Boolean circuit complexity. In 1981, S. Berkowitz [3] made a striking observation which showed that, when suitably formulated, this problem does have a simple solution. For a Boolean function $f(x_1, \cdots, x_n)$ he defines the kth *slice* function of f to equal

(i) $f(x_1, \cdots, x_n)$ if exactly k of the inputs are one,

(ii) 0 if fewer than k of the inputs are one, and

(iii) 1 if more than k of the inputs are one.

Clearly, for most purposes, the function f is well represented by its $n+1$ slices. Now suppose that $g(x_1, \cdots, x_n)$ is any slice function (i.e. for some k) and suppose that circuit $X(x_1, \cdots, x_n)$ computes it over the basis $\{\wedge, \vee, \neg\}$. It is known that X can be transformed to a monotone circuit $X^*(x_1, \cdots, x_n, z_1, \cdots, z_n)$ having at most twice as many gates [6] such that X^* computes g if $\bar{x}_j$ is substituted for each z_j. It is also known [1] that there exist $O(n \log n)$ size monotone circuits for Boolean sorting and hence for computing any threshold function $\mathrm{Th}_k(x_1, \cdots, x_n)$. Here Th_k is the function that takes value one if and only if at least k of the inputs are ones. The surprising discovery, whose correctness the reader can easily verify, is the following:

BERKOWITZ' LEMMA. *If g is a kth slice function then $X^*(x_1, \cdots, x_n, z_1, \cdots, z_n)$ will compute $g(x_1, \cdots, x_n)$ if for $1 \leqq j \leqq n$, z_j is set to $\mathrm{Th}_k(x_1, \cdots, x_{j-1}, x_{j+1}, \cdots, x_n)$.*

Let $C_{\wedge,\vee}$ and $C_{\wedge,\vee,\neg}$ be the minimal circuit size complexity measures over the two bases. Then the following is an immediate consequence:

COROLLARY 1. *If g is any slice function then*

$$C_{\wedge,\vee}(g) \leqq 2C_{\wedge,\vee,\neg}(g) + O(n^2 \log n).$$

The purpose of the current paper is to improve the additive term to $O(n(\log n)^2)$ by showing how $\bar{x}_1, \cdots, \bar{x}_n$ can be computed simultaneously by a monotone circuit of that size.

DEFINITION. A *monotone (k, n)-inverter* is a monotone circuit with inputs $x_1, \cdots, x_n$ and outputs $y_1, \cdots, y_n$ with the properties that:

(a) If exactly k inputs are 1 then $y_j = \bar{x}_j$ for $1 \leqq j \leqq n$.

(b) If more than k inputs are 1 then $y_j = 1$ for $1 \leqq j \leqq n$.

(c) If fewer than k inputs are 1 then $y_j = 0$ for $1 \leqq j \leqq n$.

THEOREM. *There is a constant α such that for all n and k there is a monotone (k, n)-inverter with fewer than $\alpha n(\log_2 n)^2$ gates.*

* Received by the editors July 16, 1984 and in revised form January 28, 1985. This work was supported in part by the National Science Foundation under grant MCS-83-02385.

† Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts 02138.

The main shortcoming of Corollary 1 as stated is that it is too weak to be relevant to known lower bounds for monotone complexity. The reason for this is that the latter are all of order $O(n^2)$, and this growth is swamped by the additive term of $\theta(n^2 \log n)$ in the corollary. In contrast to this, our theorem can be applied meaningfully to the problems such as matrix multiplication and convolution that have been analyzed in the literature [4], [7]-[11], [13], [15]. For example, in the case of Boolean matrix multplication for $m \times m$ matrices it is known that any monotone circuit requires m^3 gates while over a universal basis $O(m^{2.81})$ or even $O(m^{2.5})$ are sufficient [5], [12]. A corollary of our theorem is that the slice function that corresponds to matrix multiplication when, for example, exactly one half of the $2m^2$ inputs are one can be computed by a monotone circuit with $O(m^{2.81})$ or $O(m^{2.5})$ gates. Intuitively this appears to say that the known lower bounds arguments for monotone circuits say as much about the formulation of the problem computed as about the model of computation or the problem itself.

We shall state our improved corollary for sets of functions since that is the only case for which nonlinear lower bounds are currently available.

DEFINITION. A *slice function set* F is a set $f_1(x_1, \cdots, x_n), \cdots, f_r(x_1, \cdots, x_n)$ of Boolean functions each of which is a kth slice for the same k.

COROLLARY 2. *For all slice function sets* F *over* $\{x_1, \cdots, x_n\}$ $C_{\wedge,\vee}(F) \leqq 2C_{\wedge,\vee,\neg}(F) + O(n(\log n)^2)$.

Further results on slice functions have been obtained recently by Wegener [14].

2. The construction. The construction is based on monotone subcircuits that can merge two sorted lists of length m in $O(m \log m)$ gates. Batcher's odd-even merge suffices here [2].

As a first simplification we note that it is sufficient to construct an inverter that is correct when exactly k of the inputs are 1. From the outputs $\{y_i\}$ of such a circuit the correct outputs $\{y_i^*\}$ can be computed by letting

$$y_i^* = (y_i \wedge \mathrm{Th}_k(x_1, \cdots, x_n)) \vee \mathrm{Th}_{k+1}(x_1, \cdots, x_n)$$

for each i. Clearly the y^* outputs will be all zero if $\mathrm{Th}_k = 0$ and will be all one if $\mathrm{Th}_{k+1} = 1$. This adds only $O(n(\log n)^2)$ gates to the overall complexity if Batcher's sorter is used, or $O(n \log n)$ if [1] is used.

As a further simplification we consider only the case $k \leqq n/2$. To obtain a (k, n)-inverter for $k > n/2$ it is sufficient to construct a $(k, 2n)$-inverter and set half of the inputs to zero. For similar reasons we can also assume that n is an exact power of 2.

The circuit we construct will have depth $\theta((\log n)^2)$. We distinguish the gates at $2 \log_2 n$ of the levels and call these distinguished levels *layers*. The layers are numbered $\{i | 0 \leqq i < \log_2 n\} \cup \{2 \log_2 n - i | 0 \leqq i < \log_2 n\}$. Layers i and $2 \log_2 n - i$ consist of $n/2^i$ *lists* each containing 2^i gates. Each list A *spans* a subset span $(A) \subseteq \{x_1, \cdots, x_n\}$ indexed by $\{r2^i + 1, r2^i + 2, \cdots, r2^i + 2^i\}$ for some r $(0 \leqq r < n/2^i)$. Level 0 consists of n singleton lists, consisting of the inputs $x_1, \cdots, x_n$, respectively. Level $2 \log_2 n$ also consists of n singleton lists, consisting of the outputs $y_1, \cdots, y_n$, respectively. The span of the list consisting of y_j is $\{x_j\}$. The *complement* of a list A, denoted by $\bar{A}$, is the concatenation of all the lists other than a that are in the same layer as A.

Above we have defined the items in the lists to be gates or input variables. When the inputs are given Boolean values these items take on values (i.e. the values of the inputs or the values computed by the gates.) We shall, for convenience, identify these lists with the list of Boolean values they take. The construction will ensure that each such list is sorted in increasing order. This is ensured simply because the only subcompu-

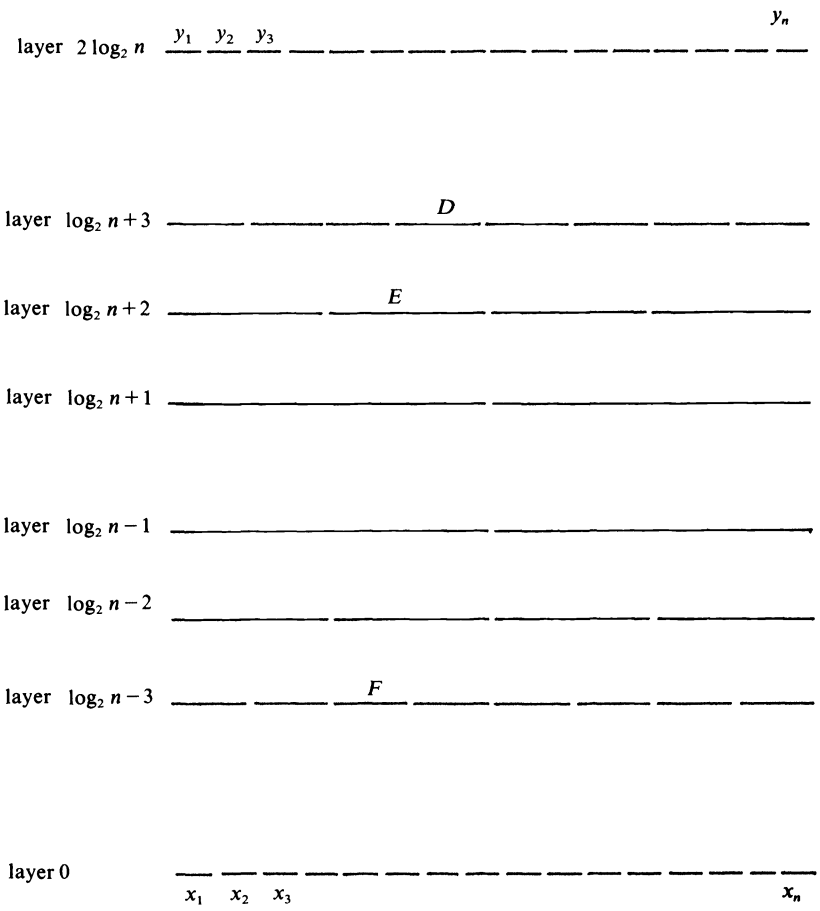


FIG. 1. *Schematic diagram of the construction.*

tations performed are merges of sorted lists. A schematic view of the construction is given in Fig. 1.

We denote lists by $A, B, C, \cdots$ and their layer numbers by layer $(A), \cdots$ etc. The description of the circuit is as follows:

(i) If layer $(A) = 0$ and span $(A) = \{x_j\}$ then A is the input x_j.

(ii) If layer $(A) = i$ and $1 \leqq i < \log_2 n$ then $A = \text{merge}(B, C)$ where layer $(B) =$ layer $(C) = i - 1$ and span $(A) =$ span $(B) \cup$ span (C).

(iii) If layer $(D) = \log_2 n + 1$ then D is the concatenation of the last k bits of G with a list of $(n/2 - k)$ 1's where layer $(G) = \log_2 n - 1$ and span $(G) \neq$ span (D). [N.B. G and D both have size $n/2$ and have complementary spans.]

(iv) If layer $(D) = 2\log_2 n - i$ and $0 \leqq i < \log_2 n - 1$, then D is the middle 2^i bits of the 3.2^i length value of merge (E, F) where layer $(E) = 2\log_2 n - i - 1$, layer $(F) = i$ and span $(E) =$ span $(D) \cup$ span (F).

(v) If layer $(D) = 2\log_2 n$ and $D = \{y_j\}$ then y_j is the output corresponding to x_j $(1 \leqq j \leqq n)$.

3. Proof of correctness. It is evident that the first $\log_2 n$ layers constitute a sorting algorithm. In particular, if layer $(A) = i$ and $0 \leqq i < \log n$ then A is a sorted list for span (A). Let $\#_a(A)$ denote the number of a's in A where $a = 0$ or 1. The main point of the construction can be expressed as follows:

CLAIM. *If* layer $(D) = 2\log_2 n - i$ *and* $0 \leqq i < \log_2 n$ *then the number of zeros in* D *is* $k - \#_1(\bar{D})$.

Note that the validity of the claim for layer $2\log_2 n$ is enough to establish the theorem since in that case if $D = \{y_j\}$ then span $(\bar{D})$ is the set of all inputs other than x_j, and hence y_j will be made zero if and only if x_j is indeed one.

Proof of Claim. The proof proceeds by induction down from $i = \log_2 n - 1$ to $i = 0$.

The base case, $i = \log_2 n - 1$, corresponds to layer $\log_2 n + 1$. By part (iii) of the construction and since $\#_1(G) \leqq k$

$$\#_1(D) = \#_1(G) + (n/2 - k).$$

Since $\#_1(G) = \#_1(\bar{D})$ and $\#_0(D) = n/2 - \#_1(D)$ it follows that

$$\#_0(D) = k - \#_1(\bar{D}).$$

For the induction step assume that the claim holds for some i $(0 < i \leqq \log_2 n - 1)$ and deduce that it must hold for $i - 1$ also. Consider the result of merge (E, F) in step (iv). By the inductive assumption

$$\#_0(E) = k - \#_1(\bar{E}).$$

Since F is just a sorted list of span (F),

$$\#_0(F) = 2^i - \#_1(F).$$

Hence

$$\#_0(D) = 2^i + k - \#_1(\bar{E}) - \#_1(F) = 2^i + k - \#_1(\bar{D}).$$

Since span $(\bar{D})$ = span $(\bar{E}) \cup$ span (F), it follows that merge (E, F) consists of 2^i zeros, followed by a further $k - \#_1(\bar{D})$ zeros, followed by 1's. Since there are k 1's altogether $\#_1(\bar{D}) \geqq k - 2^i$ and hence $k - \#_1(\bar{D}) \leqq 2^i$. It follows that the middle 2^i bits of merge (E, F) have exactly $k - \#_1(\bar{D})$ zeros as desired. □

REFERENCES

[1] M. AJTAI, J. KOMLÓS AND E. SZEMERÉDI, *An $O(n \log n)$ sorting network*, Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 1-9.

[2] K. BATCHER, *Sorting networks and their applications*, AFIPS Spring Joint Computing Conference, 32, 1968, pp. 307-314.

[3] S. J. BERKOWITZ, Personal communication (1981). Also, *On some relationships between monotone and non-monotone circuit complexity*, manuscript, University of Toronto, Computer Science Department.

[4] N. BLUM, *An $\Omega(N^{4/3})$ lower bound on the monotone network complexity of N-th degree convolution*, Proc. 22nd ACM Symposium on Foundations of Computer Science, 1981, pp. 101-108.

[5] D. COPPERSMITH AND S. WINOGRAD, *On the asymptotic complexity of matrix multiplication*, Proc. 22nd ACM Symposium on Foundations of Computer Science, 1981, pp. 82-90.

[6] M. J. FISCHER, *The complexity of negation-limited networks*, Lecture Notes in Computer Science 33, Springer-Verlag, Berlin, 1974, pp. 71-82.

[7] E. A. LAMAGNA AND J. E. SAVAGE, *Combinational complexity of some monotone functions*, Proc. 15th IEEE Symposium on Switching and Automata Theory, 1974, pp. 140-144.

[8] K. MEHLHORN AND Z. GALIL, *Monotone switching circuits and Boolean matrix product*, Computing, 16 (1976), pp. 99-111.

[9] M. S. PATERSON, *Complexity of monotone networks for Boolean matrix product*, Theoret. Comput. Sci., 1 (1975), pp. 13-20.

[10] N. J. PIPPENGER AND L. G. VALIANT, *Shifting graphs and their applications*, J. Assoc. Comput. Mach., (1976), pp. 423-433.

[11] V. R. PRATT, *The power of negative thinking in multiplying Boolean matrices*, this Journal, 4 (1975), pp. 326-330.

[12] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354-356.

[13] I. WEGENER, *Boolean functions whose monotone complexity is of size $n^2/\log n$*, Theoret. Comput. Sci., 21 (1982), pp. 213-224.

[14] ———, *On the complexity of slice functions*, Lecture Notes in Computer Science 176, Springer-Verlag, Berlin, 1984, pp. 553-561. (Also Theoret. Comput. Sci., to appear.)

[15] J. WEISS, *An $\Omega(n^{3/2})$ lower bound on the monotone complexity of Boolean convolution*, Inform. and Control, to appear.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
016

ON THE LAGARIAS–ODLYZKO ALGORITHM FOR THE SUBSET SUM PROBLEM*

A. M. FRIEZE†

Abstract. We give a simple analysis of an algorithm for solving subset-sum problems proposed by Lagarias and Odlyzko [2].

Key words. complexity, lattice algorithm, random problems

Suppose $\mathbf{e}=(e_1, e_2, \cdots, e_n)\in\{0,1\}^n$, $B_1, B_2, \cdots, B_n$ are positive integers and $B_0=\sum_{i=1}^n B_i e_i$. Then clearly $\mathbf{e}$ is a solution of

$$\sum_{i=1}^{n} B_i x_i = B_0, \qquad x_i = 0 \text{ or } 1, \quad i=1,2,\cdots,n. \tag{1}$$

The following problem arises in cryptography [4]: given $B_0, B_1, \cdots, B_n$, find $\mathbf{e}$ by solving (1).

Solving (1) is a well-known NP-complete problem and Lagarias and Odlyzko [2] describe an algorithm which almost surely[1] finds $\mathbf{e}$ assuming

(2) $B_1, B_2, \cdots, B_n$ are independently chosen at random from $1, \cdots, B=2^{cn^2}$, c sufficiently large.

In this paper we show that $c=\frac{1}{2}+\varepsilon$, $\varepsilon>0$ is sufficient. The main point of this paper is to give a simple proof of their result.

In the following analysis $\mathbf{e}$ is *fixed* and $B_1, B_2, \cdots, B_n$ are randomly generated. We note that we can assume

$$B_0 \geqq \sum_{i=1}^{n} B_i/2 \tag{3}$$

for if not, we can put $y_i=1-x_i$ and try to solve

$$\sum_{i=1}^{n} B_i y_i = \sum_{i=1}^{n} B_i - B_0, \qquad y_i = 0 \text{ or } 1, \quad i=1,2,\cdots,n. \tag{4}$$

Now let $p=\lceil n2^{n/2}\rceil$, Z be the set of integers and

$$\begin{aligned}
\mathbf{b}_0 &= (pB_0, 0, \cdots, 0)\in Z^{n+1},\\
\mathbf{b}_1 &= (-pB_1, 1, 0, \cdots, 0),\\
&\vdots\\
\mathbf{b}_n &= (-pB_n, 0, 0, \cdots, 1).
\end{aligned}$$

Let $L=\{\mathbf{z}=\sum_{i=0}^n \xi_i \mathbf{b}_i : \xi_i\in Z, i=0,1,\cdots,n\}$ be the lattice generated by $\mathbf{b}_0, \mathbf{b}_1, \cdots, \mathbf{b}_n$.

Let $\hat{\mathbf{e}}=(0, e_1, e_2, \cdots, e_n)=\mathbf{b}_0+\sum_{i=1}^n e_i\mathbf{b}_i\in L$. Note that $\|\hat{\mathbf{e}}\|\leqq n^{1/2}$, using the euclidean norm. Thus $\hat{\mathbf{e}}$ is a "short" vector of L.

* Received by the editors April 24, 1985, and in revised form February 15, 1985.

† Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213. Current address, Department of Computer Science and Statistics, Queen Mary College, London E1 4NS, England.

[1] By *almost surely* (a.s.) we mean with probability tending to 1.

Let $\|\mathbf{x}^*\| = \min(\|\mathbf{x}\| : \mathbf{x} \neq 0, x \in L)$. It is not known at present whether it is possible to find a shortest nonzero vector in L, in polynomial time. However, using the Basis Reduction Algorithm (BRA) of Lenstra, Lenstra and Lovász [3], we can in polynomial time find $\hat{\mathbf{x}} \in L$, $\hat{\mathbf{x}} \neq 0$ satisfying

$$\|\hat{\mathbf{x}}\| \leqq 2^{n/2}\|\mathbf{x}^*\| \leqq 2^{n/2}\|\hat{\mathbf{e}}\| \leqq m = 2^{n/2}n^{1/2}. \tag{5}$$

Thus we can try to solve (1) by applying BRA to L and seeing if it produces $\pm\hat{\mathbf{e}}$. There is of course the possibility that there is more than one solution to (1); however the analysis below shows this to be unlikely.

So let $\hat{\mathbf{x}}$ be the shortest vector produced by BRA and assume that $B_1, B_2, \cdots, B_n$ are distributed as in (2). We will show

$$\Pr(\hat{\mathbf{x}} \neq \pm\hat{\mathbf{e}}) \leqq (4m+1)(2m+1)^n/B = O(2^{-\varepsilon n^2/2}) \quad \text{if } B \geqq 2^{(1/2+\varepsilon)n^2}. \tag{6}$$

If $\mathbf{x} = (x_0, x_1, \cdots, x_n) \in L$, then we have

$$\mathbf{x} = x_0'\mathbf{b}_0 + x_1\mathbf{b}_1 + \cdots + x_n\mathbf{b}_n \quad \text{where } x_0 = p\left(B_0x_0' - \sum_{i=1}^{n} B_ix_i\right).$$

Let $L_0 = \{\mathbf{x} \in L: x_0 = 0\}$. It follows that

$$\mathbf{x} \in L - L_0 \text{ implies } \|\mathbf{x}\| \geqq p. \tag{7}$$

Thus (5) and (7) imply that $\hat{\mathbf{x}} \in L_0$. The lattice used in [2] has $p = 1$. Taking p large allows us to restrict our attention to L_0. It also allows us to solve one lattice problem in place of the two solved in [2]. We can prove (6) by showing

$$\Pr(A_0 \neq \varnothing) \leqq (4m+1)(2m+1)^n/B \tag{8}$$

where $A_0 = \{\mathbf{x} \in L_0: \|\mathbf{x}\| \leqq m, \mathbf{x} \neq k\hat{\mathbf{e}} \text{ for any } k \in Z\}$. (Note that $\hat{\mathbf{x}} = k\hat{\mathbf{e}}$ for $k \in Z$ implies $k = \pm 1$ if $\hat{\mathbf{x}}$ is part of a basis.)

But if $\mathbf{x} \in A_0$ then

$$|B_0\mathbf{x}_0'| = \left|\sum_{i=1}^{n} B_ix_i\right| \leqq \sum_{i=1}^{n} B_i\|x\| \tag{9}$$

and so $|\mathbf{x}_0'| \leqq 2\|x\| \leqq 2m$, using (3). So if $A_0 \neq \varnothing$ there exist $\mathbf{x} = (x_1, x_2, \cdots, x_n) \in Z^n$ and $y \in Z$ satisfying

$$\|\mathbf{x}\| \leqq m, \qquad |y| \leqq 2m, \tag{10a}$$

$$\mathbf{x} \neq k\mathbf{e} \quad \text{for any } k \in Z, \tag{10b}$$

$$\sum_{i=1}^{n} B_ix_i = yB_0. \tag{10c}$$

Consider now a *fixed* $\mathbf{x}$, y satisfying (10a) and (10b) and let $A_1 = \{\mathbf{x} \in Z^n: \|\mathbf{x}\| \leqq m\}$. We will prove that

$$\Pr(\mathbf{x}, y \text{ satisfy (10c)}) \leqq 1/B \tag{11}$$

and then

$$\Pr(\exists \mathbf{x}, y \text{ satisfying (10)}) \leqq (4m+1)|A_1|/B \leqq (4m+1)(2m+1)^n/B$$

and (8) follows.

To prove (11), note that (10c) is equivalent to $\sum_{i=1}^{n} B_iz_i = 0$ where $z_i = x_i - ye_i$. Since (10b) holds, we can assume, without loss of generality, that $z_1 \neq 0$. Letting ξ

denote $-(\sum_{i=2}^{n} B_i z_i / z_1)$,

$$\Pr\left(\sum_{i=1}^{n} B_i z_i = 0\right) = \Pr(B_1 = \xi) = \sum_{j=1}^{B} \Pr(B_1 = j | \xi = j) \Pr(\xi = j)$$

$$= \sum_{j=1}^{B} \frac{1}{B} \Pr(\xi = j) \quad \text{as } B_1 \text{ and } \xi \text{ are independent}$$

$$\leqq \frac{1}{B}.$$

This completes the proof of the main result.

Schnorr [5] has recently built on the ideas in [3] and Kannan [1] to construct a family of basis reduction algorithms, so that for any $\sigma > 1$ there is an algorithm BRA_σ in the family which runs in polynomial time (the degree of the polynomial depends on σ) which guaranteed to find a vector of length no more than $\sigma^{n-1}\|\mathbf{x}^*\|$. Using BRA_σ in place of BRA means that we can take $c = \sigma + \varepsilon$ in (2) and still a.s. solve the problem.

Now Lagarias and Odlyzko also show that if $B = 2^{cn}$, where $c > c_0 = 1.54725$, then

(12) $\hat{\mathbf{e}}$ is a.s. the shortest vector of L.

It is not difficult to see first that $B = 2^{cn}$ gives (12) for some $c > 0$ assuming we proceed exactly as above. Let $m = n^{1/2}$ and $\mathbf{x}^*$ be the shortest vector of L. If $\mathbf{x}^* \neq \pm\hat{\mathbf{e}}$ then (10) again holds. It is easy to show that $|A_1| \leqq 2^{cn}$ for some $c > 0$ and this c will suffice.

To get c as small as c_0, we have to assume that $\sum_{i=1}^{n} e_i \leqq n/2$. This is true for one of the problems (1) and (4) and so, as in [2], we solve *both* of these. We can now take $m = (n/2)^{1/2}$ in our analysis.

We cannot assume (3) for the problem in which $\sum_{i=1}^{n} e_i \leqq n/2$ but as $B_0 \geqq \min\{B_i : i = 1, 2, \cdots, n\} \geqq B/n^2$ a.s. we can assume this instead. Using this in (9) gives $|x_0| \leqq n^2 m$ and so we take $|y| \leqq n^2 m$ in (10a). Theorem 3.2 of [2] is that $|A_1| \leqq 2^{c_0 n}$ and so (12) holds as

$$\Pr((12) \text{ fails}) \leqq \Pr((10) \text{ holds}) + \Pr(B_0 < B/n^2).$$

(i) *Problems with $r > 1$ constraints.* Here one replaces c by c/r in the theorems. By multiplying the ith constraint by B^{i-1} and then adding all these constraints together we have a subset sum problem in which the coefficients are very close to being randomly chosen uniformly from $1, \cdots, B^r$.

(ii) *B_0 an independent random variable.* Suppose that instead of $\mathbf{e}$ being an a priori solution, B_0 is randomly generated in $1, \cdots, \lceil \lambda n B \rceil$ where $0 < \lambda \leqq 1$ is some constant. It is not difficult to show for $B = 2^{cn^2}$, $c > \frac{1}{2}$, that if (1) has a solution then it is a.s. unique and this approach a.s. finds it.

Acknowledgment. I am grateful to Ravi Kannan for interesting discussions on this topic.

REFERENCES

[1] R. KANNAN, *Improved algorithms for integer programming and related problems*, in Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983.

[2] J. C. LAGARIAS AND A. M. ODLYZKO, *Solving low density subset sum problems*, Proc. 25th Annual IEEE Symposium on Foundations of Computer Science, 1983, pp. 1-10.

[3] A. K. LENSTRA, H. W. LENSTRA, JR. AND L. LOVÁSZ, *Factoring polynomials with rational coefficients*, Math. Ann., 26 (1982), pp. 515–534.

[4] R. C. MERKLE AND M. E. HELLMAN, *Hiding information and signatures in trap-door knapsacks*, IEEE Trans. Inform. Theory, IT-24 (1978), pp. 525–530.

[5] C. P. SHNORR, *A hierarchy of polynomial time basis reduction algorithms*, Proc. Symposium on the Theory of Algorithms, Pe'cs, Hungary, 1984, to appear.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
017

CONSTANT TIME GENERATION OF FREE TREES*

ROBERT ALAN WRIGHT†, BRUCE RICHMOND‡, ANDREW ODLYZKO§
AND BRENDAN D. MCKAY¶

Abstract. An algorithm of Beyer and Hedetniemi [SIAM J. Comput., 9 (1980), pp. 706–712] for generating rooted unlabeled trees is extended to generate unlabeled free trees. All the nonisomorphic trees of a given size are generated, without repetition, in time proportional to the number of trees.

Key words. free tree, unrooted tree, nonisomorphic trees, constant time generation, constructive enumeration, lexicographic order, loop-free algorithm

1. Introduction. In [1], Beyer and Hedetniemi exhibit an algorithm for generating all rooted trees of a given size. The method uses a successor function to traverse an ordered set of integer sequences which represents the objects being generated. This method is based on one introduced by Ruskey and Hu [7]. In this paper the technique of Beyer and Hedetniemi is refined to produce only one member of each equivalence class of rooted trees under isomorphism of the underlying free (unrooted) trees.

Previous algorithms for generating these trees have been given by Read [6], Dinits and Zaitsev [2] and Kozina [3]. Our algorithm has an advantage over each of these, in that it only requires $O(n)$ space and constant average time per tree (independently of n). An algorithm for the corresponding random generation problem has been given by Wilf [8].

2. Representing trees by level sequences. The notation (T, z) is used here to denote the rooted tree with underlying free tree T and root vertex z. The *level* of a vertex v in a rooted tree (T, z) is one more than the distance from the vertex to the root. The root z is assigned level value 1. A *level sequence* is defined as a sequence of integers produced by listing the level of each vertex of a rooted tree in preorder. Since a preorder traversal may visit the subtrees at a given vertex in various orders, level sequences for a rooted tree are, in general, not unique. The notation $L(T, z) = [l_1, l_2, \cdots, l_n]$ will be used for any level sequence of a rooted tree (T, z) on n vertices.

In order to have a unique level sequence representation for a given rooted tree, the rules for the preorder traversal must be refined slightly. For a level sequence to be *canonical*, the traversal must visit the roots of adjacent subtrees in nonincreasing lexicographic order of the canonical level sequences of those subtrees. A simpler (and equivalent) formulation of this canonicity criterion is the requirement that the canonical sequence for a given rooted tree be the sequence which is the lexicographically greatest of all level sequences describing that same tree. A more complete discussion of canonicity (with proofs and examples) may be found in [1]. The canonical level sequence of a rooted tree (T, z) will be denoted by $L^*(T, z)$.

3. Generating all canonical level sequences of given length. Beyer and Hedetniemi have described in [1] an algorithm which generates all canonical level sequences of

* Received by the editors March 4, 1983, and in revised form February 6, 1985.
† 1948d Adams Avenue, Costa Mesa, California 92626.
‡ Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.
§ AT&T Bell Laboratories, Murray Hill, New Jersey 07974.
¶ Computer Science Department, Australian National University, Canberra ACT 2601, Australia.

given length in lexicographic order. Their method involves an iterative algorithm which has as its basis a *successor function*, which when given any canonical level sequence, will generate the next canonical sequence, with respect to lexicographic order. The precise definition is this: Let $L = L^*(T, z)$ be a canonical level sequence of length n. Let p be the largest integer such that $l_p \neq 2$ and let q be the largest integer such that $q < p$ and $l_q = l_p - 1$. The successor $s(L) = [s_1, s_2, \cdots, s_n]$ of L then is given by:

$$s_i = \begin{cases} l_i, & \text{for } 1 \leqq i < p, \\ s_{i-(p-q)}, & \text{for } p \leqq i \leqq n. \end{cases}$$

In [1] it is proved that this function transforms any canonical level sequence other than $[1, 2, 2, \cdots, 2]$ into the next canonical level sequence, in decreasing lexicographic order.

We now will extend these results by deriving an algorithm which generates a subset of the canonical level sequences corresponding to the set of free trees on a given number of vertices.

4. Extracting a nonisomorphic subset of rooted trees by root selection. The objective here is to place requirements on the root vertex, which may only be satisfied by one rooted tree with a given underlying tree. First we will require that the root be an element of the *center* of the tree, which is defined to be the set of vertices whose maximum distance from the other vertices is least. Since all trees have either one vertex or two adjacent vertices in the center, we now need only refine this rule for the bicentral case. Let T be a bicentral tree, and consider the subtrees T_1 and T_2 remaining when the edge joining the two candidate roots, z_1 and z_2, is deleted. Two rooted trees, (T_1, z_1) and (T_2, z_2) are formed in this way. Either these two trees are isomorphic, in which case the rooted trees (T, z_1) and (T, z_2) are isomorphic, or they can be distinguished by their size, or by the precedence of their canonical level sequences (when they are the same size). The root selected for T will be z_1 if T_1 has fewer vertices than T_2 or if they have the same order and $L^*(T_1, z_1)$ is lexicographically less than $L^*(T_2, z_2)$. Otherwise we select z_2.

We will call the root uniquely selected by the above criteria the *primary root* of the tree T, and denote it by $\bar{z}(T)$, or just $\bar{z}$. Then for any given tree T of size n, there is exactly one member of the set of all rooted trees on n vertices with root meeting the above criteria, namely $(T, \bar{z})$. A level sequence $L(T, \bar{z})$ will be called a *primary level sequence* of T, and $L^*(T, \bar{z})$ will be called the *primary canonical level sequence* of T.

With the above criteria, a maximal set of nonisomorphic free trees may be extracted from the set of all rooted trees of a given size by choosing only trees whose roots are primary. But we need to apply these criteria not to the trees per se, but to their canonical level sequences.

5. Refining the canonicity criteria to obtain only primary level sequences. We will now translate the previously established rules for primary root selection to conditions sufficient for a canonical level sequence to be primary. Let (T, z) be the tree in question, and let $L = L^*(T, z)$. The first requirement was that z be in the center of the tree. We will need to use the fact that z is in the center of T if and only if it is in the center of every path of maximum length in T. The position of z in such a path can be readily checked for a canonical level sequence. To do this, consider the structure of L when viewed as the level number of z (namely 1) concatenated with the level sequences of each of the components remaining when z is deleted from T. These component level sequences, which we will call the *principal subsequences* of L, begin with level 2, instead

of 1, but otherwise are canonical level sequences themselves, due to the recursive definition of canonicity. These subsequences are, moreover, ordered in L by height, so the position of z within one path of maximum length can be determined by looking at the height of the first two principal subsequences, which we will call S_1 and S_2. This is given precisely by the highest level numbers in those respective subsequences. We will identify the position of the first occurrence of the highest level number in the ith principal subsequence by the subscript h_i. The height of S_1 then is $l_{h_1}-2$, and the height of S_2 (if it exists) is $l_{h_2}-2$. We then conclude that the maximum length of a path in T is $l_{h_1}+l_{h_2}-2$, and z is in the center of T if and only if h_2 exists and $l_{h_1}-l_{h_2}$ is either 0 or 1. In addition, if $l_{h_1}-l_{h_2}=0$, then T is unicentral, and hence L is known to be primary without further checking.

In the bicentral case, we must be able to compare $L_1=L(T_1, z_1)$ and $L_2=L(T_2, z_2)$. The two central vertices are those which are represented by level numbers l_1 and l_2, so if we let $z_2=z$, and z_1 be the vertex corresponding to l_2, then L_1 is the same as S_1 (though level numbers will be greater by 1), and L_2 is the same as L with L_1 removed. Size comparison of L_1 and L_2 can be accomplished trivially if we know the position of the start of the second principal subsequence: if the level number at this position is denoted l_m, then the size of L_1 is $m-2$, and that of L_2 is $n-m+2$ (where n is the length of L). Finally, by checking lexicographic precedence of $L_1=[l_2-1, l_3-1, \cdots, l_{m-1}-1]$ and $L_2=[l_1, l_m, l_{m+1}, \cdots, l_n]$, we have that L is primary for T if and only if L_1 is identical to L_2, or L_1 has lower precedence than L_2. This can be summarized as follows:

Let (T, z) be a rooted tree with n vertices and a canonical level sequence $L=[l_1, l_2, \cdots, l_n]$. Then using the definitions of h_1, h_2, m, L_1 and L_2 given above, we have that L is the primary canonical level sequence of T if and only if all of the following hold for L:

(i) h_2 (and hence m) exists,

(ii) $l_{h_2} \geqq l_{h_1}-1$,

(iii) if equality holds in (ii), then $m-2 \leqq n-m+2$,

(iv) if equality holds in (iii), then either $L_1=L_2$, or L_1 is shorter than L_2, or L_1 has the same length as L_2 but precedes L_2 lexicographically.

Now we are ready to derive an algorithm for generating all such primary canonical level sequences for a given value of n.

6. Generating all primary canonical level sequence of a given size. We wish to derive a successor function which, if given an appropriate starting sequence, will efficiently generate all primary canonical level sequences of the same size. It is not obvious at first that such a successor function follows naturally from the one defined by Beyer and Hedetniemi. However, due to certain choices made in the definition of the primary canonical level sequences, the previously defined function will usually yield a primary canonical level sequence when it transforms a sequence which is primary and canonical. Thus our goal becomes to detect the cases where the s function will fail and to take alternate action for those cases. As it turns out, the alternate action is trivial, although it has a significant effect on the length of the algorithm. To determine the "failure cases" for s, we will examine what the input sequence must look like in order to cause s to produce a sequence violating one of the conditions for primary sequences.

The first case we will look at is the case where s transforms a primary canonical sequence L into a sequence $s(L)$ for which the condition (ii) is violated. (Note that condition (i) can never be violated in this way). In this case, the value of l_{h_2} is changed from $l_{h_1}-1$ to $l_{h_1}-2$, which means that the p value for L is h_2, and $l_i=2$ for $p<i \leqq n$. Hence this type of failure will occur if and only if $l_{h_2}=l_{h_1}-1$ and $p=h_2$, with the one

exception being $L=[1, 2, 3, 2, 2, \cdots, 2]$, which is not a problem since in this case $s(L)$ is the last sequence to be generated.

The second case where s will fail is when $s(L)$ violates rule (iii) but not rule (ii). This occurs when L_1 is larger than L_2 in the converted sequence. For this to happen, l_{h_1} must be equal to l_{h_2}, and p must be equal to h_2 (since l_{h_2} must change). In addition, we must have $m-2>n-m+2$ for L_1 to end up larger than L_2. It should also be clear that these conditions are sufficient for a failure to occur as well.

The final kind of failure with which we must deal occurs when condition (iv) alone is violated. But this happens precisely when $L_1=L_2$, since the precedence of L_2 is always diminished when L_1 remains unchanged (and when L_1 does change, this failure cannot occur). Hence this failure condition can be detected by comparison of L_1 and L_2.

We are left with the question of what to do when we encounter one of these failure cases. Once again, the definition of the primary canonical sequences has been so chosen as to make this easy. Very simply, to get the next primary canonical sequence from a sequence L which satisfies one of the failure conditions described above, we first set p to $m-1$, apply s to get $s(L)$, and if $l_{m-1}>3$ (with the old value of m), we replace the final h_1-1 elements of $s(L)$ with $2, 3, \cdots, h_1$. The reason for this is straightforward: l_{m-1} must change (i.e. L_1 must change), since changes beyond $m-1$ will only result in L_2 having lower precedence, which can never result in all of conditions (i)-(iv) being satisfied again. Now, when L_1 does change, there are two cases to consider. Either l_{m-1} was a 3, in which case the action of s on L will result in copies of L_1 being made starting at l_m and repeating through l_n, or $l_{m-1}>3$, in which case no L_2 sequence will occur in $s(L)$. In the former case, $s(L)$ is primary, and so we are done, but in the latter case, we must correct the fact that L_2 has been eliminated. By replacing the last h_1-1 level numbers with $2, 3, \cdots, h_1$, we replace the fewest number of level numbers that we can (L_2 must have the same height as L_1), and the replacement is the highest sequence of its length which retains canonicity. Thus we necessarily have the primary canonical sequence of highest precedence which has lower precedence than the value of L which we started with.

7. Generating trees in lexicographic order. In order to be able to check the failure conditions described in the previous section, and to do so without increasing the complexity of the succession algorithm, the values of h_1, h_2, p, etc., are to be maintained. In addition, an index c to the first element of L_2 which is not the same as the corresponding element of L_1 must be kept, so that it will be apparent when $L_1=L_2$ (to facilitate detection of the third kind of failure condition described above). Lastly, a sequence $W=[w_1, w_2, \cdots, w_n]$ will be kept such that w_i is the subscript of the level number in L corresponding to the parent of the vertex corresponding to l_i in the tree represented by L.

The procedure below will accept any primary canonical level sequence other than $[1, 2, 2, \cdots, 2]$ and produce the next primary canonical level sequence in canonical order. The parameters are as we have defined except for r, which is one less than m. The value of c is occasionally set to ∞ when it will not be needed at the next iteration.

The first primary canonical level sequence is that of a path rooted at its center. To find its parameters (for $n\geqq 4$), let $k=\lfloor n/2\rfloor+1$. Then $L=[1, 2, \cdots, k, 2, 3, \cdots, n-k+1]$, $W=[0, 1, \cdots, k-1, 1, k+1, \cdots, n-1]$, $p=n$ (except that $p=3$ when $n=4$), $q=n-1$, $h_1=k$, $h_2=n$ and $r=k$. Correct operation is assured if c is initialised to ∞ for odd n and $n+1$ for even n. The last tree has been generated when the procedure returns with $q=0$.

```
procedure nexttree (L, W, n, p, q, h1, h2, c, r)
  fixit ← false
  if c = n+1 or p = h2 and (l_{h1} = l_{h2}+1 and n − h2 > r − h1 or
                l_{h1} = l_{h2} and n − h2 + 1 < r − h1) then
    if l_r > 3 then
      p ← r; q ← w_r
      if h1 = r then h1 ← h1 − 1 endif
      fixit ← true
    else
      p ← r; r ← r − 1; q ← 2
    endif
  endif
  needr ← false; needc ← false; needh2 ← false
  if p ≦ h1 then h1 ← p − 1 endif
  if p ≦ r then needr ← true
  elseif p ≦ h2 then needh2 ← true
  elseif l_{h2} = l_{h1} − 1 and n − h2 = r − h1 then
      if p ≦ c then needc ← true endif
  else c ← ∞
  endif
  oldp ← p; δ ← q − p; oldlq ← l_q; oldwq ← w_q; p ← ∞
  for i ← oldp to n do
    l_i ← l_{i+δ}
    if l_i = 2 then w_i ← 1
    else
      p ← i
      if l_i = oldlq then q ← oldwq
      else q ← w_{i+δ} − δ
      endif
      w_i ← q
    endif
    if needr and l_i = 2 then
      needr ← false; needh2 ← true; r ← i − 1
    endif
    if needh2 and l_i ≦ l_{i−1} and i > r+1 then
      needh2 ← false; h2 ← i − 1
      if l_{h2} = l_{h1} − 1 and n − h2 = r − h1 then needc ← true
      else c ← ∞
      endif
    endif
    if needc then
      if l_i ≠ l_{h1−h2+i} − 1 then needc ← false; c ← i
      else c ← i + 1
      endif
    endif
  endfor
  if fixit then
    r ← n − h1 + 1
    for i ← r+1 to n do
      l_i ← i − r + 1; w_i ← i − 1
    endfor
```

```
    w_{r+1} ← 1; h_2 ← n; p ← n; q ← p − 1; c ← ∞
  else
    if p = ∞ then
      if l_{oldp−1} ≠ 2 then p ← oldp − 1
      else p ← oldp − 2
      endif
      q ← w_p
    endif
    if needh2 then
      h_2 ← n
      if l_{h_2} = l_{h_1} − 1 and h_1 = r then c ← n + 1
      else c ← ∞
      endif
    endif
  endif
endprocedure.
```

8. Proof of the constant time property. Let t_n and T_n denote the number of free and rooted unlabeled trees of order n, respectively. A fundamental tool in our analysis is the following result of Pólya [5] and Otter [4].

THEOREM 1. *$T_n \sim C_1 n^{-3/2}\rho^{-n}$ and $t_n \sim C_2 n^{-5/2}\rho^{-n}$ as $n \to \infty$, where $C_1 \approx 0.4399$, $C_2 \approx 0.5349$ and $\rho \approx 0.3383$.*

To begin, we must establish that the total number of steps taken to do all simple successions does not exceed $O(t_n)$. (That is, we are not yet counting the steps used in recovering from any of the three failure conditions which can be encountered.) The number of steps in each of these conversions is of the order of $n-p+1$, i.e. the number of elements of the sequence which change. Hence we must sum all of the $n-p+1$ values for arbitrary n. Since the number $n-p$ is just the number of leaves adjacent to the root, if we denote the number of trees of size n with exactly k root-adjacent leaves by $t_{n,k}$, the desired sum is then $\sum_{k=0}^{n-1}(k+1)t_{n,k}$. Observing that $t_{n,k} \leqq t_{n-k}$, we then have that the above sum is no more than $\sum_{k=0}^{n-1}(k+1)t_{n-k} \leqq 4t_n$.

We now need only count the steps used in correcting failure conditions. To do this, we note that the number of steps needed to correct a single instance of a failure condition is no more than $O(n)$, and show that the number of failure cases becomes so small compared to t_n for large n that they are not significant. We will handle the three kinds of failure conditions separately.

The third kind of failure can be dispensed with easily, since it occurs only when $L_1 = L_2$, which means that the tree is symmetric about a central edge, and thus there are at most $T_{n/2}$ instances of this failure for given n. By Theorem 1, $T_{n/2}/t_n \to 0$ exponentially fast, which finishes this case.

For the first kind of failure condition, the starting sequence must have the form

$$L = [1, 2, 3, \cdots, l_{h_1}, \cdots, l_m = 2, 3, \cdots, l_{h_2} = l_{h_1} - 1, 2, 2, \cdots, 2].$$

In addition, $h_1 < m < a$, where $a = \lfloor (n+1)/2 \rfloor$, and so clearly the sum of all such cases for given n does not exceed T_a, which behaves the same as the sum did for the third kind of failure condition.

The second kind of failure condition occurs with greatest frequency. For this kind of failure to occur, the starting sequence must have the form

$$L = [1, 2, \cdots, l_{h_1}, \cdots, l_m = 2, 3, \cdots, l_{h_2} = l_{h_1}, 2, 2, \cdots, 2].$$

Also, $3 \leqq h_1 \leqq n - m - 2$ for $n/2 \leqq m \leqq n-3$ so if we denote the number of rooted trees

of height h and size n by T_n^h, and the number of rooted trees of height no more than h and size n by S_n^h, the number of sequences of the above form is no more than

$$\sum_{m=n/2}^{n-3} \sum_{h_1=2}^{n-m-1} T_m^{h_1-1} < \sum_{k=1}^{n/2} S_{n-k}^k.$$

The latter sum can be divided into two sums

$$\sum_{k=1}^{n/2} S_{n-k}^k = \sum_{k=1}^{\lfloor 3 \log n \rfloor} S_{n-k}^k + \sum_{k=\lfloor 3 \log n \rfloor + 1}^{n/2} S_{n-k}^k.$$

The second sum is much less than $nT_{n-\lfloor 3 \log n \rfloor}$, which is $o(n^{-1}t_n)$, by Theorem 1. To handle the remaining sum, we just need the following.

THEOREM 2. *There is a constant $\delta > 0$ such that uniformly for $1 \leqq h \leqq n$ we have*

$$S_h^n = O(T_n n^{3/2} \exp(-\delta n/h^2)).$$

The proof of Theorem 2 will follow from several auxiliary results. We first prove the known fact that $T(\rho) = 1$. We define

$$J(x, y) = x \exp\left(y + \sum_{k=2}^{\infty} k^{-1} T(x^k)\right) - y.$$

The functional equation satisfied by $T(x)$ states that $J(x, T(x)) = O$. Since $T(x)$ has a singularity at $x = \rho$, and $J(x, y)$ is analytic in both x and y separately in a neighborhood of $(\rho, T(\rho))$, it follows that we must have

$$\left.\frac{\partial J(x, y)}{\partial y}\right|_{\substack{x=p \\ y=T(\rho)}} = 0,$$

which says that

$$1 = \rho \exp\left(T(\rho) + \sum_{k=2}^{\infty} k^{-1} T(\rho^k)\right) = T(\rho).$$

LEMMA 1. *Define $x_h > 0$ by $S_h(x_h) = 1$. Then there exists a constant C such that for all $h \geqq 1$,*

$$S_h'(x_h) \leqq Ch.$$

Proof. Since $\frac{1}{2} = x_1 > x_2 > \cdots$, we see that $x_h \leqq 3\rho/2$ for all $h \geqq 1$. Let

$$C = \max\left(4, \rho^{-1} + \sum_{k=2}^{\infty} (3\rho/2)^{k-1} T'((3\rho/2)^k)\right).$$

Since $3\rho/2 < 0.55$, the series above converges. Since $S_1'(x_1) = 4$, the lemma clearly holds for $h = 1$. Suppose that the lemma holds for h. Then

$$S_{h+1}'(x) = \left(1 + x \sum_{k=1}^{\infty} S_h'(x^k) x^{k-1}\right) \exp\left(\sum_{k=1}^{\infty} k^{-1} S_h(x^k)\right)$$

$$= \left(\frac{1}{x} + \sum_{k=1}^{\infty} S_h'(x^k) x^{k-1}\right) S_{h+1}(x).$$

Hence, using $S_{h+1}(x_{h+1}) = T(\rho) = 1$, we find

$$S_{h+1}'(x_{h+1}) \leqq \rho^{-1} + \sum_{k=2}^{\infty} (3\rho/2)^{k-1} T'((3\rho/2)^k) + S_h'(x_{h+1}) \leqq C + Ch = C(h+1),$$

which proves the lemma by induction.

LEMMA 2. *There exists a constant K such that for all $h \geqq 1$,*

$$1 - S_h(\rho) \geqq K/h.$$

Proof. Clearly the lemma is true for $1 \leqq h \leqq 2$ and some $K < \frac{1}{2}$. Suppose the lemma holds for $h \leqq H-1$, $H \geqq 3$. Now

$$\begin{aligned} S_H(\rho) &= \rho \exp\left(S_{H-1}(\rho) + \sum_{k=2}^{\infty} k^{-1} S_{H-1}(\rho^k) \right) \\ &= \rho \exp\left(T(\rho) + \sum_{k=2}^{\infty} k^{-1} T(\rho^k) + S_{H-1}(\rho) - T(\rho) + \sum_{k=2}^{\infty} k^{-1}(S_{H-1}(\rho^k) - T(\rho^k)) \right) \\ &= T(\rho) \exp\left(S_{H-1}(\rho) - T(\rho) + \sum_{k=2}^{\infty} k^{-1}(S_{H-1}(\rho^k) - T(\rho^k)) \right) \\ &= \exp\left(S_{H-1}(\rho) - 1 + \sum_{k=2}^{\infty} k^{-1}(S_{H-1}(\rho^k) - T(\rho^k)) \right). \end{aligned}$$

Now, since $S_{H-1}(\rho^k) \leqq T(\rho^k)$, the induction hypothesis yields

$$\begin{aligned} S_H(\rho) &\leqq \exp\left(-\frac{K}{H-1} \right) \leqq 1 - \frac{K}{H-1} + \frac{K^2}{2(H-1)^2} \\ &= 1 - \frac{K}{H} + K\left(\frac{1}{H} - \frac{1}{H-1} \right) + \frac{K^2}{2(H-1)^2} \\ &= 1 - \frac{K}{H} + K\left(\frac{K}{2(H-1)^2} - \frac{1}{H(H-1)} \right) < 1 - \frac{K}{H}. \end{aligned}$$

Thus the lemma follows by induction for all h.

We can now prove Theorem 2. From Lemma 2,

$$S_h'(x_h)(x_h - \rho) \geqq S_h(x_h) - S_h(\rho) = 1 - S_h(\rho) \geqq \frac{K}{h}.$$

Now Lemma 1 gives

$$x_h - \rho \geqq \frac{K}{S_h'(x_h)h} \geqq \frac{K}{Ch^2} \geqq \frac{C_1}{h^2},$$

or

$$x_h \geqq \rho\left(1 + \frac{C_2}{h^2} \right).$$

Finally for any x, $0 < x < 1$,

$$S_h^n \leqq S_h(x) x^{-n},$$

so

$$S_h^n \leqq S_h(x_h) x_h^{-n} \leqq \rho^n (1 + C_2 h^{-2})^{-n} = O(T_n n^{3/2} \exp(-\delta n/h^2)).$$

9. Concluding remarks. The algorithm described in this paper has been implemented in BLIS10 on a DEC-System-10 (KL) at Vanderblit University. The program generates trees at the rate of 15-20 thousand trees per second, irrespective of n.

REFERENCES

[1] T. BEYER AND S. M. HEDETNIEMI, *Constant time generation of rooted trees*, this Journal, 9 (1980), pp. 706–712.

[2] E. A. DINITS AND M. A. ZAITSEV, *Algorithm for the generation of nonisomorphic trees*, Avtomat. Telem., 4 (1977), pp. 121–126; Automatic and Remote Control, 38 (1977), pp. 554–558.

[3] A. V. KOZINA, *Coding and generation of nonisomorphic trees*, Kibernetika, 5 (1979), pp. 38–43; Cybernetics, 15 (1975), pp. 645–651.

[4] R. OTTER, *The number of trees*, Ann. Math., 49 (1948), pp. 583–599.

[5] G. PÓLYA, *Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen*, Acta Math., 68 (1937), pp. 145–254.

[6] R. C. READ, *How to grow trees*, in Combinatorial Structures and their Applications, Gordon and Breach, New York, 1970.

[7] F. RUSKEY AND T. C. HU, *Generating binary trees lexicographically*, this Journal, 6 (1977), pp. 745–758.

[8] H. S. WILF, *The uniform selection of free trees*, J. Algorithms, 2 (1981), pp. 204–207.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986
© 1986 Society for Industrial and Applied Mathematics
018

BOUNDS FOR WIDTH TWO BRANCHING PROGRAMS*

ALLAN BORODIN†#, DANNY DOLEV‡#, FAITH E. FICH§# AND WOLFGANG PAUL¶

Abstract. Branching programs have been studied as a fundamental model for space bounded computations and, in particular, as a model in which to try to establish nontrivial space lower bounds and time-space trade-offs. At present, there still do not exist any results for single output functions. We consider a class of severely constrained programs (those having width 2) and establish characterizations as well as lower bounds for some Boolean functions computable within this model.

Key words. computational complexity, branching programs, lower bounds, Boolean formulae, Boolean circuits

AMS(MOS) subject classification. 68Q05

1. Introduction. *Branching programs* for the computation of Boolean functions were first studied in the Master's thesis of Masek [10]. In a rather straightforward manner they generalize the concept of a decision tree to a *decision graph.* Formally, they can be defined as labelled acyclic digraphs with the following properties.

1. There is exactly one source.
2. Every node has outdegree at most 2.
3. For every node v with outdegree 2, one of the edges leaving v is labelled by a Boolean variable x_i and the other edge is labelled by its complement $\bar{x}_i$.
4. Every sink is labelled by 0 or 1.

Let P be a branching program with edges labelled by the Boolean variables $x_1, \cdots, x_n$ and their complements. Given an input $a = (a_1, \cdots, a_n) \in \{0, 1\}^n$, program P computes a *function value* $f_P(a)$ in the following way. The computation starts at the source. If the computation has reached a node v and if only one edge leaves v, then the computation proceeds via that edge. If two edges, with labels x_i and $\bar{x}_i$, leave v, then the computation proceeds via the edge labelled x_i if $a_i = 1$, and via the edge labelled $\bar{x}_i$ otherwise. Once the computation reaches a sink, the computation ends and $f_P(a)$ is defined to be the label of that sink.

The nodes of P play the role of states or configurations. In particular, sinks play the role of final states or stopping configurations. We call sinks *accepting* if they are labelled 1 and *rejecting* otherwise.

The *length* of program P is the length of the longest path in P. Following Cobham [3], the *capacity* of the program is defined to be the logarithm to the base 2 of the number of nodes in P. Length and capacity are lower bounds on time and space requirements for any reasonable model of sequential computation. Clearly, any n-variable Boolean function can be computed by a branching program of length n *if* the capacity is not constrained. (For example, consider a complete binary tree with 2^n leaves, one for each input.) Since space lower bounds in excess of $\log n$ remain a fundamental challenge, we consider restricted branching programs in the hope of gaining insight into this problem and the closely related problem of time-space trade-offs.

* Received by the editors March 18, 1983, and in final revised form June 17, 1985.
† University of Toronto, Toronto, Ontario, Canada M5S 1A4.
‡ Hebrew University, Givat Ram, 91904 Jerusalem, Israel.
§ University of Washington, Seattle, Washington 98195.
¶ IBM Research Laboratory, San Jose, California 95193.
# This work was done while this author was a visitor at IBM, San Jose, California.

We call a digraph *levelled* if its nodes can be partitioned into levels $L_0, L_1, \cdots$ such that, for all i, an edge leaving a node in level L_i ends at a node in level L_{i+1}. The *width* of such a graph is the maximum number of nodes at any level. Every branching program can easily be transformed into a levelled program that computes the same function, has the same length, and has at most twice the capacity of the original program [2]. Therefore, if we are interested in asymptotic bounds on length and capacity, then, without loss of generality, we can assume branching programs to be levelled. In this way, the level of a node represents the time needed to reach the node starting from the source.

For any node v in a branching program P, let $I_P(v)$ be the set of inputs a such that the computation of P given a reaches v. If P is levelled, then, for each i, the system of sets $\{I_P(v): v \in L_i\}$ is a partition of the input that mirrors the knowledge (or lack of knowledge) about the input at the level L_i.

The notation $\#S$ is used to denote the cardinality of the set S. For $a \in \{0,1\}^n$, let $S(a) \subseteq \{1, \cdots, n\}$ be the set of indices i such that $a_i = 1$. The *weight* $w(a)$ of a is $\#S(a)$.

A *parity function* is a function of the form $x_1 \oplus \cdots \oplus x_n$ where $n \geqq 1$ and $x_1, \cdots, x_n$ are Boolean literals associated with distinct variables. (A Boolean literal is either a Boolean variable or the complement of a Boolean variable.) A Boolean function is a parity function if and only if it changes value when one of its arguments changes value.

The n-ary functions $E^n_{h,k}$ are defined by

$$E^n_{h,k}(a) = 1 \quad \text{if and only if} \quad h \leqq w(a) \leqq k.$$

We write E^n_k for $E^n_{k,k}$ and drop the superscript n if the number of arguments is clear from the context. $E^n_{\lceil n/2 \rceil, n}$ is called the *majority function* and $E^n_{\lceil n/2 \rceil}$ is called the *exactly-half function*. Masek [10] made two observations concerning the latter function.

1. Suppose that during any computation each input variable is examined only once. Then, for $i \leqq n/2$, level L_i must contain $i+1$ nodes. Hence, a branching program of minimum length, n, must have capacity at least $2 \log_2 n +$ a constant. This lower bound can be achieved by a branching program which counts the number of input variables that have value 1.

2. By modular counting, the capacity requirement could be reduced at the expense of increased time.

In fact, both the exactly-half function and the majority function possess algorithms which *simultaneously* achieve capacity $O(\log n)$ and length $O(n)$. However, if we severely restrict the width of the programs, we begin to observe some potentially strange behavior. This we hope gives insight into how computations become confused (and hence prolonged) if we do not allow enough states.

Independently, Furst, Saxe and Sipser [7] were led to the study of such functions in trying to establish the relativized NP-hierarchy. They proved a very nontrivial lower bound: constant depth circuits computing a parity function of n variables must be of size greater than any polynomial in n. Similar lower bounds were established for other functions (including the majority function) based on suitable reductions of parity functions to these functions. We note that Boolean $(\wedge, \vee, \neg)$-circuits with unbounded fan-in of depth d and size s can be simulated by branching programs of width $d+1$ and length s^d.

Clearly a parity function of n variables can be computed by branching programs of width 2 and length n. But what about the majority function? It has recently been shown by Chandra, Furst and Lipton [4] that the majority function cannot be computed in bounded width and linear length. We would like to show that this function or the closely related exactly-half function cannot be computed in bounded width and

polynomial length. In fact, we conjecture that, in these cases, bounded width implies exponential length. Thus far we have only been able to establish much weaker results, dealing with branching programs of width 2. Even so, we found that width 2 branching programs offer some surprises and challenges. This is unlike the situation for depth 2 circuits which are characterized by disjunctive normal form and conjunctive normal form.

Hong-Jai Wei [16] first examined the computation of the exactly-half function using restricted width 2 branching programs. In this paper, we obtain an exponential lower bound on length for computing both exactly-half and majority using a less restricted class of width 2 branching programs. We also prove an $\Omega(n^2/\log n)$ lower bound on the length of arbitrary width 2 branching programs which compute majority.

Based on the preliminary version of this work [1], some very exciting progress has been made. For a restricted class of width 2 branching programs, Plumstead and Plumstead [12] obtained an exponential lower bound on the length necessary to compute parity functions. Shearer [14] proved that any width 2 branching program that determines whether the weight of the input is divisible by 3 must have length $\Omega((\frac{3}{2})^n)$. Yao [17] has shown that the length of width 2 branching programs computing majority grows faster than any polynomial in the number of variables. Pudlák [13] showed that any branching program for the majority function requires $\Omega(n \log\log n/\log\log\log n)$ nodes and hence the same lower bound applies to the length of any constant width branching program for majority. As Pudlák notes, branching programs are a special case of contact networks and therefore Neciporuk's [11] $\Omega(n^2/\log^2 n)$ lower bound applies to the number of nodes required for the specific function given in Neciporuk's paper. However, the Neciporuk technique cannot yield nonlinear lower bounds for symmetric functions.

The *formula size* of a Boolean function is the minimum number of occurrences of literals in any Boolean formula (over the basis of all binary operations) which describes the function. Although most Boolean functions of n variables have formula size $\Omega(2^n/\log n)$, the best lower bound for specific examples is $\Omega(n^2/\log n)$ due to Neciporuk [11]. Fischer, Meyer and Paterson [6] have shown that most symmetric Boolean functions, including $E_{k,n}$ for $k, n-k=\Omega(n/\log n)$ have formula size $\Omega(n \log n)$. In fact, every symmetric Boolean function has polynomial formula size [9], [15].

We will show that lower bounds for formula size directly translate into two lower bounds for the length of bounded width branching programs. Precisely because bounded width branching programs constitute a more restrictive model, there is hope that better lower bounds can be more easily achieved.

2. Strict width 2 branching programs and their characterization. In order to understand branching programs of width 2 (henceforth called W_2-programs), it is useful to study even more restrictive models of computation. Specifically, we can require that accepting or rejecting nodes occur only at the last level of the program.

We call a width 2 branching program *monotone* if it has exactly one rejecting node. A *strict* width 2 branching program has exactly one accepting node and exactly one rejecting node. Without loss of generality, these nodes will occur at the last level.

In monotone programs no intermediate rejecting nodes are allowed. In strict programs neither intermediate accepting nodes nor intermediate rejecting nodes are allowed. Any W_2-program with t sinks can be decomposed into $t-1$ strict W_2-programs in an obvious way.

By considering disjunctive normal form, it is clear that every Boolean function is computable by a monotone W_2-program. The usual counting argument establishes the existence of functions whose branching programs have length exponential in n if the width is bounded by a polynomial in n. However, we are looking for lower bounds for effectively defined functions. (A sequence of n-ary Boolean functions f_n, $n = 1, 2, \cdots$ is effectively defined if $\bigcup_n f_n^{-1}(1) \in NP$.)

It is not a priori clear whether strict W_2-programs are powerful enough to compute every Boolean function. Let SW_2 denote the class of functions computable by strict W_2-programs. In this section we give a characterization of SW_2 and use it to show that some simple functions are not in SW_2. For instance E_1^4 is not in SW_2. It is somewhat surprising that E_1^3 and E_2^4 are in SW_2. The lower bounds which we derive later are based on the results and techniques developed here. Our characterization reveals some of the subtleties and the power of strict W_2-programs. It is not surprising that parity should play a prominent role here. We will occasionally abuse notation and identify Boolean formulas and the Boolean functions defined by the formulas.

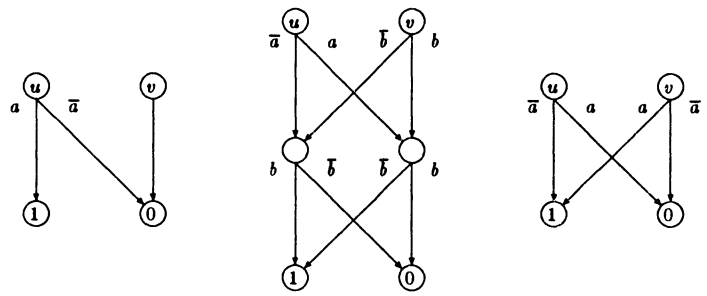
THEOREM 1. *SW_2 is the smallest class $\mathscr{C}$ of Boolean functions containing the constant functions 0 and 1 which has the following closure properties: if a and b are literals or constants and $f \in \mathscr{C}$ then*

(R1) $f \wedge a \in \mathscr{C}$.

(R2) $f \wedge (a \oplus b) \in \mathscr{C}$.

(R3) $f \oplus a \in \mathscr{C}$.

Proof. The constant functions are obviously in SW_2. Let f be computed by a strict W_2-program P with accepting node u and rejecting node v. In order to compute $f \wedge a$, $f \wedge (a \oplus b)$, and $f \oplus a$, extend P by the program segments shown in Figs. 1(i), 1(ii), and 1(iii), respectively. Thus SW_2 has the desired closure properties.



(i) (ii) (iii)

FIG. 1

It remains to show $SW_2 \subseteq \mathscr{C}$. Clearly $\mathscr{C}$ contains all functions that can be computed by strict W_2-programs of length 0. Now suppose $\mathscr{C}$ contains all functions that can be computed by a strict W_2-program of length n. Consider any function g computed by a strict W_2-program P of length $n+1$. The last two levels (i.e., levels n and $n+1$) of P are illustrated in Fig. 2, with a and b denoting (not necessarily distinct) literals or constants.

For convenience, we also allow the edges leaving a node of outdegree two to have the labels 0 and 1. The intended interpretation is that the node has outdegree one, the edge labelled 0 is absent, and the edge labelled 1 is present (and unlabelled).

Let f be the function computed by the program obtained from P by deleting level $n+1$, making u accepting and v rejecting. Then $g = [f \wedge (a \oplus \bar{b})] \oplus \bar{b}$. □

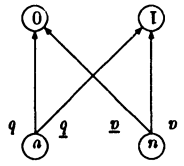


FIG. 2

The proof of Theorem 1 gives a constructive procedure for obtaining a formula for the Boolean function computed by a strict W_2-program, given the program. This enables us to relate the two complexity measures, formula size and program length.

THEOREM 2. *Any Boolean function that can be computed by a W_2-program of length L has formula size at most $3L$.*

Proof. Any W_2-program P can be uniquely decomposed into strict W_2-programs $Q_1, \cdots, Q_t$. The proof proceeds by induction on t.

If $t=1$ then P is strict and the result follows directly from the second part of the proof of Theorem 1. Now suppose $t>1$.

Let v be the sink of Q_1 which is also a sink of P and let u be the other node of P at the same level. Consider the W_2-program P' obtained from P by deleting all of Q_1, except for the node u. Let f' be the function computed by P' and let f_1 be the function computed by Q_1.

If v is labelled by 0, let $f=\bar{f}_1 \wedge f'$ and if v is labelled by 1, let $f=f_1 \vee f'$. Clearly, f is the function computed by P. By the induction hypothesis, f_1 and f' have formula size at most three times the length of Q_1 and P', respectively. Therefore the formula size of f is at most three times the length of P. □

More generally, any Boolean function computed by a constant width branching program has polynomial formula size. The proof is essentially an application of the divide and conquer technique used to compute transitive closure fast in parallel. One recursively constructs formulae which are true if, for the given input, the branching program starting at node i in level k eventually reaches node j in level l.

Hoover [8] showed that constant width polynomial length branching programs compute exactly the same class of functions as constant width polynomial size Boolean circuits with one output. He also obtained a machine characterization for uniform constant width polynomial size Boolean circuits with one output.

Theorem 1, the fact that $\bar{f}=f\oplus a\oplus \bar{a}$, and deMorgan's laws enable us to find more closure properties of SW_2. Specifically, if $f\in SW_2$ and a, b are literals, then $\bar{f}, f\vee a$, and $f\vee(a\oplus b)$ are in SW_2. Notice that

$$E_1^3(x_1, x_2, x_3)=[(x_1\wedge x_3)\vee(x_1\oplus x_2)]\oplus x_3,$$

$$E_{0,1}^3(x_1, x_2, x_3)=[(x_1\oplus \bar{x}_3)\vee(x_1\oplus x_2)]\oplus x_3,$$

$$E_2^4(x_1, x_2, x_3, x_4)=[((x_1\oplus x_3)\vee(x_1\oplus x_2))\wedge(x_3\oplus \bar{x}_4)]\oplus x_1\oplus x_2,$$

$$E_0^n(x_1, \cdots, x_n)=\bar{x}_1\wedge \bar{x}_2\wedge\cdots\wedge \bar{x}_n,$$

$$E_n^n(x_1, \cdots, x_n)=x_1\wedge x_2\wedge\cdots\wedge x_n \quad \text{and}$$

$$E_{1,n-1}^n(x_1, \cdots, x_n)=(x_1\oplus x_2)\vee(x_1\oplus x_3)\vee\cdots\vee(x_1\oplus x_n).$$

Thus all these functions are in SW_2.

Now consider any function $g\in SW_2$. By Theorem 1, g is either a constant function or can be obtained from one of these by repeated applications of rules R1 through

R3. Notice that if only rule R3 is applied, then the resulting functions are constant or parity functions. Recall that a parity function has the form $x_{i_1} \oplus \cdots \oplus x_{i_t}$ where $x_{i_1}, \cdots, x_{i_t}$ are Boolean literals.

Therefore, if g is not a constant or parity function, then either

$$g = (f \wedge a) \oplus \bigoplus_{i=1}^{m} c_i \quad \text{or} \quad g = (f \wedge (a \oplus b)) \oplus \bigoplus_{i=1}^{m} c_i$$

where $m \geq 0$, $f \in SW_2$, and a, b, and c_i, for $1 \leqq i \leqq m$, are literals.

In the first case, substituting 0 for a turns g into a constant or parity function, namely $\bigoplus_{i=1}^{m} c_i$, while in the second case, identifying a and b turns g into a constant or parity function. This observation yields the following result.

LEMMA 3. *Let $g \in SW_2$. Then one of the following conditions holds:*

1. *g is a constant or parity function.*
2. *There is a literal a such that substituting 0 for a turns g into a constant or parity function.*
3. *There are two literals a and b such that identifying a and b turns g into a constant or parity function.*

Lemma 3 is useful for showing that certain functions are not in SW_2. Consider the following example. The function E_1^4 is not a constant or parity function. Let $f(x_2, x_3, x_4) = E_1^4|_{x_1=0}$. Since $f(0, 0, 1) = 1$ and $f(0, 1, 1) = 0$, f is not constant. Also, notice that $f(1, 1, 1) = 0$ and, hence, f is not a parity function. Similarly, let $g(x_2, x_3, x_4) = E_1^4|_{\bar{x}_1=0}$. Then $g(0, 0, 0) = 1$ and $g(0, 0, 1) = g(0, 1, 1) = 0$. If $h(x_1, x_3, x_4) = E_1^4|_{x_1=x_2}$, then $h(0, 0, 0) = h(1, 0, 0) = 0$ and $h(0, 1, 0) = 1$. Finally, let $k(x_1, x_3, x_4) = E_1^4|_{\bar{x}_1=x_2}$. Then $k(1, 0, 0) = 1$ and $k(1, 1, 0) = k(1, 1, 1) = 0$. Thus g, h and k are all neither constant nor parity functions. Since E_1^4 is a symmetric function, it follows from Lemma 3 that $E_1^4 \notin SW_2$.

Together with similar arguments one can show that $E_{h,k}^n \in SW_2$ if and only if one of the following conditions is true.

1. $n \leqq 3$.
2. $n = 4$ and $h = k = 2$.
3. $h = k = 0$.
4. $h = k = n$.
5. $h \leqq 1$ and $k \geqq n - 1$.

We are also able to show that all functions in SW_2 can be computed by short strict W_2-programs.

LEMMA 4. *If $g \in SW_2$ is a Boolean function of n variables, then there is a strict W_2-program of length $O(n^2)$ that computes g.*

Proof. By induction on n.

All Boolean functions of 1 variable can be computed by strict W_2-programs of length 1. If g is a constant or parity function, then g can be computed by a strict W_2-program of length 0 or n, respectively. Therefore we may assume that $n \geqq 2$ and either

$$g = (f \wedge a) \oplus \bigoplus_{i=1}^{m} c_i \quad \text{or} \quad g = (f \wedge (a \oplus b)) \oplus \bigoplus_{i=1}^{m} c_i$$

where $m \geqq 0$, $f \in SW_2$, and a, b and c_i, for $1 \leqq i \leqq m$, are literals.

Consider the function $\bigoplus_{i=1}^{m} c_i$. If it is the constant 0 function (which is the case when $m = 0$), then $g = f \wedge a$ or $g = f \wedge (a \oplus b)$. When $\bigoplus_{i=1}^{m} c_i$ is the constant 1 function,

a program to compute g can be obtained from a program to compute $f \wedge a$ or $f \wedge (a \oplus b)$ by interchanging the labels of the two sinks. Now suppose $\bigoplus_{i=1}^{m} c_i$ is not constant. Since $c \oplus c = 0$ and $c \oplus \bar{c} = 1$, it is unnecessary to have $c_i = c_j$ or $c_i = \overline{c_j}$ for $1 \leqq i \neq j \leqq m$. In particular, this implies $m \leqq n$. Therefore the length of the shortest strict W_2-program that computes g exceeds the length of the shortest strict W_2-program that computes f at most $n+2$.

Finally, we may assume, without loss of generality that neither a nor $\bar{a}$ appear in f. Otherwise, in the first case, by replacing all occurrences of a and $\bar{a}$ by 1 and 0, respectively, we could obtain a new function f' containing neither a nor $\bar{a}$ such that $f' \wedge a = f \wedge a$. Similarly, in the second case, all occurrences of a and $\bar{a}$ can be replaced by $\bar{b}$ and b, respectively.

Since f contains at most $n-1$ variables, it follows that there is a strict W_2-program of length $O(n^2)$ that computes g. □

Our next result shows that, in a geometric sense, the functions computed by strict W_2-programs are not too complicated. We have to introduce some notation.

A *cube* is a subset of $\{0, 1\}^n$ of the form

$$\{x | x_{i_1} = a_1, \cdots, x_{i_r} = a_r\}$$

where $a_1, \cdots, a_r \in \{0, 1\}$ and $0 \leqq r \leqq n$. The *dimension* of the cube is defined to be $n-r$. A *striped cube* is a subset of $\{0, 1\}^n$ of the form $\{x | x_{i_1} = a_1, \cdots, x_{i_r} = a_r$ and $x_{j_1} \oplus \cdots \oplus x_{j_t} = b\}$ where $0 \leqq r, t \leqq n$ and $a_1, \cdots, a_r, b \in \{0, 1\}$. As above, $n-r$ is called the *dimension* of the striped cube. Let Z_n be the smallest number such that, for all n-ary functions $f \in SW_2$, the set of accepted inputs $f^{-1}(1)$, can be represented as a disjoint union of Z_n striped cubes.

LEMMA 5. $Z_n \leqq 4 \times 2^{n/2} - 2$.

Proof. By induction on n. The theorem is clearly true for $n = 1$. Consider any n-ary Boolean function $f \in SW_2$ and let Z be the smallest number such that $f^{-1}(1)$ can be represented as a disjoint union of Z striped cubes. One of the cases of Lemma 3 applies.

If f is a constant or parity function, then $Z \leqq 1$.

If there is a literal a such that substituting 0 for a turns f into a constant or parity function f_1, then $f = (f_1 \wedge \bar{a}) \vee (f_2 \wedge a)$ where f_2 is a function of $n-1$ variables. In this case $Z \leqq 1 + Z_{n-1}$.

Finally suppose there are two literals, a and b which, when identified, turn f into a constant or parity function. Then $f = (f_1 \wedge a \wedge b) \vee (f_2 \wedge \bar{a} \wedge \bar{b}) \vee (f_3 \wedge a \wedge \bar{b}) \vee (f_4 \wedge \bar{a} \wedge b)$ where f_1 and f_2 are constant or parity functions and f_3 and f_4 depend on at most $n-2$ variables. In this case $Z \leqq 2 + 2Z_{n-2}$.

Since f was arbitrary, we have $Z_n \leqq \max\{1, 1 + Z_{n-1}, 2 + 2Z_{n-2}\}$. □

Now consider the Boolean function

$$f(x_1, \cdots, x_n) = (x_1 \oplus x_2) \wedge (x_3 \oplus x_4) \wedge \cdots \wedge (x_{n-1} \oplus x_n).$$

From Theorem 1 it is easy to see that $f \in SW_2$. Each accepted input contains exactly $n/2$ variables with value 1. Therefore if $S \subseteq f^{-1}(1)$ is a striped cube, then $\# S \leqq 2$. In particular, this implies that $Z_n \geqq 2^{n/2-1}$.

3. Lower bounds for monotone W_2-programs. Lemma 5 can be used to obtain lower bounds on the length of W_2-programs and monotone W_2-programs.

Let $\mathcal{S}$ be a system (i.e. collection) of subsets of $\{0,1\}^n$. For example, $\mathcal{S}$ might be the system of cubes or the system of striped cubes. An $\mathcal{S}$-program is a sequence

$$(S_1, a_1), (S_2, a_2), \cdots, (S_m, a_m)$$

where $S_i \in \mathcal{S}$ and $a_i \in \{0,1\}$ for all i. The *length* of the program is m. This program computes an n-ary function f in the following way: Let $b \in \{0,1\}^n$. If $b \notin \bigcup_i S_i$ then $f(b) = 0$. If $b \in S_i - \bigcup_{j<i} S_j$, then $f(b) = a_i$. For any Boolean function f, the $\mathcal{S}$-complexity $C_{\mathcal{S}}(f)$ of f is defined to be the length of the shortest $\mathcal{S}$-program that computes f.

By Lemma 5, for any function g computable by a strict W_2-program, $g^{-1}(1)$ can be represented as a disjoint union of at most $4\times 2^{n/2}-2$ striped cubes. Therefore, if $\mathcal{S}$ is the system of all striped cubes, then $C_{\mathcal{S}}(f)/(4\times 2^{n/2}-2)$ is a lower bound for the number of strict W_2-programs comprising any W_2-program that computes f and, hence, the length of any W_2-program that computes f.

If $\mathcal{S}$ is the system of all cubes, then Plumstead and Plumstead [12] have shown that $2(3/2)^{n-1} \leqq C_{\mathcal{S}}(x_1 \oplus \cdots \oplus x_n) \leqq 5^{n/3}$.

We call an $\mathcal{S}$-program $(S_1, a_1), \cdots, (S_m, a_m)$ *monotone* if $a_i = 1$ for all i. For any Boolean function f, the monotone $\mathcal{S}$-complexity $MC_{\mathcal{S}}(f)$ is defined as the length of the shortest monotone $\mathcal{S}$-program that computes f. By Lemma 5, $MC_{\mathcal{S}}(f)/(4\times 2^{n/2}-2)$ is a lower bound for the length of any monotone W_2-program that computes f.

THEOREM 6. *Every monotone W_2-program that computes E_k^n or $E_{k,n}^n$ has length at least $\binom{n}{k}/((4\times 2^{n/2}-2)n)$.*

Proof. Let $S = \{x | x_{i_1} = a_1, \cdots, x_{i_r} = a_r$ and $x_{j_1} \oplus \cdots \oplus x_{j_t} = b\}$ be any striped cube occurring in a monotone $\mathcal{S}$-program for E_k^n or $E_{k,n}^n$. It is easily seen that at least $k-1$ of the a_i's are 1. Otherwise the $\mathcal{S}$-program would accept an input in which fewer than k variables have value 1. Hence the number of inputs x, such that $x \in S$ and $w(x) = k$ is at most $n - r \leqq n$. Thus $MC_{\mathcal{S}}(E_k^n)$, $MC_{\mathcal{S}}(E_{k,n}^n) \geqq \binom{n}{k}/n$. □

Let $\mathcal{D} = \{S_1, \cdots, S_m\}$ be a system of sets. $\mathcal{D}$ is called a Δ-system if $S_i \cap S_j = \bigcap_{h=1}^m S_h$ for all $i \neq j$. Stated alternatively, any element in $\bigcup_{h=1}^m S_h$ is either contained in every set or is contained in exactly one set.

Erdös and Rado [5] showed that, for all natural numbers p and k, if $\mathcal{E}$ is a system of more than $F(k,p) = k + k^k(p-1)^{k+1}$ sets each of cardinality at most k, then $\mathcal{E}$ contains a subsystem of p sets which is a Δ-system. We will use this fact in order to derive lower bounds for the length of monotone W_2-programs that compute the functions E_k^n.

THEOREM 7. *Let P be a monotone W_2-program that computes E_k^n. Then the length of P is at least $n\binom{n}{k}/F(k,4)$.*

Proof. Suppose that the input $a \in \{0,1\}^n$ is accepted by P at some accepting node v. Among the strict W_2-programs comprising P, let Q be the one which contains v. Recall that any W_2-program can be uniquely decomposed into strict W_2-programs.

If the length of Q is less than n, then some variable x_i would not be tested during the computation of Q on input a. Let a' be the input obtained from a by changing the value of its ith component. Then $a' \in I_Q(v)$. Recall that $I_Q(v)$ is the set of inputs which cause the computation of Q to reach vertex v. Also note that $w(a') \neq k$ and therefore a' is not accepted by P. Since P is monotone, the computation of P on input a' must reach the source of Q. It will continue from there to v, thereby accepting a'. Hence the length of Q is at least n.

Next we show that $\#I_P(v) \leqq F(k,4)$. Suppose, to the contrary, that $\#I_P(v) > F(k,4)$. Let $\mathcal{E} = \{S(a) | a \in I_P(v)\}$. Then $\mathcal{E}$ contains a Δ-system $\mathcal{D} = \{D_1, D_2, D_3, D_4\}$. Let $G = \bigcap_{i=1}^4 D_i$ and $H = \bigcup_{i=1}^4 D_i$.

A new strict W_2-program Q' can be obtained from Q by the following modifications. For all $j \in G$, delete all edges labelled $\overline{x_j}$ and delete all occurrences of the label x_j. This corresponds to fixing the value of the variable x_j to be 1. For all $j \notin H$, delete all edges labelled x_j and delete all occurrences of the label $\overline{x_j}$. This corresponds to fixing the value of the variable x_j to be 0. For $i = 1, 2, 3, 4$, choose a new variable y_1. Then, for each $j \in D_i - G$, replace the labels x_j and $\overline{x_j}$ by y_i and $\overline{y_i}$, respectively.

Let $b = (b_1, \cdots, b_4) \in \{0, 1\}^4$. Then $f_{Q'}(b) = f_Q(c)$ where $c_j = 1$ if $j \in G$, $c_j = 0$ if $j \notin H$, and $c_j = b_i$ if $j \in D_i - G$. Notice that $w(c) = \#G + \sum_{i=1}^4 b_i(\#D_i - \#G)$. Since $\#D_1 = \#D_2 = \#D_3 = \#D_4 = k > \#G$, it follows that $w(c) = k$ if and only if $b_i = 1$ for exactly one value of i. In this case $S(c) = D_i \in \mathscr{E}$. Thus $E_1^4(b) = 1$ implies $c \in I_P(v)$. If $E_1^4(b) = 0$, then P does not accept c and the computation of P on input c does not reach the accepting node v. However, since P is monotone, the computation does reach the source of Q. It follows that $f_{Q'} = E_1^4$. This contradicts the fact that Q' is a strict W_2-program.

Therefore P must be comprised of at least $\binom{n}{k}/F(k, 4)$ strict W_2-programs, each of length at least n. □

A similar argument can be used to show that the length of any program which computes $E_{h,k}^n$ is at least $\binom{n}{k}(n-k+h)/F(k, 4)$ for $0 \leqq h \leqq k \leqq n$.

4. A lower bound for W_2-programs. The following lower bound relies on the similarity between the majority function and its complement.

THEOREM 8. *Every W_2-program P that computes $E_{\lceil n/2 \rceil, n}^n$ has length $\Omega(n^2/\log n)$.*

Proof. Decompose P into strict W_2-programs $Q_1, Q_2, \cdots$ such that, for all l, the nodes in Q_l are closer to the source of P than the nodes in Q_{l+1}. For $l = 1, 2, \cdots$ let v_l be a sink of P which is also a sink of Q_l.

Consider the border region $B = \{x \in \{0, 1\}^n \mid \lceil n/2 \rceil - 2 \leqq w(x) \leqq \lceil n/2 \rceil + 1\}$ and, for $l = 1, 2, \cdots$, let $\tau_l = \sum_{d \leqq l} \#(I_P(v_d) \cap B)$. We want to find a recurrence relation for the numbers τ_l.

By Lemma 5, $I_{Q_l}(v_l)$ can be represented as a disjoint union of striped cubes $S_1, \cdots, S_m$ where $m \leqq 4 \times 2^{n/2} - 2$. Consider any such striped cube

$$S = \{x \mid x_{i_1} = a_1, \cdots, x_{i_r} = a_r \text{ and } x_{j_1} \oplus \cdots \oplus x_{j_t} = b\}.$$

We can assume that $\{i_1, \cdots, i_r\} \cap \{j_1, \cdots, j_t\} = \varnothing$ and $t \neq 1$. Let $\sigma_d(S) = \#(I_P(v_d) \cap B \cap S)$.

First suppose that v_l is an accepting node of P. When $l = 1$, no inputs have yet been rejected. Therefore, at least $\lceil n/2 \rceil - 1$ of the a_i's are 1 and $\sigma_1(S) = \#(S \cap B) \leqq n^2$. Hence $\tau_1 = \sum_{h=1}^m \sigma_1(S_h) \leqq n^2(4 \times 2^{n/2} - 2)$.

More generally, if at least $\lceil n/2 \rceil - 2$ of the a_i's are 1, we have $\sigma_l(S) \leqq \#(S \cap B) \leqq n^3$. Now consider the case when fewer than $\lceil n/2 \rceil - 2$ of the a_i's are 1. Let $x \in I_P(v_l) \cap B \cap S$.

Since $w(x) = \lceil n/2 \rceil$ or $\lceil n/2 \rceil + 1$, there exist at least three indices $q \notin \{i_1, \cdots, i_r\}$ such that $x_q = 1$. Among these indices, at least two, say q_1 and q_2, must both be elements of $\{j_1, \cdots, j_t\}$ or both be elements of the complement of this set. In either case, let x' be obtained from x by changing both x_{q_1} and x_{q_2} to 0. Then $x' \in S$. However, $x' \notin I_P(v_l)$ because v_l is an accepting node and $w(x') = \lceil n/2 \rceil - 2$ or $\lceil n/2 \rceil - 1$. It follows that x' was rejected previously and thus $x' \in \bigcup_{d<l} (I_P(v_d) \cap B \cap S)$. On the other hand every such x' can be obtained in this way from at most $\binom{n}{2}$ vectors x. Therefore

$$\sigma_l(S) \leqq \max\left\{n^3, \binom{n}{2} \# \bigcup_{d<l} (I_P(v_d) \cap B \cap S)\right\}.$$

Since the cubes $S_1, \cdots, S_m$ are disjoint,

$$\begin{aligned}
\tau_l &= \tau_{l-1} + \sum_{h=1}^{m} \sigma_l(S_h) \\
&\leqq \tau_{l-1} + \sum_{h=1}^{m} \left[n^3 + \binom{n}{2} \sum_{d<l} \#(I_P(v_d) \cap B \cap S_h) \right] \\
&= \tau_{l-1} + n^3 m + \binom{n}{2} \sum_{d<l} \sum_{h=1}^{m} \#(I_P(v_d) \cap B \cap S_h) \\
&\leqq \tau_{l-1} + n^3(4 \times 2^{n/2} - 2) + \binom{n}{2} \sum_{d<l} \#(I_P(v_d) \cap B) \\
&= n^3(4 \times 2^{n/2} - 2) + \left(\binom{n}{2} + 1 \right) \tau_{l-1} \\
&< n^3 2^{n/2+2} + \frac{n^2}{2} \tau_{l-1}.
\end{aligned}$$

Similarly, if v_l is a rejecting node, then the same inequalities concerning τ_l can be derived. Thus

$$\tau_l < 2^{n/2-l+4} n^{2l}.$$

Let $L = (5n/16 - 4)/(2 \log n - 1)$. Then $\tau_l < 2^{13n/16}$ for all $l \leqq L$. Therefore P must be composed of more than L strict W_2-programs.

Let $\lambda = \min \{\text{length}(Q_d) | 1 \leqq d \leqq L\}$. If $\lambda \leqq n/8$ the theorem is proven. Thus, assume $\lambda < n/8$.

Consider Q_l where $1 \leqq l \leqq L$ and length $(Q_l) = \lambda$. On any path from the source of Q_l to v_l at most λ variables are examined. Therefore $I_{Q_l}(v_l)$ can be represented as a union of striped cubes of dimension at least $n - \lambda$. (This follows directly from the proof of Lemma 5.) Consider any such striped cube

$$S = \{x | x_{i_1} = a_1, \cdots, x_{i_r} = a_r \text{ and } x_{j_1} \oplus \cdots \oplus x_{j_t} = b\}$$

where $r \leqq \lambda$. Assume v_l is an accepting node of P. Let α be the number of a_i's that are 1 and let β be the number of a_i's that are 0. We want to estimate, from below, the number of vectors in S that have weight $\lceil n/2 \rceil - 1$ or $\lceil n/2 \rceil - 2$. We first consider

$$C = \{x | x_{i_1} = a_1, \cdots, x_{i_r} = a_r \text{ and } w(x) = \lceil n/2 \rceil - 1 \text{ or } \lceil n/2 \rceil - 2\}.$$

Then,

$$\begin{aligned}
\#C &= \binom{n-r}{\lceil n/2 \rceil - 1 - \alpha} + \binom{n-r}{\lceil n/2 \rceil - 2 - \alpha} \\
&= \binom{n+1-r}{\lceil n/2 \rceil - 1 - \alpha} \\
&= \binom{n+1-\alpha-\beta}{\lceil n/2 \rceil + (1-\alpha-\beta)/2 - (3+\alpha-\beta)/2} \\
&= \binom{n+1-\alpha-\beta}{\lceil (n+1-\alpha-\beta)/2 \rceil - (c+\alpha-\beta)/2}
\end{aligned}$$

where $c \in \{2, 3, 4\}$. This is minimized if $\beta = 0$ and $\alpha = r$. Hence

$$\begin{aligned}
\#C &\geqq \binom{n+1-r}{\lceil n/2 \rceil - 1 - r} \\
&\geqq \binom{n+1-\lambda}{\lceil n/2 \rceil - 1 - \lambda} \\
&\geqq \binom{7n/8}{3n/8} \Big/ n \\
&= (7n/8)^{7n/8} / ((3n/8)^{3n/8} (n/2)^{n/2} O(n^{3/2})) \\
&= 2^{n[7(\log 7 - 3)/8 + 3(3 - \log 3)/8 + 1/2]} / O(n^{3/2}) \\
&> n^2 2^{n(-14/80 + 39/80 + 1/2)} \\
&= n^2 2^{13n/16}.
\end{aligned}$$

Next we show that any vector $x \in C$ can be obtained from some vector x' in $C \cap S$ by changing at most two bits of x'. From any $x' \in C \cap S$, we can obtain fewer than n^2 vectors x in this way. Therefore $n^2 \#(S \cap C) \geqq \#C$.

Let $x \in C - S$. We construct x'. We first assume that $w(x) = \lceil n/2 \rceil - 2$. If $x_q = 0$ for some $q \in \{j_1, \cdots, j_t\}$, set $x_q = 1$. If $x_{j_1} = \cdots = x_{j_t} = 1$ find a component $q \notin \{a_1, \cdots, a_r\}$ such that $x_q = 0$; this is possible since $\lambda \leqq n/8$. Set $x_{j_1} = 0$ and $x_q = 1$. The case $w(x) = \lceil n/2 \rceil - 1$ is handled similarly.

Recall that v_l is assumed to be an accepting node of P. Hence all elements in $S \cap C$ must have been rejected previously. Thus

$$2^{13n/16} < \#(S \cap C) \leqq \tau_{l-1} < 2^{13n/16}.$$

If v_l is a rejecting node of P, the same contradiction is derived using analogous arguments. □

5. Conclusions. Obviously, we are just beginning to understand the limited power of constant width branching programs. The few examples given (both positive and negative) all concern *counting*. In some cases (e.g. E_2^4) we observe nontrivial ways of counting.

Of course, we need not restrict ourselves to counting functions. However, counting is a basic component in many computationally nontrivial problems, and eventually we hope to understand the extent to which bounded width branching programs can count.

Acknowledgments. The authors would like to thank Michael Fischer, for helpful discussion, and Ron Graham, for giving us reference [5].

REFERENCES

[1] ALLAN BORODIN, DANNY DOLEV, FAITH E. FICH AND WOLFGANG PAUL, *Bounds for width two branching programs*, Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 87–93.

[2] A. BORODIN, M. FISCHER, D. KIRKPATRICK, N. LYNCH AND M. TOMPA, *A time-space tradeoff for sorting on non-oblivious machines*, Proc. 20th Annual Symposium on Foundations of Computer Science, 1979, pp. 319–327.

[3] A. COBHAM, *The recognition problem for the set of perfect squares*, Research Paper RC-1704, IBM Watson Research Center, Yorktown Heights, NY, April 1966.

[4] A. K. CHANDRA, M. L. FURST AND R. J. LIPTON, *Multi-party protocols*, Proc. 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 94–99.

[5] P. ERDÖS AND R. RADO, *Intersection theorems for systems of sets*, J. London Math. Society, 35 (1960), pp. 85-90.

[6] MICHAEL J. FISCHER, ALBERT R. MEYER AND MICHAEL S. PATERSON, $\Omega(n \log n)$ *lower bounds on length of Boolean formulas*, this Journal, 11 (1982), pp. 416-427.

[7] M. FURST, J. SAXE AND M. SIPSER, *Parity, circuits and the polynomial-time hierarchy*, Proc. 22nd Annual Symposium on Foundations of Computer Science, 1981, pp. 260-270.

[8] J. HOOVER, *On circuit width*, unpublished manuscript.

[9] V. M. KHRAPCHENKO, *The complexity of realization of symmetrical functions by formulae*, Mat. Zametki, 11, 1 (1972), pp. 109-120; Math. Notes of the Academy of Sciences of the USSR, 11 (1972), pp. 70-76.

[10] W. MASEK, *A fast algorithm for the string editing problem and decision graph complexity*, M.Sc. Thesis, Massachusetts Institute of Technology, Cambridge, May 1976.

[11] E. I. NECIPORUK, *A Boolean function*, Soviet Math. Dokl., 2, 4 (1966), pp. 999-1000.

[12] B. R. PLUMSTEAD AND J. B. PLUMSTEAD, *Bounds for cube coloring*, SIAM J. Alg. Discr. Meth., 6 (1985), pp. 73-78.

[13] PUDLÁK, *A lower bound on the complexity of branching programs*, in Lecture Notes in Computer Science 176, M. P. Chytil and V. Koubek, eds., Springer-Verlag, Berlin, 1984, pp. 480-489.

[14] JAMES B. SHEARER, private communication.

[15] B. VILFAN, *The complexity of finite functions*, Ph.D. Thesis, Dept. Electrical Engineering Tech. Report 97, Project MAC, Massachusetts Institute of Technology, Cambridge, 1972.

[16] HONG-JAI WEI, private communication.

[17] A. YAO, *Lower bounds by probabilistic arguments*, Proc. 24th Annual Symposium on Foundations of Computer Science, 1983, pp. 420-428.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

ON THE PROBABLE PERFORMANCE OF HEURISTICS FOR BANDWIDTH MINIMIZATION*

JONATHAN S. TURNER†

Abstract. Most research in algorithm design relies on worst-case analysis for performance comparisons. Unfortunately, worst-case analysis does not always provide an adequate measure of an algorithm's effectiveness. This is particularly true in the case of heuristic algorithms for hard combinatorial problems. In such cases, analysis of the probable performance can yield more meaningful results and can provide insight leading to better algorithms. The problem of minimizing the bandwidth of a sparse symmetric matrix by performing simultaneous row and column permutations, is an example of a problem for which there are well-known heuristics whose practical success has lacked a convincing analytical explanation. A class of heuristics introduced by Cuthill and McKee in 1969, and referred to here as the *level algorithms*, are the basis for bandwidth minimization routines that have been widely used for over a decade. At the same time, it is easy to construct examples, showing that the level algorithms can produce solutions that differ from optimal by an arbitrarily large factor. This paper provides an analytical explanation for the practical success of the level algorithms, by showing that for random matrices having optimal bandwidth no larger than k, any level algorithm will produce solutions that differ from optimal by a small constant factor. The analysis also suggests another class of algorithms with better performance. One algorithm in this class is shown to produce solutions that are nearly optimal.

Key words. bandwidth minimization problem, heuristic algorithms, approximation algorithms, probable performance, probabilistic analysis, NP-completeness

1. Introduction. Let M be a symmetric matrix and let k be the largest integer for which there is a nonzero entry $M[i, i+k]$; k is called the bandwidth of M. It is often possible to reduce the bandwidth of a matrix by performing simultaneous row and column permutations. Most common matrix operations can be performed more efficiently if the matrices are in small bandwidth form. The matrices can also be stored more efficiently in this form. The matrix bandwidth minimization problem is usually re-cast as a graph theory problem; for any matrix M, the graph corresponding to M has an edge joining vertices i and j if and only if $M[i, j]$ is nonzero.

Let $G=(V, E)$ be a graph with $V=\{1, 2, \cdots, n\}$. A *layout* of G is a permutation on $\{1, 2, \cdots, n\}$. Define the *bandwidth* of G with respect to a layout τ by $\phi_\tau(G)=\max_{\{u,v\}\in E}|\tau(u)-\tau(v)|$. The *bandwidth* of G is defined by $\phi(G)=\min_\tau \phi_\tau(G)$. The *bandwidth minimization problem* (for graphs) is to determine for a graph G and an integer k if $\phi(G)\leqq k$. Papadimitriou [9] first showed that the bandwidth minimization problem is NP-complete. Garey, Graham, Johnson and Knuth [7] later strengthened this result, showing that the problem remains NP-complete when restricted to free binary trees. Several heuristic algorithms for bandwidth minimization were proposed in the late sixties and early seventies. More recently, Saxe [10] has found a dynamic programming algorithm which can determine if a graph has bandwidth k in time $O(n^{k+1})$ for any fixed value of k. Monien and Sudborough [8] showed how to reduce the time bound to $O(n^k)$. One of the most successful heuristic algorithms is one discovered by Cuthill and McKee [5] which is a member of a class of algorithms which are referred to here as the *level algorithms*. An algorithm is classified as a level algorithm if for all graphs $G=(V, E)$ the layout τ produced by the algorithm satisfies

$$\forall u, v\in V \qquad d(\tau^{-1}(1), u)<d(\tau^{-1}(1), v)\Rightarrow\tau(u)<\tau(v)$$

* Received by the editors March 5, 1984, and in revised form February 28, 1985. A preliminary version of this paper appeared in the Proceedings of the Fifteenth Annual ACM Symposium on the Theory of Computing, Boston, Massachusetts, April 1983, © 1983, Association for Computing Machinery, Inc.

† Computer Science Department, Washington University, St. Louis, Missouri 63130.

where $d(x, y)$ denotes the length of the shortest path connecting vertices x and y. The level algorithms are reasonably fast and have proved to be quite successful in practice. On the other hand, one can easily construct examples in which the ratio of the bandwidth of the layout produced by a level algorithm to the actual bandwidth of the graph is arbitrarily large. Consequently one must resort to probabilistic analysis to gain insight to their practical success.

Let $G=(V, E)$ be generated by the following random experiment.

- Let $V=\{1, 2, \cdots, n\}$, $E=\varnothing$.
- For each $\{u, v\}$, $1 \leqq u < v \leqq n$, add the edge $\{u, v\}$ to E with probability p.

The probability distribution defined by this experiment is denoted $\mathbf{\Gamma}_n(p)$ and the notation $G \in \mathbf{\Gamma}_n(p)$ means that G is a random graph generated in this way. In § 2 it is shown that for almost all $G \in \mathbf{\Gamma}_n(p)$, $n/\phi(G) \leqq 1+\varepsilon$ when $p \geqq (c \ln n)/n$ and $\varepsilon > 0$, $c > 0$ are fixed. (We say that a property holds for *almost all* graphs if the probability of the property holding approaches one as the number of vertices gets large. This notion is often described by the phrase "in probability".) Consequently, if τ is any layout at all, $\phi_\tau(G)/\phi(G) \leqq (1+\varepsilon)$ for almost all random graphs $G \in \mathbf{\Gamma}_n(p)$. This makes it pointless to compare the probable performance of bandwidth minimization algorithms on random graphs in $\mathbf{\Gamma}_n(p)$. Therefore another class of probability distributions is introduced and used for most of the results given here. Let $G=(V, E)$ be generated by the following random experiment.

- Let $V=\{1, 2, \cdots, n\}$, $E=\varnothing$.
- For each $\{u, v\}$, $1 \leqq u < v \leqq n$ such that $|u-v| \leqq \psi$ include the edge $\{u, v\}$ in E with probability p.

The probability distribution defined by this experiment is denoted $\mathbf{\Psi}_n(\psi, p)$. Now, let $G \in \mathbf{\Psi}_n(\psi, p)$ and randomly re-number the vertices of G. The resulting distribution is denoted $\mathbf{\Omega}_n(\psi, p)$. Note that if $G \in \mathbf{\Omega}_n(\psi, p)$ then $\phi(G) \leqq \psi$. Also, if H is a graph with $\phi(H) \leqq \psi$, then H can be generated by $\mathbf{\Omega}_n(\psi, p)$. Furthermore, in § 2 we show that for large enough ψ, almost all $G \in \mathbf{\Omega}_n(\psi, p)$ satisfy $\psi/\phi(G) \leqq 1+\varepsilon$ for any fixed $\varepsilon > 0$. The use of $\mathbf{\Omega}_n(\psi, p)$ allows us to explore properties that are common to most graphs having bandwidth $\leqq \psi$, but rare for unrestricted graphs. Heuristics like the level algorithms exploit such properties to produce good layouts for most graphs.

It is shown in § 3 that if A is any level algorithm and $A(G)$ is the bandwidth of the layout produced by A on the graph G then $A(G) < (1+\varepsilon)(3-p)\phi(G)$ for almost all $G \in \mathbf{\Omega}_n(\psi, p)$, where $\varepsilon > 0$, $0 < p < 1$ are fixed, and $\ln n = o(\psi)$. If in addition $\psi < n/2$, then $(1-\varepsilon)(2-p)\phi(G) < A(G)$. The analysis leads to a new class of algorithms called the *modified level algorithms*, for which it is shown that $A(G) \leqq 2\phi(G) + O(\log n)$ for any modified level algorithm A and $G \in \mathbf{\Omega}_n(\psi, p)$. In § 4, a specific modified level algorithm, MLA1 is studied and it is shown that $\mathrm{MLA1}(G) = \phi(G) + O(\log n)$ for almost all $G \in \mathbf{\Omega}_n(\psi, p)$ when $\psi < n/4$. § 5 presents several other modified level algorithms, discusses running times and summarizes empirical studies comparing their performance. § 6 shows how to improve the running times of the above algorithms through more careful selection of the "starting vertex". Finally, § 7 contains several results concerning properties of random graphs. Conditions are given for connectivity of random graphs in $\mathbf{\Psi}_n(\psi, p)$ and probable upper bounds are given for the diameter of random graphs in $\mathbf{\Gamma}_n(p)$ and $\mathbf{\Psi}_n(\psi, p)$.

A word of caution. All but a few of the results proved in this paper are probabilistic in nature. That is, they hold for almost all graphs under some probability distribution. The statements of lemmas and theorems include the phrase "almost all" and specify the probability distribution, but to avoid being tedious, the proofs assert various properties without repeating this qualification.

2. Bandwidth of graphs in $\Gamma_n(p)$ and $\Psi_n(\psi, p)$. The following results demonstrate that almost all random graphs in the usual model, have bandwidth nearly as large as the number of vertices.

Define $\lambda_n(c) = -\ln n/\ln c$. Note that $\lambda_n(c) > 0$ when $0 < c < 1$ and $n > 1$, $c^{\lambda_n(c)} = 1/n$ and $\lim_{n\to\infty} \lambda_n(c) = \infty$ for c fixed $0 < c < 1$. We will usually write $\lambda(c)$ for $\lambda_n(c)$.

THEOREM 2.1. *Let $0 < p < 1$ be fixed. For almost all $G \in \Gamma_n(p)$, $\phi(G) > n - 4\lambda(1-p)$.*

THEOREM 2.2. *Let $\varepsilon > 0$, $c > 0$ be fixed, $p = (c \ln n)/n$. For almost all $G \in \Gamma_n(p)$, $\phi(G) \geqq n(1-\varepsilon)$.*

For $G = (V, E)$, the notation u—v means $\{u, v\} \in E$ and $u \not\!\!-\!\!- v$ means that $\{u, v\} \notin E$. Similarly if $U \subseteq V$ and $W \subseteq V$ then U—W means that some vertex in U is adjacent to some vertex in W. The proofs of Theorems 2.1 and 2.2 require the following lemmas.

LEMMA 2.1. *Let $G = (V, E)$ be a graph on n vertices. $\phi(G) \leqq n - 2k \Rightarrow \exists V_1, V_2 \subseteq V$ such that $|V_1| = |V_2| = k$ and $V_1 \not\!\!-\!\!- V_2$.*

Proof. If $\phi(G) \leqq n - 2k$ then there is a layout τ such that u—$v \Rightarrow |\tau(u) - \tau(v)| \leqq n - 2k$. Let $V_1 = \{\tau^{-1}(1), \cdots, \tau^{-1}(k)\}$ and $V_2 = \{\tau^{-1}(n-k+1), \cdots, \tau^{-1}(n)\}$. If V_1—V_2 then there are vertices $u \in V_1$ and $v \in V_2$ such that u—v. But by the definition of V_1 and V_2, $\tau(u) \leqq k$ and $\tau(v) \geqq n - k + 1$; hence $|\tau(u) - \tau(v)| > n - 2k$, which contradicts the definition of τ. □

LEMMA 2.2. *Let $0 < p < 1$ and $G = (V, E) \in \Gamma_n(p)$.*

$$P(\phi(G) \leqq n - 2k) \leqq \left(\frac{en}{k}(1-p)^{k/2}\right)^{2k}.$$

Proof. By Lemma 2.1, $P(\phi(G) \leqq n-2k) \leqq P(\exists V_1, V_2$ such that $|V_1| = |V_2| = k \wedge V_1 \not\!\!-\!\!- V_2)$. Since there are k^2 "potential edges" joining V_1 and V_2, all of which must be absent if $V_1 \not\!\!-\!\!- V_2$, this last probability is

$$\leqq \binom{n}{k}\binom{n-k}{k}(1-p)^{k^2} \leqq \left(\frac{en}{k}\right)^{2k}(1-p)^{k^2} = \left(\frac{en}{k}(1-p)^{k/2}\right)^{2k}. \qquad \square$$

Proof of Theorem 2.1. Applying Lemma 2.2 with $k = 2\lambda(1-p)$ gives

$$P(\phi(G) \leqq n - 4\lambda(1-p)) \leqq \left(\frac{en}{2\lambda(1-p)}(1-p)^{\lambda(1-p)}\right)^{4\lambda(1-p)}$$

$$= \left(\frac{e}{2\lambda(1-p)}\right)^{4\lambda(1-p)} \to 0. \qquad \square$$

Proof of Theorem 2.2. Applying Lemma 2.2 with $k = \varepsilon n/2$ gives

$$P(\phi(G) \leqq n(1-\varepsilon)) \leqq \left(\frac{en}{\varepsilon n/2}(1-p)^{\varepsilon n/4}\right)^{\varepsilon n} \leqq \left(\frac{2e}{\varepsilon}e^{-\varepsilon np/4}\right)^{\varepsilon n}$$

$$= \left(\frac{2e}{\varepsilon}n^{-\varepsilon c/4}\right)^{\varepsilon n} \to 0. \qquad \square$$

Theorems 2.1 and 2.2 show that even for sparse random graphs $G \in \Gamma_n(p)$, $(n/\phi(G)) \to 1$. Consequently, even the most trivial algorithms (for example, the algorithm that always outputs the identity layout) can produce layouts having bandwidth close to $\phi(G)$ as n gets large. If one is to make meaningful distinctions among algorithms based on their probable performance, some other probability distribution is required. The distributions $\Omega_n(\psi, p)$ and $\Psi_n(\psi, p)$ are used here. Obviously, any structural property of a graph occurs with the same probability in both distributions. It is clear

that if $G \in \Psi_n(\psi, p)$ then $\phi(G) \leqq \psi$. The following theorem gives a probabilistic lower bound on $\phi(G)$.

THEOREM 2.3. *Let* $0<p<1$ *be fixed,* $\ln n \leqq \psi < n$. *For almost all* $G \in \Psi_n(\psi, p)$, $\phi(G) > \psi - 4\lambda(1-p)$.

Proof. Let $G' \subseteq G$ be the subgraph induced by vertices $\{1, 2, \cdots, \psi\}$. Note that G' is a random graph with distribution $\Gamma_\psi(p)$. Applying Theorem 2.1, $\phi(G') > \psi - 4\lambda_\psi(1-p)$. The theorem follows from the fact that $\phi(G) \geqq \phi(G')$. □

An immediate consequence of this result is that as ψ gets large, it comes within a factor of $1+\varepsilon$ of $\phi(G)$, for any fixed $\varepsilon > 0$. While Theorem 2.3 is sufficient for the results proved here, it is interesting to consider a tighter relationship between ψ and $\phi(G)$.

CONJECTURE. *Let* $0<p<1$ *be fixed. There is some constant* $c = c(p) > 0$ *such that if* $c \ln n \leqq \psi \leqq n - c \ln n$ *then for almost all* $G \in \Psi_n(\psi, p)$, $\phi(G) = \psi$.

3. Probabilistic algorithms for bandwidth minimization. Before proceeding we need the following definitions. Let $G = (V, E)$ and define $V_i(u) = \{v | d(u, v) = i\}$ for all $u \in V$. Also let $V_i = V_i(1)$. Next, define $l_i(u) = \min V_i(u)$ and $r_i(u) = \max V_i(u)$. Let $l_i = l_i(1)$ and $r_i = r_i(1)$. Note that $|V_i| \leqq r_i - l_i$. Define

$$level\,(G) = \min_{u \in V} \max_{i \geqq 0} |V_i(u)|,$$

$$LEVEL\,(G) = \min_{u \in V} \max_{i \geqq 0} |V_i(u) \cup V_{i+1}(u)| - 1.$$

Note that if A is any level algorithm at all

$$level\,(G) \leqq A(G) \tag{1}$$

and if A makes the best possible choice for $\tau^{-1}(1)$

$$A(G) \leqq LEVEL\,(G). \tag{2}$$

In the next few sections, we will consider only algorithms that do always make the best choice. We can satisfy this requirement by trying all possible choices for $\tau^{-1}(1)$, at a cost of a factor of n in the running time. In § 6, we will relax this restriction.

Consider the tree T in Fig. 1. It is not difficult to see that $\phi(T) = 2$ and $level\,(T) = 4$. The example is readily extended. For any integer $k > 0$ one can construct a tree T_k

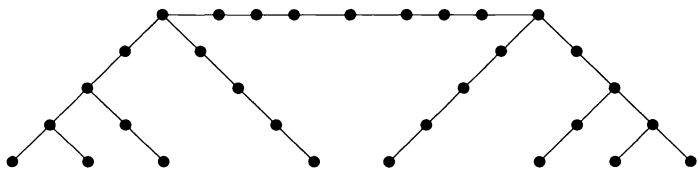


FIG. 1. *Tree demonstrating poor worst-case performance of level algorithms.*

such that $\phi(T_k) = 2$ and $level\,(T_k) = k$. (This result can be improved. There is a similar but more complicated construction which gives trees T_k with n vertices, $level\,(T_k) = \Omega(n/\log n)$ and $\phi(T_k) = o(\log n)$.) This implies that the worst case performance of the level algorithms can be arbitrarily poor. In spite of this, the level algorithms perform quite well on random graphs.

THEOREM 3.1. *Let* $\varepsilon > 0$, $0<p<1$ *be fixed,* $\psi < n$, $\ln n = o(\psi)$. *For almost all* $G \in \Omega_n(\psi, p)$, $LEVEL\,(G) < (1+\varepsilon)(3-p)\phi(G)$.

The theorem is proved by deriving probable upper bounds on $|V_i|$ and then using the definition of *LEVEL*. These bounds are contained in the following lemma.

LEMMA 3.1. *Let* $\varepsilon > 0$, $0 < p < 1$ *be fixed,* $\psi < n$, $\ln n = o(\psi)$. *For almost all* $G \in \Psi_n(\psi, p)$,

$$|V_1| < (1+\varepsilon)p\psi,$$
$$|V_2| < (1+\varepsilon)(2-p)\psi,$$
$$|V_i| < (1+\varepsilon)\psi, \quad \textit{for } i \geqq 3.$$

The proof of Lemma 3.1 appears in § 3.1 along with several technical lemmas required for its proof. We now use it to prove Theorem 3.1.

Proof of Theorem 3.1. By Lemma 3.1 there exists a vertex u for which $\max_{i \geqq 0} |V_i(u) \cup V_{i+1}(u)| - 1 < (1+\varepsilon')(3-p)\psi$ for any fixed $\varepsilon' > 0$. Hence, $LEVEL(G) < (1+\varepsilon')(3-p)\psi$ for any fixed $\varepsilon' > 0$. By Theorem 2.3, $\psi < (1+\varepsilon')\phi(G)$. Selecting ε' so that $(1+\varepsilon')^2 = (1+\varepsilon)$ yields the theorem. □

In Lemma 3.5, it is shown that $|V_2| > (1-\varepsilon)(2-p)$ when $\ln n = o(\psi)$ and $\psi < n/2$. This result is easily extended to show that for all $u \in V$, $|V_2(u)| > (1-\varepsilon)(2-p)\psi$ and hence $level(G) > (1-\varepsilon)(2-p)\psi$. The details are left to the reader. A consequence of this is that the level algorithms are not capable of near optimal performance. However a related class of algorithms, called the *modified level algorithms* is. We will now describe the class of modified level algorithms. In the next section, we describe a specific member of this class that achieves near optimal performance. Define

$$V_2'(u) = \begin{cases} V_2(u) & \text{if } V_3(u) = \varnothing, \\ V_2(u) \cap \{v | v\text{—}w \in V_3(u)\} & \text{if } V_3(u) \neq \varnothing, \end{cases}$$
$$V_1'(u) = (V_1(u) \cup V_2(u)) - V_2'(u),$$
$$V_i'(u) = V_i(u), \qquad i = 0,\ i \geqq 3.$$

Also, let $V_i' = V_i'(1)$, $l_i'(u) = \min V_i'(u)$, $r_i'(u) = \max V_i'(u)$, $l_i' = l_i'(1)$, $r_i' = r_i'(1)$. Formally, A is a modified level algorithm, if the layout τ produced for the graph $G = (V, E)$ satisfies

$$\forall u, v \in V \quad u \in V_i'(\tau^{-1}(1)) \wedge v \in V_{i+1}'(\tau^{-1}(1)) \Rightarrow \tau(u) < \tau(v).$$

Let

$$level'(G) = \min_{u \in V} \max_{i \geqq 0} |V_i'(u)|,$$

$$LEVEL'(G) = \min_{u \in V} \max_{i \geqq 0} |V_i'(u) \cup V_{i+1}'(u)| - 1.$$

If A is any modified level algorithm then $level'(G) \leqq A(G)$ and if A makes the best possible choice for the starting vertex then $A(G) \leqq LEVEL'(G)$. For the modified level algorithms, we can show that for almost all $G \in \Psi_n(\psi, p)$, $|V_i'| < \psi + O(\log n)$ for all $i \geqq 0$ when $\ln n = o(\psi)$, $\psi \leqq (1-\varepsilon)n/2$. From this we obtain the following result.

THEOREM 3.2. *Let* $0 < p < 1$ *be fixed,* $\ln n = o(\psi)$, $\psi < n$. *For almost all* $G \in \Omega_n(\psi, p)$ $LEVEL'(G) = 2\phi(G) + O(\log n)$.

The proof of Theorem 3.2 is contained in § 3.2. This result shows that the class of modified level algorithms is capable of better performance than the class of level algorithms. In § 4, we focus on a specific modified level algorithm and show that it produces nearly optimal layouts.

3.1. Technical lemmas. The following lemmas are used in the proof of Lemma 3.1.

LEMMA 3.2. *Let $\varepsilon > 0$, $0 < p < 1$ be fixed, $\alpha = (1+\varepsilon)\lambda(1-p^2) \leqq \psi < n$. For almost all $G \in \Psi_n(\psi, p)$, there exists a path of length two between every pair of vertices u, v such that $|u-v| \leqq 2\psi - \alpha$.*

Proof. Let $u, v \in V$ with $|u-v| \leqq 2\psi - \alpha$. Let $i = 2\psi - |u-v|$. The probability that $d(u, v) > 2$ is $\leqq (1-p^2)^i$. Since for each i there are no more than n such pairs, the probability that any pair is not joined by a 2-path is

$$\leqq \sum_{i=\lceil \alpha \rceil}^{2\psi-1} n(1-p^2)^i < n(1-p^2)^\alpha \sum_{i=0}^{\infty} (1-p^2)^i = p^{-2}n^{-\varepsilon} \to 0. \qquad \square$$

LEMMA 3.3. *Let $\varepsilon > 0$, $0 < p < 1$ be fixed, $\alpha = (1+\varepsilon)\lambda(1-p^2) \leqq \psi < n$. For almost all $G \in \Psi_n(\psi, p)$, $r_i - 3\psi \leqq r_{i-3} < l_i - (2\psi - \alpha)$, for all $i \geqq 3$.*

Proof. The shortest path from 1 to r_i must pass through some $u \in V_{i-3}$. Clearly $r_i - u \leqq 3\psi$; hence $r_i - 3\psi \leqq u \leqq r_{i-3}$. To see that $r_{i-3} < l_i - (2\psi - \alpha)$, assume otherwise. Then there is some vertex v on the shortest path from 1 to r_{i-3} such that $l_i - (2\psi - \alpha) \leqq v < l_i$ and $d(1, v) \leqq i-3$. By Lemma 3.2 there is a 2-path from v to l_i, giving $d(1, l_i) \leqq i-1$, which is a contradiction. $\square$

LEMMA 3.4. *Let $\varepsilon > 0$, $0 < p < 1$ be fixed, $\alpha = (1+\varepsilon)\lambda(1-p^2) \leqq \psi < n$. For almost all $G \in \Psi_n(\psi, p)$, $r_i - l_i < \psi + \alpha$, for all $i \geqq 3$.*

Proof. By Lemma 3.3 $r_i \leqq r_{i-3} + 3\psi$ and $l_i > r_{i-3} + (2\psi - \alpha)$. Hence,

$$r_i - l_i < (r_{i-3} + 3\psi) - (r_{i-3} + (2\psi - \alpha)) = \psi + \alpha. \qquad \square$$

From Lemma 3.4, we conclude that $|V_i| < \psi + \alpha$ for $i \geqq 3$, but the lemma says nothing about the size of V_1 and V_2. As we shall see, these cases differ from the rest and will be handled in Lemma 3.5. First however, we need a proposition concerning the binomial distribution, $\mathbf{B}(n, p)$. By definition if $x \in \mathbf{B}(n, p)$ then $P(x = k) = \binom{n}{k} p^k (1-p)^{n-k}$. The following proposition is from Angluin and Valiant [1].

PROPOSITION 3.1. *If $x \in \mathbf{B}(n, p)$ then for any ε, $0 < \varepsilon < 1$, $P(x \leqq (1-\varepsilon)np) < e^{-\varepsilon^2 np/2}$ and $P(x \geqq (1+\varepsilon)np) < e^{-\varepsilon^2 np/3}$.*

LEMMA 3.5. *Let $\varepsilon > 0$, $0 < p < 1$ be fixed, $v = -(1+\varepsilon)/\ln(1-p^2)$, $\alpha = c \ln n \leqq \psi < n$. For almost all $G \in \Psi_n(\psi, p)$, $(1-\varepsilon)p\psi < |V_1| < (1+\varepsilon)p\psi$ and $|V_2| < (1+\varepsilon)(2-p)\psi$. Also, if $\psi < n/2$ then $(1-\varepsilon)(2-p)\psi - \alpha < |V_2|$.*

Proof. $|V_1|$ is a binomial random variable in $\mathbf{B}(\psi, p)$. By Proposition 3.1,

$$P(|V_1| < (1-\varepsilon)p\psi) < e^{-\varepsilon^2 p\psi/2} \to 0,$$
$$P(|V_1| > (1+\varepsilon)p\psi) < e^{-\varepsilon^2 p\psi/3} \to 0.$$

This establishes the bounds on $|V_1|$. Since $|V_2| \leqq 2\psi - |V_1|$,

$$|V_2| < 2\psi - (1-\varepsilon)p\psi < (1+\varepsilon)(2-p)\psi.$$

When $\psi < n/2$, Lemma 3.2 gives

$$|V_2| \geqq (2\psi - \alpha) - |V_1| > (2 - (1+\varepsilon)p)\psi - \alpha > (1-\varepsilon)(2-p)\psi - \alpha. \qquad \square$$

Proof of Lemma 3.1. Immediate from Lemmas 3.4 and 3.5. $\square$

3.2. More technical lemmas. The following lemmas are used in the proof of Theorem 3.2.

LEMMA 3.6. *Let $\varepsilon > 0$, $0 < p < 1$ be fixed, $(1+\varepsilon)\lambda(1-p^2) \leqq \psi < n$. For almost all $G = (V, E) \in \Psi_n(\psi, p)$, $u \in V \wedge |\{u-\psi, \cdots, u+\psi\} \cap V_i| \geqq (1+\varepsilon)\lambda(1-p) \Rightarrow u$—$V_i$.*

Proof. By Lemma 3.2, any pair of vertices u, v with $|u-v| \leqq \psi$, is joined by a 2-path and hence $|d(1, u)-d(1, v)| \leqq 2$. Thus, for each vertex u there are at most five sets V_i such that $|\{u-\psi_1, \cdots, u+\psi\} \cap V_i| \geqq 1$. Hence, the probability that for any $G \in \Psi_n(\psi, p)$, the assertion is not true is $\leqq 5n(1-p)^{(1+\varepsilon)\lambda(1-p)} = 5n^{-\varepsilon} \to 0$. □

LEMMA 3.7. *Let $\varepsilon > 0$, $0 < p < 1$ be fixed, $\alpha = (1+\varepsilon)\lambda(1-p^2) \leqq \psi < n$. For almost all $G \in \Psi_n(\psi, p)$ there exists a path of length three between every pair of vertices u, v such that $|u-v| \leqq 3\psi - \alpha$.*

Proof. Let $u, v \in V$ be such that $i = 3\psi = |u-v| \geqq \alpha$. Let $x_j = u + \psi - j$ for $0 \leqq j \leqq i$, as illustrated in Fig. 2. Clearly any 3-path connecting u and v must pass through one of $x_0, \cdots, x_i$. The probability that no 3-path joins u and v is

$$
\begin{aligned}
&= P(\text{no 3-path} \wedge u \not\!\!-\!\!- x_0 \wedge \cdots \wedge u \not\!\!-\!\!- x_i) \\
&\quad + \sum_{j=0}^{i} P(\text{no 3-path} \wedge u \not\!\!-\!\!- x_0 \wedge \cdots \wedge u \not\!\!-\!\!- x_{j-1} \wedge u \text{—} x_j) \\
&= (1-p)^{i+1} P(\text{no 3-path} | u \not\!\!-\!\!- x_0 \wedge \cdots \wedge u \not\!\!-\!\!- x_i) \\
&\quad + \sum_{j=0}^{i} p(1-p)^j P(\text{no 3-path} | u \not\!\!-\!\!- x_0 \wedge \cdots \wedge u \not\!\!-\!\!- x_{j-1} \wedge u \text{—} x_j) \\
&< (1-p)^{i+1} \\
&\quad + p \sum_{j=0}^{i} (1-p)^j P(\text{no 3-path} | u \not\!\!-\!\!- x_0 \wedge \cdots \wedge u \not\!\!-\!\!- x_{j-1} \wedge u \text{—} x_j \wedge u \not\!\!-\!\!- x_{j+1} \wedge \cdots \wedge u \not\!\!-\!\!- x_i) \\
&= (1-p)^{i+1} + p \sum_{j=0}^{i} (1-p)^j (1-p^2)^{i-j+1} \\
&= (1-p)^{i+1} + p(1+p)(1-p)^{i+1} \sum_{j=0}^{i} (1+p)^j \\
&= (1-p)^{i+1} + p(1+p)(1-p)^{i+1} \frac{(1+p)^{i+1}-1}{p} \\
&< (1+p)(1-p^2)^{i+1}.
\end{aligned}
$$

Since for each value of i there are at most n vertex pairs u, v such that $|u-v| = i$, the probability that any pair u, v with $|u-v| \leqq 3\psi - \alpha$ is not connected by a 3-path is

$$
\leqq \sum_{i=\lceil \alpha \rceil}^{3\psi-1} n(1+p)(1-p^2)^{i+1} < n(1+p)(1-p^2)^{\alpha} \sum_{i=0}^{\infty} (1-p^2)^i = \frac{1+p}{p^2} n^{-\varepsilon} \to 0. \qquad \square
$$

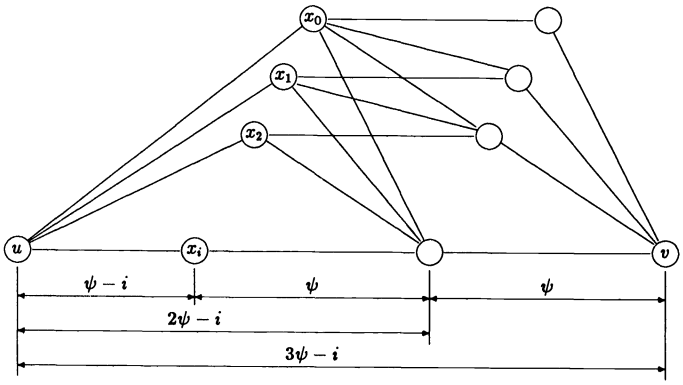


FIG. 2. *Definition of x_is.*

LEMMA 3.8. *Let* $\varepsilon > 0$, $0 < p < 1$ *be fixed,* $\alpha = (1+\varepsilon)\lambda(1-p^2)$, $\beta = (1+\varepsilon)\lambda(1-p)$ *and* $\max(\alpha, 2\beta) \leqq \psi \leqq (n-\beta)/2$. *For almost all* $G \in \Psi_n(\psi, p) |V_i'| \leqq \psi + \alpha$ *for* $i \geqq 0$.

Proof. The result follows from Lemma 3.4 for $i \geqq 3$ and is immediate for $i = 0$. Before proving the theorem for $1 \leqq i \leqq 2$ we first need to show that $|V_3 \cap \{\psi+2, \cdots, 2\psi + \lceil\beta\rceil\}| \geqq \beta$. Let $A = \{\psi+2, \cdots, 2\psi+1\}$. By Lemmas 3.4 and 3.7 $A \subseteq V_2 \cup V_3$. Let $x = |A \cap V_3|$. Clearly if $x \geqq \beta$ then we are done. Assume then that $x < \beta$ and let $B = \{2\psi+2, \cdots, 2\psi + (\lceil\beta\rceil - x) + 1\}$ and let $y = |B|$. If $u \in B$, then $|\{u-\psi, \cdots, 2\psi+1\} \cap V_2| \geqq \psi - \beta$. Since $\psi \geqq 2\beta$, $u \in B \Rightarrow |\{u-\psi, \cdots, u+\psi\} \cap V_2| \geqq \beta$. Thus by Lemma 3.6 $B \subseteq V_3$. Since $x + y \geqq \beta$ we have that $|V_3 \cap \{\psi+2, \cdots, 2\psi + \lceil\beta\rceil\}| \geqq \beta$.

Now, by Lemma 3.3, $l_3 \geqq 2\psi - \alpha + 1$. This implies that $l_2' \geqq \psi - \alpha + 1$ and since $r_2' \leqq 2\psi + 1$, it follows that $|V_2'| \leqq \psi + \alpha$ as claimed. Finally, note that if $u \in A$ and $u \geqq \psi + \beta$ then using Lemma 3.6, one can show that u—V_3 and hence $u \notin V_1'$. Thus $|V_1'| \leqq \psi + \beta < \psi + \alpha$ as claimed. □

Proof of Theorem 3.2. If $\psi > (n - 2\lambda(1-p))/2$, then since $LEVEL'(G) < n$

$$LEVEL'(G) < 2\psi + 2\lambda(1-p) = 2\psi + O(\log n) = 2\phi(G) + O(\log n).$$

If $\psi \leqq (n - 2\lambda(1-p))/2$ then we can apply Lemma 3.8 giving

$$LEVEL'(G) < 2\psi + 4\lambda(1-p^2) = 2\psi + O(\log n) = 2\phi(G) + O(\log n). \qquad \square$$

A similar analysis yields $level'(G) = \phi(G) + O(\log n)$.

4. Obtaining nearly optimal layouts. In this section a specific modified level algorithm denoted MLA1 is described and analyzed. It is shown that MLA1 is capable of producing nearly optimal layouts for random graphs in $\Omega_n(\psi, p)$.

For a graph $G = (V, E)$, we define the *grandchildren* of v with respect to u by

$$gc_u(v) = V_2(v) \cap V_{i+2}'(u) \quad \forall v \in V_i'(u)$$

and the *grandparents* of v with respect to u by

$$gp_u(v) = V_2(v) \cap V_{i-2}'(u) \quad \forall v \in V_i'(u).$$

Also let $gc(v) = gc_1(v)$, $gp(v) = gp_1(v)$. The algorithm we will analyze is based on the observation that for $G \in \Psi_n(\psi, p)$ if $u, v \in V_i'$ and $v - u$ is not too small, then with high probability $|gc(u)| < |gc(v)|$ and $|gp(u)| > |gp(v)|$. The algorithm MLA1 is described in Fig. 3. Define MLA1 (G) as the bandwidth of the layout produced by MLA1 on graph G.

For each $u \in V$
  Let τ be any layout that satisfies the following conditions for all $x, y \in V$.
    (a) $x \in V_i'(u) \wedge y \in V_{i+1}'(u) \Rightarrow \tau(x) < r(y)$
    (b) $1 \leqq i \leqq 2 \wedge x, y \in V_i'(u) \wedge |gc_u(x)| < |gc_u(y)| \Rightarrow \tau(x) < \tau(y)$
    (c) $i \geqq 3 \wedge x, y \in V_i'(u) \wedge |gp_u(x)| > |gp_u(y)| \Rightarrow \tau(x) < \tau(y)$
  Output the layout having minimum bandwidth.

FIG. 3. *Modified level algorithm* 1.

THEOREM 4.1. *Let* $0 < p < 1$ *be fixed,* $\ln n = o(\psi)$, $\psi \leqq n/4$. *For almost all* $G \in \Omega_n(\psi, p)$ MLA1 $(G) = \phi(G) + O(\log n)$.

The key fact used in the proof of Theorem 4.1 is contained in the following lemma.

LEMMA 4.1. *Let* $\varepsilon > 0$, $0 < p < 1$ *be fixed,* $\alpha = (1+\varepsilon)\lambda(1-p^2)$, $\ln n = o(\psi)$, $\psi \leqq n/4$. *For almost all* $G \in \Psi_n(\psi, p)$, $|\tau(u) - u| \leqq 4\alpha$, *where* $u \in V$ *and* τ *is the layout produced by* MLA1 *for which* $\tau(1) = 1$.

Proof of Theorem 4.1. By Lemma 4.1, MLA1 will compute a layout in which no vertex is more than 4α from the "right position". This implies that the bandwidth of the layout output by MLA1 is at most $\psi+8\alpha=\phi(G)+O(\log n)$. □

The proof of Lemma 4.1 requires the following technical lemmas.

LEMMA 4.2. *Let* $\varepsilon<0$, $0<p<1$ *be fixed,* $\alpha=(1+\varepsilon)\lambda(1-p^2)$, $2\alpha\leqq\psi<n$. *For almost all* $G\in\Psi_n(\psi,p)$

$$l'_1=2, \qquad \psi-\alpha\leqq r'_1\leqq\psi+\alpha,$$

$$r'_1-2\alpha<l'_2\leqq r'_1+1, \qquad r'_1+\psi-\alpha\leqq r'_2\leqq r'_1+\psi,$$

$$r'_{i-1}-\alpha<l'_i\leqq r'_{i-1}+1, \qquad r'_{i-1}+\psi-\alpha\leqq r'_i\leqq r'_{i-1}+\psi \quad \textit{for } i\geqq 3.$$

Proof. For $1\leqq i\leqq 2$ the result is implicit in the proof of Lemma 3.8. For $i\geqq 3$, Lemma 3.3 gives $l'_i>r'_{i-3}+2\psi-\alpha$. Since $r'_{i-1}\leqq r'_{i-3}+2\psi$, $l'_i>r'_{i-1}-\alpha$. By Lemmas 3.2 and 3.7 $\{r'_{i-3}+\psi+1,\cdots,r'_{i-3}+2\psi-\alpha\}\subseteq V'_{i-1}$ and $\{r'_{i-3}+2\psi+1,\cdots,r'_{i-3}+3\psi-\alpha\}\subseteq V'_i$ and $\{r'_{i-3}+2\psi-\alpha+1,\cdots,r'_{i-3}+2\psi\}\subseteq V'_{i-1}\cup V'_i$. This implies $r'_{i-1}+1\in V'_i$; hence $l'_i\leqq r'_{i-1}+1$. For $i\geqq 3$, $r'_i\leqq r'_{i-1}+\psi$ is immediate and $r'_i\geqq r'_{i-1}+\psi-\alpha$ follows from Lemma 3.7 and $r'_{i-1}\leqq r'_{i-3}+2\psi$. □

A consequence of Lemma 4.2 is that at least $\psi-\alpha$ of the vertices in V'_i are found in a region containing only vertices in V'_i. The regions associated with V'_i and V'_{i+1} are separated by a transition region containing at most 2α vertices.

LEMMA 4.3. *Let* $\varepsilon>0$, $0<p<1$ *be fixed,* $\alpha=(1+\varepsilon)\lambda(1-p^2)$, $4\alpha\leqq\psi\leqq n/4$. *For almost all* $G\in\Psi_n(\psi,p)$ $1\leqq i\leqq 2\wedge u$, $v\in V'_i\wedge u-v\geqq 4\alpha\Rightarrow|gc(u)|>|gc(v)|$.

Proof. Lemmas 3.2 and 4.2 imply that if u, $v\in V'_i$ and $u-v\geqq\alpha$ then $gc(v)\subseteq gc(u)$. It remains only to show that there is some vertex $x\in gc(u)-gc(v)$. Let $x=u+2\psi-\lceil\alpha\rceil$. If $u-v\geqq 4\alpha$, Lemma 4.2 yields,

$$r'_{i+1}\leqq r'_{i-1}+2\psi<l'_i+2\psi+2\alpha\leqq v+2\psi+2\alpha<u+2\psi-2\alpha<x.$$

Thus $x\notin V'_{i+1}$ and also by Lemma 4.2, $x\notin V'_j$ for any $j\leqq i$. Since, by Lemma 3.2, there is a 2-path from u to x, $x\in gc(u)$. Since $x>v+2\psi$, $x\notin gc(v)$. □

LEMMA 4.4. *Let* $\varepsilon>0$, $0<p<1$ *be fixed,* $\alpha=(1+\varepsilon)\lambda(1-p^2)$, $4\alpha\leqq\psi\leqq n/4$. *For almost all* $G\in\Psi_n(\psi,p)$ $i\geqq 3\wedge u$, $v\in V'_i\wedge u-v\geqq 4\alpha\Rightarrow|gp(u)|<|gp(v)|$.

Proof. Lemmas 3.2 and 4.2 imply that if u, $v\in V'_i$ and $u-v\geqq\alpha$ then $gp(u)\subseteq gp(v)$. It remains to show that there exists some vertex x in $gp(v)-gp(u)$. Let $x=v-2\psi+\lceil\alpha\rceil$. If $u-v\geqq 4\alpha$, Lemma 4.2 yields

$$l'_{i-1}>r'_{i-2}-2\alpha\geqq r'_i-2\psi-2\alpha\geqq u-2\psi-2\alpha\geqq v-2\psi+2\alpha>x.$$

Thus $x\notin V'_{i-1}$ and also by Lemma 4.2, $x\notin V'_j$ for any $j\geqq i$. Since, by Lemma 3.2, there is a 2-path from v to x, $x\in gp(v)$. Since $x<u-2\psi$, $x\notin gp(u)$. □

Proof of Lemma 4.1. By Lemmas 4.2 to 4.4, if $u-v>4\alpha$ then $\tau(v)<\tau(u)$. Consequently, for any u there can be at most 4α vertices v such that $u>v$ and $\tau(u)<\tau(v)$. Similarly, there can be at most 4α vertices w such that $u<w$ and $\tau(u)>\tau(w)$. Hence, $|\tau(u)-u|\leqq 4\alpha$. □

5. Pragmatics. This section reports on the results of empirical studies of several modified level algorithms, including MLA1, described in the previous section. It also contains some implementation details and analyses of the algorithms' running times. Four modified level algorithms were studied. They are denoted here as MLA1 through MLA4. An implementation of MLA1 is shown in Fig. 4. This procedure returns a layout τ, with u as the starting vertex. The strategy for ordering the vertices within levels is the one described in the previous section. The procedure shown calls several

```
(1)  procedure MLA1(G=(V, E), u, τ)
(2)      n := |V|;
(3)      make_mod_levels(G, u, V', ···, V'_{n-1});
(4)      count_gc(G, u, V'_0, ···, V'_{n-1}, ngc, 1, 2);
(5)      count_gp(G, u, V'_0, ···, V'_{n-1}, ngp, 3, n-1);
(6)      for i ∈ |1, 2| → sort(V'_i, ngc(x) < ngc(y)) rof;
(7)      for i ∈ |3, n-1| → sort(V'_i, ngp(x) > ngp(y)) rof;
(8)      next := 1;      {next position in layout}
(9)      for i ∈ |0, n-1| →
(10)         for x ∈ V'_i → τ(x) := next; next := next+1 rof;
(11)     rof;
(12)     return;
(13) end;
```

FIG. 4. *Implementation details for* MLA1.

others. *Make_mod_levels* $(G, u, V'_0, \cdots, V'_{n-1})$ computes $V'_i(u)$ and returns it in the list V'_i for $0 \leqq i \leqq n-1$, using breadth-first-search. The procedures *count_gc* and *count_gp* count the number of "grandchildren" and "grandparents" for vertices in the levels specified by the last two arguments. (For example, the call to *count_gc* in line (4) counts the grandchildren of all vertices in the first two levels and returns the results in the array *ngc*.) The procedure *sort* $(L, R(x, y))$ sorts the list L so that x precedes y in the sorted list if and only if x is related to y under R. For example, *sort* $(V'_i, ngc(x) < ngc(y))$ sorts V'_i so that if $ngc(x) < ngc(y)$ then x precedes y in V'_i. The running time of MLA1 is dominated by the *count_gc* and *count_gp* functions. A straightforward implementation of these gives a running time of $O(n\psi^2)$. The procedure *make_mod_levels* can be implemented to run in $O(|E|) = O(n\psi)$ time, and the sorting steps in lines [6] and [7] require at most $O(n \log n)$.

There are other possible strategies for arranging the vertices within each level. Cuthill and McKee [5], who first suggested the level algorithms, arranged the vertices within levels according to the order in which they were visited by a breadth-first search algorithm. This results in an arbitrary ordering of the first level and arranges each vertex in subsequent levels based on the position of its "leftmost" neighbor. Cheng [2], [3] refined this strategy by ordering the vertices in the first level in increasing order of the number of neighbors in the next level. Adapting this algorithm to the modified level strategy gives the algorithm MLA2, which is shown in Fig. 5. MLA2 calls the procedure *count_ch*, which counts the number of neighbors each vertex has in the "next level". As with *count_gc* and *count_gp*, the calculation is done only for those levels specified by the last two arguments (in this case, just the first level). This can be done in $O(\psi^2)$ time, while the remainder of MLA2 can be done in $O(n\psi)$ time.

The procedure MLA3, shown in Fig. 6 is a cross between MLA1 and MLA2. It uses the strategy of MLA1 to order the vertices in the first level, then reverts to the strategy of MLA2 for all subsequent levels. Computing *ngc* for the vertices in the first level requires $O(\psi^3)$ time. The remainder of MLA3 can be done in $O(n\psi)$.

MLA4 is a refinement of MLA3 designed to improve the running time when the bandwidth is fairly large. Instead of using the number of "grandchildren" to order the vertices in the first level, it uses the number of paths to grandchildren. This can be computed more quickly, since it eliminates the necessity of throwing out duplicates. The total running time of MLA4 is $O(n\psi)$.

MLA2 through MLA4 are more difficult to analyze than MLA1 because decisions made in ordering each level affect the ordering of subsequent levels. Consequently,

```
(1)  procedure MLA2(G = (V, E), u, τ)
(2)      n := |V|;
(3)      make_mod_levels(G, u, V'_0, ..., V'_{n-1});
(4)      count_ch(G, u, V'_0, ..., V'_{n-1}, nch, 1, 1);
(5)      sort(V'_1, nch(x) < nch(y));
(6)      for x ∈ V → τ(x) := 0 rof;      {0 denotes undefined}
(7)      τ(u) := 1;
(8)      next := 2;                      {next position in layout}
(9)      for x ∈ V'_1 → τ(x) := next; next := next + 1 rof;
(10)     left := 2;                      {left end of V'_1 in layout}
(11)     right := next - 1;              {right end of V'_1 in layout}
(12)     for i ∈ |2, n - 1| →
(13)         do left ≦ right →
(14)             x := τ^{-1}(left);
(15)             for {x, y} ∈ E →
(16)                 if τ(y) = 0 → τ(y) := next; next := next + 1 fi
(17)             rof;
(18)             left := left + 1;
(19)         od;
(20)         right := next - 1;          {right end of V'_i}
(21)     rof;
(22)     return
(23) end
```

FIG. 5. *Implementation details for* MLA2.

```
(1)  procedure MLA3(G = (V, E), u, τ)
(2)      n := |V|;
(3)      make_mod_levels(G, u, V'_0, ..., V'_{n-1});
(4)      count_gc(G, u, V'_0, ..., V'_{n-1}, ngc, 1, 1);
(5)      sort(V'_1, ngc(x) < ngc(y));
(6)      for x ∈ V → τ(x) := 0 rof;      {0 denotes undefined}
(7)      τ(u) := 1;
(8)      next := 2;                      {next position in layout}
(9)      for x ∈ V'_1 → τ(x) := next; next := next + 1 rof;
(10)     left := 2;                      {left end of V'_1 in layout}
(11)     right := next - 1;              {right end of V'_1 in layout}
(12)     for i ∈ |2, n - 1| →
(13)         do left ≦ right →
(14)             x := τ^{-1}(left);
(15)             for {x, y} ∈ E →
(16)               if τ(y) = 0 → τ(y) := next; next := next + 1 fi
(17)             rof
(18)             left := left + 1;
(19)         od;
(20)         right := next - 1;          {right end of V'_i}
(21)     rof;
(22)     return
(23) end
```

FIG. 6. *Implementation details for* MLA3.

one might expect that errors made in ordering the early levels could accumulate and cause large errors further on. Experimental results suggest that in fact this does not happen, that the process is self-limiting. However, straightforward analytical techniques for bounding the error give unsatisfactory results.

Figures 7 through 9 summarize the results of a series of experiments that were undertaken to verify the theoretical performance bounds described in the previous sections for MLA1, provide tighter bounds for graphs of moderate size and compare MLA1 to the other modified level algorithms. For each of the data points shown in Fig. 7, ten random graphs in $\Psi_n(n/4, 1/2)$ were generated and each of the algorithms was run. For each algorithm, these ten results were averaged and the difference between these averages and $n/4$ were plotted. The results show that all the algorithms produce good layouts. All of the results are within 20% of $n/4$ and the best are within 2%.

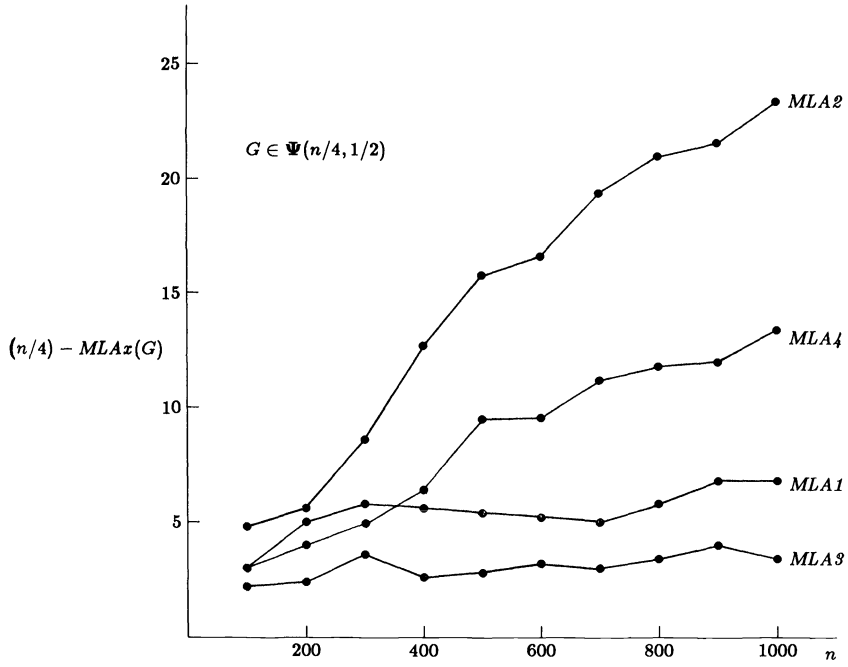


FIG. 7. *Performance of modified level algorithms.*

Figure 8 shows the measured execution times for these runs. (The algorithms were coded in the C programming language and run on a VAX 11/750 under Unix[1].) Here, MLA2 and MLA4 enjoy a substantial advantage. Of course, this speed advantage is directly related to the large value of ψ relative to n. For smaller values of ψ the differences would be less.

One last set of results is shown in Fig. 9. This shows how the performance of the algorithms deteriorates as ψ becomes large relative to n. MLA1 deteriorates first, when $\psi \approx n/4$. This is because the strategy used to order the levels becomes less effective when V_4' becomes much smaller than ψ. MLA3 is not affected by this phenomenon until $\psi \approx n/3$ since the "grandchildren" strategy is used only to order the first level. MLA2 and MLA4 are more robust, maintaining their good performance until $\psi \approx n/2$. At this point all four degenerate from modified level algorithms to level algorithms.

[1] Unix is a trademark of AT&T.

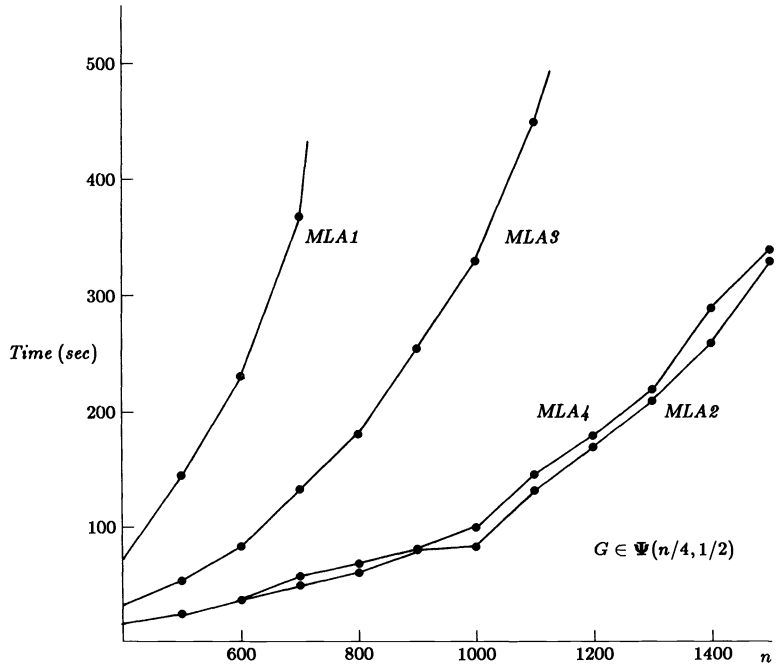


FIG. 8. *Running time of modified level algorithms.*

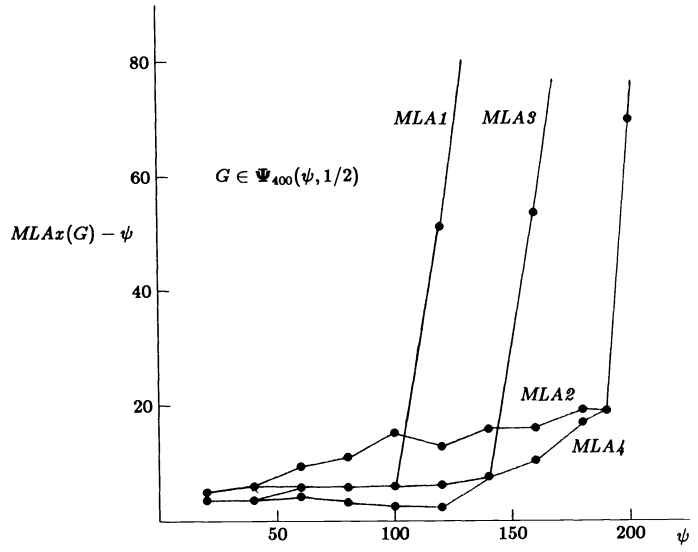


FIG. 9. *Deterioration of modified level algorithms as ψ grows.*

6. Selection of starting vertices. Up until this point we have largely ignored the question of how one selects a good starting vertex in the modified level algorithm. Of course, the brute force solution is simply to try all possibilities and pick the best result. This adds a factor of n to the running times quoted in the previous sections, but does ensure the best possible choice. In this section, we consider strategies that permit us to select small sets of candidate starting vertices, that with high probability, contain a good choice.

The most obvious strategy (suggested by Cuthill and McKee) is to concentrate on vertices with small degree. For $G \in \Psi_n(\psi, p)$ it is reasonable to expect the degree of vertex 1 will be smaller than the degree of most other vertices. The following lemma puts a probable upper bound on the number of low degree vertices that need to be tried to obtain near optimal performance. For $G = (V, E)$, define the set of *low degree vertices by* $ld(G) = \{v \in V | d(v) \leqq d(1)\}$.

LEMMA 6.1. *Let* $\varepsilon > 0$, $0 < p < 1$ *be fixed,* $12(1+\varepsilon)(1/p) \ln n \leqq \psi < n$. *For almost all* $G \in \Psi_n(\psi, p)$, $|ld(G)| < 4\sqrt{(3/p)(1+\varepsilon)\psi \ln n}$.

Proof. Let $0 < \alpha \leqq \psi p/2$. By Proposition 3.1

$$P(d(1) \geqq \psi p + \alpha) < e^{-\alpha^2/3\psi p}.$$

For $v \in V$ such that $(2\alpha/p) < v < n - (2\alpha/p)$

$$P(d(v) \leqq \psi p + \alpha) < e^{-\alpha^2/2\psi p}.$$

Letting $\alpha = \sqrt{3(1+\varepsilon)\psi p \ln n}$ yields

$$P(d(v) < d(1)) < 2\, e^{-\alpha^2/3\psi p} = 2n^{-(1+\varepsilon)}.$$

Since there are $< n$ such vertices v,

$$P(\exists v | (2\alpha/p) < v < n - (2\alpha/p) \wedge d(v) < d(1)) < 2n^{-\varepsilon} \to 0.$$

Consequently there are at most $4\alpha/p$ vertices in $ld(G)$. □

Lemma 6.1 gives us a way of ensuring a good starting vertex. The cost is an added factor of $O(\sqrt{\psi \ln n})$ in the running time.

The next theorem suggests another method for identifying a good starting vertex. Let $L_x(G)$ be the layout τ of G produced by MLA1 for which $\tau(x) = 1$ and let $\text{MLA1}_x(G)$ be the bandwidth of G with respect to $L_x(G)$.

THEOREM 6.1. *Let* $0 < p < 1$ *be fixed,* $\ln n = o(\psi)$, $\psi \leqq n/16$. *For almost all* $G \in \Psi_n(\psi, p)$ $(x \in V \wedge \tau = L_x(G) \wedge y = \tau^{-1}(n)) \Rightarrow \text{MLA1}_y(G) = \phi(G) + O(\log n)$.

The procedure suggested by Theorem 6.1 is this. Pick an arbitrary vertex x and run MLA1 with x as the starting vertex. Let y be the "rightmost vertex" in the resulting layout. Now, re-run MLA1 with y as the starting vertex. Theorem 6.1 states that the resulting layout is close to optimal. The proof of Theorem 6.1 requires the following lemmas.

LEMMA 6.2. *Let* $\varepsilon > 0$, $0 < p < 1$ *be fixed,* $\alpha = (1+\varepsilon)\lambda(1-p^2)$, $\ln n = o(\psi)$, $\psi \leqq n/16$. *For almost all* $G \in \Psi_n(\psi, p)$ $(x \in V \wedge \tau = L_x(G) \wedge y = \tau^{-1}(n)) \Rightarrow (y < 4\alpha \vee y > n - 4\alpha)$.

Proof. Let $x \in V$, $\tau = L_x(G)$ and $y = \tau^{-1}(n)$. Also let G_l be the subgraph induced by $\{1, 2, \cdots, x\}$ and let G_r be the subgraph induced by $\{x, \cdots, n\}$. Note that $G_l \in \Psi_x(\psi, p)$ and $G_r \in \Psi_{n-x+1}(\psi, p)$. Next, let $\tau_l = L_x(G_l)$, $\tau_r = L_x(G_r)$ and let $y_l = \tau_l^{-1}(x)$, $y_r = \tau_r^{-1}(n - x + 1)$. The analysis now divides into several cases.

Case 1. $x < n/4$. For any $z \in \{1, \cdots, x\}$, Lemma 3.2 implies

$$d(x, z) \leqq 2\frac{x}{\psi} < \frac{n}{2\psi} \quad \text{and} \quad d(x, n) \geqq \frac{n-x}{\psi} > \frac{3n}{4\psi}.$$

Thus, $d(x, z) < d(x, n)$ for any $z < x$. This implies that $y \in \{x, \cdots, n\}$. In fact, $y = y_r$. To see this, first note that for $i \geqq 0$, $V_i'(x, G_r) \subseteq V_i'(x, G)$ (the notation for the vertices belonging to each level has been extended to distinguish between the two graphs). Furthermore, if $z \in V_i'(x, G_r)$ where $i \geqq 4$ then $gp_x(z, G_r) = gp_x(z, G)$ (the notation for

the grandparents of z has been similarly extended). Consequently, $y = y_r$. Applying Lemma 4.1 to G_r yields $y > n - 4\alpha$.

Case 2. $n/4 \leqq x \leqq 3n/4 \wedge y \in \{x, \cdots, n\}$. By the same argument used in case 1, $y = y_r$ and applying Lemma 4.1 to G_r yields $y > n - 4\alpha$.

Cases 3, 4 are symmetric with 1, 2. □

Proof of Theorem 6.1. Let $x \in V$, $\tau = L_x(G)$ and $y = \tau^{-1}(n)$. By Lemma 6.2, either $y < 4\alpha$ or $y > n - 4\alpha$. Since the two cases are symmetric, we will only discuss the former. Let $\sigma = L_y(G)$, G_r be the subgraph induced by $\{y, \cdots, n\}$ and $\sigma_r = L_y(G_r)$. Note that the restriction of σ to $\{y, \cdots, n\}$ is the same as σ_r. By Lemma 3.2, every vertex in $\{1, \cdots, y-1\}$ is connected to y by a 2-path and no vertex in $\{1, \cdots, y-1\}$ is adjacent to any vertex in $V_3'(y)$. Consequently, $\{1, \cdots, y-1\} \subseteq V_1'(y)$. Let $\{u, v\} \in E$, and consider the following three cases.

Case 1. $\{u, v\} \subseteq \{y, \cdots, n\}$. By Lemma 4.1, $|\sigma_r(u) - \sigma_r(v)| \leqq \psi + 8\alpha$. Consequently, $|\sigma(u) - \sigma(v)| \leqq \psi + 12\alpha$.

Case 2. $\{u, v\} \subseteq \{1, \cdots, y-1\}$. Since $\{u, v\} \subseteq V_1'(y)$ and by Lemma 3.8, $|V_1'(y)| \leqq 4\alpha + (\psi + \alpha) = \psi + 5\alpha$, it follows that $|\sigma(u) - \sigma(v)| \leqq \psi + 5\alpha$.

Case 3. $u \in \{1, \cdots, y-1\}$, $v \in \{y, \cdots, n\}$. Because $u < y$, $v < y + \psi$. By Lemma 4.1, $|\sigma(v) - v| \leqq 4\alpha$, giving $\sigma(v) < y + \psi + 4\alpha < \psi + 8\alpha$. Since $u \in V_1'$, $\sigma(u) \leqq \psi + 5\alpha$. Thus, $|\sigma(u) - \sigma(v)| \leqq \psi + 8\alpha$.

In all three cases above, we conclude that $|\sigma(u) - \sigma(v)| \leqq \psi + 12\alpha = \psi + O(\log n) = \phi(G) + O(\log n)$. □

The method for selecting a starting vertex outlined above can be refined in several directions. One way is to run MLA1 several times, each time using the rightmost vertex from the previous run as the starting vertex for the next run. This extends the applicability of the method to larger values of ψ. Another refinement is to run MLA1 several times as just described, but then take the 4α rightmost vertices from the last run and use these as a set of candidate starting vertices. With high probability, either vertex 1 or vertex n is in this set. The results obtained in this way may be somewhat closer to optimal, but the cost is an extra $O(\log n)$ factor in the running time.

7. Properties of random graphs. This section is largely independent and examines several properties of random graphs, particularly graphs in $\Psi_n(\psi, p)$. The following theorem is a special case of a result proved in Erdös and Renyi in [6].

THEOREM 7.1. *Let* $-1 < \varepsilon < 1$ *be fixed,* $p = (1+\varepsilon)(\ln n)/n$, $G \in \Gamma_n(p)$. *If* $\varepsilon > 0$, G *is almost always connected. If* $\varepsilon < 0$, G *is almost always disconnected.*

The following is a similar result for random graphs with small bandwidth.

THEOREM 7.2. *Let* $-1 < \varepsilon < 1$ *be fixed,* $0 < p < 1$, $\psi = (1+\varepsilon)\lambda(1-p)/2$, $\psi \to \infty$. *If* $\varepsilon > 0$ *then almost all* $G \in \Psi_n(2\psi, p)$ *are connected. If* $\varepsilon < 0$ *then almost all* $G \in \Psi_n(\psi, p)$ *are disconnected.*

To prove Theorem 7.2, we need to introduce another probability distribution and prove two lemmas. Let n and ψ be positive integers, $\psi < n$, $0 < p < 1$, and let $G = (V, E)$ be a random variable defined by the following experiment.

- Let $V = \{1, 2, \cdots, n\}$.
- For each pair u, v, $1 \leqq u < v \leqq n$ and $|u - v| \leqq \psi \vee |u - v| \geqq n - \psi$ include the edge $\{u, v\}$ in E with probability p.

The probability distribution defined by this experiment is denoted $\Psi_n^c(\psi, p)$.

LEMMA 7.1. *Let* $-1 < \varepsilon < 1$ *be fixed,* $0 < p < 1$, $\psi = (1+\varepsilon)\lambda(1-p)/2$, $1 \leqq \psi \leqq n/2$, $G \in \Psi_n^c(\psi, p)$. *If* $\varepsilon > 0$ *then* G *almost always contains no isolated vertex. If* $\varepsilon < 0$ *then* G *almost always contains at least one isolated vertex.*

Proof. First part $-\varepsilon > 0$. Let

$$X_v = \begin{cases} 1 & \text{if } v \text{ is isolated,} \\ 0 & \text{if } v \text{ is not isolated,} \end{cases}$$

$$X = X_1 + X_2 + \cdots + X_n,$$

$$\mu = E(X) = \sum_{v=1}^{n} E(X_v) = n(1-p)^{2\psi}.$$

Then,

$$P(X \geqq 1) \leqq \mu = n(1-p)^{2\psi} = n^{-\varepsilon} \to 0.$$

This completes the proof of the first part.

Second part $-\varepsilon < 0$. Let $X, X_1, \cdots, X_n$ be defined as before.

$$E(X^2) = \sum_{u=1}^{n} \sum_{v=1}^{n} E(X_u X_v) = \sum_{u=1}^{n} \sum_{v=1}^{n} P(u \text{ and } v \text{ are both isolated})$$

$$= n(1-p)^{2\psi} + 2\psi n(1-p)^{4\psi-1} + n(n-2\psi-1)(1-p)^{4\psi}.$$

By Chebyshev's inequality,

$$P(X=0) \leqq \frac{\sigma^2}{\mu^2} = \frac{E(X^2)-\mu^2}{\mu^2} = \frac{1}{\mu} + \frac{2\psi p}{n(1-p)} - \frac{1}{n} < n^{\varepsilon} + (1+\varepsilon)\frac{p \ln n}{n(1-p)\ln(1/(1-p))}.$$

The function $p/((1-p)\ln(1/(1-p)))$ gets large as $p \to 1$. However, $1 \leqq \psi < (1+\varepsilon)\lambda(1-p) \Rightarrow p \leqq 1 - n^{-(1+\varepsilon)}$. Hence,

$$P(X=0) < n^{\varepsilon} + (1+\varepsilon)\frac{\ln n}{n^{-\varepsilon}\ln n^{1+\varepsilon}} = 2n^{\varepsilon} \to 0. \qquad \square$$

Let $D(G)$ denote the diameter of G.

LEMMA 7.2. *Let $0 < \varepsilon \leqq p < 1$ where ε is fixed, and let $G \in \Gamma_n(p)$. Then*

$$P(D(G) > 2) \leqq \binom{n}{2}(1-\varepsilon^2)^{n-2}.$$

Proof. Let u and v be any two vertices in G. The number of possible 2-paths between them is $n-2$ and the probability that any one of them is absent is $1-p^2$. Hence the probability that u and v are not connected by a 2-path is $(1-p^2)^{n-2}$. Consequently, the probability that any pair of vertices is not connected by a 2-path is

$$\leqq \binom{n}{2}(1-p^2)^n - 2 \leqq \binom{n}{2}(1-\varepsilon^2)^{n-2}. \qquad \square$$

Proof of Theorem 7.2. First part $-\varepsilon > 0$. G is connected if the first 2ψ vertices induce a connected subgraph and all other vertices have at least one edge to a lower numbered vertex. By Lemma 7.2, if $p > \alpha$ for some $\alpha > 0$ then the probability that the first 2ψ vertices induce a subgraph of diameter greater than two is

$$\leqq \binom{2\psi}{2}(1-\alpha^2)^{2\psi-2} \to 0.$$

Hence, if p is bounded below, the first 2ψ vertices almost always induce a connected subgraph. If on the other hand $p \to 0$ we must use Theorem 7.1 to establish that the first 2ψ vertices induce a connected subgraph. This requires that we show that there

exists some $\gamma > 0$ such that $p \geqq (1+\gamma)(\ln(2\psi))/(2\psi)$. From the hypothesis of the theorem

$$\frac{p(2\psi)}{\ln(2\psi)} = (1+\varepsilon)\frac{p \ln n}{\ln(1/(1-p))\ln(2\psi)} > (1+\varepsilon/2)$$

for large enough n since $p \to \ln(1/(1-p))$ as $p \to 0$ and $n > 2\psi$. Now, the probability that any of the remaining vertices have no edges to lower numbered vertices is $< n(1-p)^{2\psi} = n^{-\varepsilon} \to 0$. This completes the proof of the first part of Theorem 7.2.

Now let $\varepsilon < 0$ and let $G' \in \Psi_n^c(\psi, p)$. Clearly, $P(G \text{ is connected}) \leqq P(G' \text{ is connected})$ and since by Lemma 7.1, G' is almost always disconnected, it follows that G is almost always disconnected. □

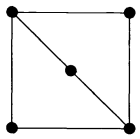
A simple lower bound for the bandwidth of any connected graph is given by

$$\phi(G) \geqq \omega(G) = \left\lceil \frac{n-1}{D(G)} \right\rceil$$

since the first and last vertices in any optimal layout are connected by a path of length at most $D(G)$ and hence at least one edge in this path has length $\geqq \omega(G)$. Chvatal [4] was apparently the first to notice this. A more general lower bound is given by

$$\phi(G) \geqq \omega^*(G) = \max_{G'} \omega(G')$$

where G' ranges over all connected subgraphs of G. The graph shown in Fig. 10 shows that $\omega^*(G) \neq \phi(G)$ in general. It is natural to ask if there is any constant c such that



$\phi(G) = 3 \qquad \omega^*(G) = 2$

FIG. 10. *Graph showing* $\omega^*(G) \neq \phi(G)$.

for all connected graphs $\phi(G) \leqq c\omega^*(G)$. Ronald Graham has pointed out that this is not the case. The argument is given in [11]. In spite of this result however, we can show that if $\ln n = o(\psi)$, then for almost all $G \in \Psi_n(\psi, p)$,

$$D(G) < (1+\varepsilon)\frac{n}{\phi(G)} + 5.$$

THEOREM 7.3. *Let* $\varepsilon > 0, 0 < p < 1$ *be fixed,* $\alpha = (1+\varepsilon)\lambda(1-p^2) \leqq \psi < n$. *For almost all* $G \in \Psi_n(\psi, p)$, $D(G) < n/(\psi - \alpha/3) + 5$.

Proof. By Lemma 3.7, there exists a path $Q = (v_0, \cdots, v_{3r})$ that satisfies

$$v_0 = 1, \quad v_{3r} \geqq n - 3\psi, \quad v_{3(i+1)} \geqq v_{3i} + 3\psi - \alpha, \quad 0 \leqq i \leqq r.$$

Since $n \geqq v_{3r}$ and $v_{3r} > 3\psi r - \alpha r$, $3r < n/(\psi - \alpha/3)$.

By Lemma 3.2, any vertex $u < v_{3r}$ is connected by a 2-path to some vertex in $\{v_1, \cdots, v_{3r}\}$, and any vertex $u > v_{3r}$ is connected by a 4-path to v_{3r}. Thus, every pair of vertices is joined by a path of length at most $3r+5$. □

By Lemma 7.2, if $p \geqq \varepsilon > 0$ then for almost all $G \in \Gamma_n(p)$, $D(G) = 2$. When p is allowed to approach zero as n gets large the diameter can become larger. By Theorem 7.1, when p is much less than $(\ln n)/n$ the graph is likely to be disconnected. We now consider the probable diameter of random graphs in $\Gamma_n(p)$ when $p = (c \ln n)/n$ and

c is a constant. We do this by examining the probable size of $V_1, V_2, \cdots$. Let $n_i = |V_i|$. Clearly,

$$n_0 = 1,$$
$$n_2 \in \mathbf{B}(n-1, p),$$
$$n_3 \in \mathbf{B}(n-(n_1+1), 1-(1-p)^{n_1}),$$
$$\vdots$$
$$n_{k+1} \in \mathbf{B}(n-s_k, 1-(1-p)^{n_k})$$

where $s_k = \sum_{j=0}^{k} n_j$. Define $\hat{n}_0 = 1$, $\hat{n}_{k+1} = (n-\hat{s}_k)(1-(1-p)^{\hat{n}_k})$, where $\hat{s}_k = \sum_{j=0}^{k} \hat{n}_j$. We can use $\hat{n}_k$ as an estimator for n_k. Figure 11 gives values of $\hat{n}_k$ for particular values of n and p. The sequence grows very rapidly until a large fraction of the vertices in the graph has been "captured". Then the remaining vertices are taken in the last step. The figure also gives values of the function $(np)^k$. For $k \leqq 3$, $(np)^k$ gives an excellent estimate for $\hat{n}_k$.

k	$\hat{n}_k$	$(np)^k$
0	1	1
1	28	28
2	763	763
3	20,850	21,096
4	428,450	582,890
5	549,910	

FIG. 11. *Comparison of $\hat{n}_k$ with $(np)^k$ for $n = 10^6$, $p = (2 \ln n)/n$.*

Let k^* be such that $s_{k^*} = n$. In the following we show that for $k \leqq k^* - 2$, $n_k > (np/8)^k$ with high probability. We can use this to get a probabilistic upper bound on k^* and hence on $D(G)$. The main results are

THEOREM 7.4. *Let $c > 8$ be fixed, $p = (c \ln n)/n$, $\gamma = np/8$. For almost all $G \in \Gamma_n(p)$, $1 \leqq k \leqq k^* - 2 \Rightarrow n_k > \gamma^k$.*

THEOREM 7.5. *Let $c > 8$ be fixed, $p = (c \ln n)/n$, $\gamma = np/8$. For almost all $G \in \Gamma_n(p)$,*

$$D(G) \leqq 2\left(\left\lceil \frac{\ln (1/p)}{\ln \gamma} \right\rceil + 2\right).$$

The proof of Theorem 7.4 is contained in the following lemmas.

LEMMA 7.3. *Let $c > 8$ be fixed, $p = (c \ln n)/n$, $\gamma = np/8$. For almost all $G \in \Gamma_n(p)$, $1 \leqq k \leqq k^* - 2 \wedge n_{k-1} < 1/p \wedge s_{k-1} \leqq n/2 \Rightarrow n_k > \gamma n_{k-1}$.*

Proof. Since $n_k \in \mathbf{B}(n - s_{k-1}, 1-(1-p)^{n_{k-1}})$,

$$\bar{n}_k = E(n_k) = (n - s_{k-1})(1-(1-p)^{n_{k-1}}) \geqq \frac{n}{2} p n_{k-1}(1 - p n_{k-1}/2) > \frac{np}{4} n_{k-1} = 2\gamma n_{k-1}.$$

By Proposition 3.1

$$P(n_k \leqq \gamma n_{k-1}) \leqq P(n_k \leqq \bar{n}_k/2) < e^{-\bar{n}_k/8} < e^{-\gamma n_{k-1}/4}.$$

Let A_k denote the event $n_k \leqq \gamma n_{k-1}$. The probability that there exists a k satisfying the

hypothesis of the lemma, such that A_k holds is

$$\leqq P(A_1)+P(A_2|\bar{A}_1)+\cdots+P(A_{k^*-2}|\bar{A}_1\cdots\bar{A}_{k^*-3})$$
$$\leqq e^{-\gamma/4}+e^{-\gamma^2/4}+\cdots+e^{-\gamma^{k^*-2}/4}\to 0. \qquad \square$$

LEMMA 7.4. *Let $c>8$ be fixed, $p=(c\ln n)/n$. For almost all $G\in\Gamma_n(p)$, $1\leqq k\leqq k^*-2\alpha s_{k-1}\leqq n/2\Rightarrow n_{k-1}<1/p$.*

Proof. Assume that $n_{k-1}\geqq 1/p$. Then since $n_k\in\mathbf{B}(n-s_{k-1},1-(1-p)^{n_{k-1}})$,

$$\bar{n}_k=E(n_k)=(n-s_{k-1})(1-(1-p)^{n_{k-1}})\leqq\frac{n}{2}(1-1/e)>n/4.$$

By Proposition 3.1

$$P(n_k\leqq n/8)\leqq P(n_k\leqq\bar{n}_k/2)<e^{-\bar{n}_k/8}<e^{-n/32}\to 0.$$

Hence, assume $n_k>n/8$. Then the probability that any of the remaining vertices is not adjacent to something in V_k is

$$\leqq(n-s_k)(1-p)^{n_k}<n\,e^{-np/8}=n^{1-c/8}\to 0.$$

This implies that $k^*\leqq k+1$ which is a contradiction. $\square$

LEMMA 7.5. *Let $c>8$ be fixed, $p=(c\ln n)/n$. For almost all $G\in\Gamma_n(p)$, $1\leqq k\leqq k^*-2\Rightarrow s_{k-1}\leqq n/2$.*

Proof. Assume that $s_{k-1}>n/2$ and let k' be the smallest integer such that $s_{k'}>n/2$. By Lemma 7.4, $n_{k'-1}<1/p$ and by Lemma 7.3, for all $k\leqq k'$, $n_k>\gamma n_{k-1}$, where $\gamma=np/8$. Since for large n, $\gamma>2$, we have $s_k>2s_{k-1}$ for $k\leqq k'$. Consequently $n_{k'}=s_{k'}-s_{k'-1}>s_{k'}/2>n/4$. Now, the probability that any of the vertices in $V-(V_0\cup V_1\cup\cdots\cup V_{k'})$ is not adjacent to some vertex in $V_{k'}$ is

$$<(n-s_{k'})(1-p)^{n_{k'}}<n\,e^{-np/4}=n^{1-c/4}\to 0.$$

This implies that $k^*\leqq k'+1$ which is a contradiction. $\square$

This establishes Theorem 7.4.

Proof of Theorem 7.5. Note that $D(G)\leqq 2k^*$. Let k' be the smallest integer such that $\gamma^{k'}\geqq 1/p$. Clearly $k'=\lceil\ln(1/p)/\ln\gamma\rceil$. If $k'>k^*-2$, we are done. If $k'\leqq k^*-2$ we can apply Theorem 7.4 giving $n_{k'}\geqq 1/p$. By the argument used in the proof of Lemma 7.4, this implies $k^*\leqq k'+2$. $\square$

8. Conclusions. The work reported here is part of an ongoing research effort aimed at developing better methods for evaluating the performance of heuristic algorithms for hard combinatorial problems. This is an area where the usual analytical tools often fail us, and the available results are unsatisfying. To be useful, a performance evaluation method must satisfy two basic criteria. First, it must be able to explain the practical success of popular algorithms and the differences observed between competing algorithms. Second, it should provide insight suggesting new and better algorithms, and supply a basis for making predictions about their success in practice. The ultimate utility of such a method depends on how accurately it predicts the performance of algorithms in real applications.

Worst-case analysis is inadequate for evaluating the performance of heuristics for bandwidth minimization, precisely because it fails to satisfy the criteria given above. As shown in Theorems 2.1 and 2.2, even probabilistic analysis can be of little use if one is naive in choosing the probability distribution. The key to the work reported here is in the choice of distribution. Because $\Psi_n(\psi,p)$ generates only graphs having

bandwidth $\leqq \psi$, we can explore properties that are common to most such graphs, even though they may be rare among unrestricted graphs. The success of heuristics like the level algorithms is due to the fact that they exploit these properties.

The methods used in this paper at least partially satisfy the criteria outlined above. They provide the first satisfactory analytical explanation of the practical success of the level algorithms and they provide insight leading to methods, which at least in theory are better. If the modified level algorithms fare as well in practice as they do on paper, the utility of these methods will have been demonstrated.

Acknowledgments. I want to thank my good friend and Ph.D. advisor, Hal Sudborough, who supervised the research reported here. I also wish to thank two anonymous referees whose careful reading and thoughtful comments uncovered several errors in an early draft and led to significant improvements in the presentation.

REFERENCES

[1] D. ANGLUIN AND L. G. VALIANT, *Fast probabilistic algorithms for Hamiltonian circuits and matchings*, J. Comput. System Sci., 18 (1979), pp. 155-193.

[2] K. Y. CHENG, *Minimizing the bandwidth of sparse symmetric matrices*, Computing, 11 (1973), pp. 103-110.

[3] ———, *Note on minimizing the bandwidth of sparse symmetric matrices*, Computing, 11 (1973), pp. 27-30.

[4] V. CHVATAL, *A remark on a problem of Harary*, Czechoslovak Math. J., 20 (1970), p. 95.

[5] E. CUTHILL AND J. MCKEE, *Reducing the bandwidth of sparse symmetric matrices*, in ACM National Conference Proceedings, 24, 1969, pp. 157-172.

[6] P. ERDOS AND A. RENYI, *On random graphs* I, in Publications Mathematicae, (1959), pp. 290-297.

[7] MICHAEL R. GAREY, R. L. GRAHAM, DAVID S. JOHNSON AND D. E. KNUTH, *Complexity results for bandwidth minimization*, SIAM J. Appl. Math., 34 (1978), pp. 477-495.

[8] B. MONIEN AND I. H. SUDBOROUGH, *Bandwidth problems in graphs*, in Proc. 18th Annual Allerton Conference on Communication, Control, and Computing, 1980, pp. 650-659.

[9] CHRISTOS H. PAPADIMITRIOU, *The* NP*-completeness of the bandwidth minimization problem*, Computing, 16 (1976), pp. 263-270.

[10] JAMES B. SAXE, *Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time*, Technical Report, Carnegie-Mellon Univ., Pittsburgh, 1980.

[11] JONATHAN S. TURNER, *Bandwidth and probabilistic complexity*, Ph.D. thesis, Northwestern University, Evanston, IL, 1982.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
020

THE COMPLEXITY OF THE MEMBERSHIP PROBLEM FOR TWO SUBCLASSES OF POLYNOMIAL IDEALS*

DUNG T. HUYNH†

Abstract. This paper shows that the membership problems for two subclasses of polynomial ideals are NP-hard. The first subclass is defined by bounding the number of variables ($\leqq 4$), whereas the second is defined by considering polynomials of the form $Y - M$, where Y is a variable and M is a monomial.

Key words. computational complexity, NP-hard, algebraic manipulation, membership problem, polynomial ideal, commutative semigroup

1. Introduction. One of the currently active areas of research in "Symbolic and Algebraic Manipulation" is the design and analysis of algorithms for computational problems concerning polynomial ideals, specifically for testing membership for polynomial ideals. The earliest algorithms for problems in polynomial ideal theory have been provided by Hermann [6], and subsequently improved by Seidenberg [14].

Concerning the intrinsic complexity of the membership problem for polynomial ideals, there is still no "exact" classification. Indeed, this has been stated as an open question in [12]. The best lower bound that is known is the exponential-space lower bound obtained by Cardoza, Lipton, Mayr and Meyer (cf. [4], [12]). On the other hand, for solving the polynomial ideal membership problem there is an interesting approach that is called the method of Gröbner bases [1], [2]. In spite of the simplicity of the Gröbner basis algorithm, little is known about its complexity. Recently, a double-exponential lower bound for the degrees of Gröbner bases has been announced in [13] and independently obtained by the author in [9] in connection with a lower bound proof for Church–Rosser commutative Thue systems. Our results in [9] imply that any algorithm that transforms a given polynomial ideal basis into a Gröbner basis requires double-exponential time.

An important technique that has been provided in [4], [12] and is employed again in [9], [13] is the method of reducing the computation of a double-exponentially space-bounded counter machine to the membership problem for polynomial ideals, showing its exponential space-hardness. This reduction depends on two parameters of the ideal basis: the number of variables, which grows linearly, and the form of the polynomials in the basis. In particular, if the number of variables is bounded, then the above reduction does not work. Indeed, our result in [8] shows that with a bounded number ($\geqq 6$) of variables a reduction of this kind can be at best a reduction from an exponentially space-bounded counter machine.

In view of recent efforts for showing upper and lower bounds for the degrees of Gröbner bases in the special cases of 2 variables [3], [10] and 3 variables [15], it seems interesting to show whether an intractability proof can still be carried out when the two parameters mentioned above are restricted. In this paper, we will consider 2 subclasses of polynomial ideals: (1) ideals of polynomials in 4 variables and (2) ideals of polynomials of the form $Y - M$, where Y is a variable and M is a monomial. We will show that the membership problems for these 2 subclasses of polynomial ideals are NP-hard. The proofs of these two results are entirely different from those in [4], [8], [12].

* Received by the editors June 4, 1984, and in revised form February 10, 1985.
† Computer Science Department, Iowa State University, Ames, Iowa 50011.

2. Definitions and results. In the following let $\mathbb{Q}$ denote the set of rationals, $\mathbb{N}$ the set of nonnegative integers. $\mathbb{Q}[X_1, \cdots, X_n]$ denotes the ring of polynomials in $X_1, \cdots, X_n$ with rational coefficients. In this paper we consider ideals in $\mathbb{Q}[X_1, \cdots, X_n]$. For polynomials $Q_1, \cdots, Q_s$ in $\mathbb{Q}[X_1, \cdots, X_n]$ let $\langle Q_1, \cdots, Q_s\rangle$ denote the ideal generated by $Q_1, \cdots, Q_s$.

The membership problem for polynomial ideals, denoted by MEMBER, is defined as follows.

MEMBER.

Input: Polynomials $P, Q_1, \cdots, Q_s \in \mathbb{Q}[X_1, \cdots, X_n]$.

Question: Is $P \in \langle Q_1, \cdots, Q_s\rangle$?

The first special case of MEMBER, denoted by MEMBER(1), is:

MEMBER(1).

Input: Polynomials $P, Q_1, \cdots, Q_s \in \mathbb{Q}[X_1, X_2, X_3, X_4]$

Question: Is $P \in \langle Q_1, \cdots, Q_s\rangle$?

The second special case of MEMBER is defined by restricting the form of $P, Q_1, \cdots, Q_s$.

MEMBER(2).

Input: Polynomials $Y - M, Y_1 - M_1, \cdots, Y_s - M_s \in \mathbb{Q}[X_1, \cdots, X_n]$ where $Y, Y_1, \cdots, Y_s \in \{X_1, \cdots, X_n\}$ and $M_1, \cdots, M_s$ are monomials.

Question: Is $Y - M \in \langle Y_1 - M_1, \cdots, Y_s - M_s\rangle$?

Our results are the following two theorems.

THEOREM 1. MEMBER (1) *is* NP-*hard.*[1]

THEOREM 2. MEMBER (2) *is* NP-*hard.*

Theorems 1 and 2 are proved by making use of a well-known connection between the membership problem for polynomial ideals and the uniform word problem for commutative semigroups which we briefly sketch here. For more details the reader is referred to [5] or [12].

For a finite set X let $X^{\oplus}$ denote the free commutative monoid generated by X. An element $W \in X^{\oplus}$ has the form $W = X_1^{e_1} \cdots X_n^{e_n}$, where $X = \{X_1, \cdots, X_n\}$ and $e_1, \cdots, e_n \in \mathbb{N}$.

Let $\mathcal{R} \subseteq X^{\oplus} \times X^{\oplus}$ be a finite set. Define the relation $\vdash_{\mathcal{R}}$ as follows. For $W, W' \in X^{\oplus}$, $W \vdash_{\mathcal{R}} W'$ if there is $(U_1, V_1) \in \mathcal{R}$ such that $W = UU_1$ and $W' = UV_1$ for some $U \in X^{\oplus}$. Let $\dashv\vdash_{\mathcal{R}}$ denote the symmetric closure of $\vdash_{\mathcal{R}}$. Then $W_1 \dashv\vdash_{\mathcal{R}} W_2 \dashv\vdash_{\mathcal{R}} \cdots \dashv\vdash_{\mathcal{R}} W_n$ is called a derivation. $\equiv_{\mathcal{R}}$ denotes the reflexive and transitive closure of $\dashv\vdash_{\mathcal{R}}$. Obviously, $\equiv_{\mathcal{R}}$ is a congruence relation. The semigroup presented by $\mathcal{R}$ is the factor semigroup $X^{\oplus}/_{\equiv_{\mathcal{R}}}$. $\mathcal{R}$ is also called a defining relation system. An element of $\mathcal{R}$ is called a defining relation.

The uniform word problem for commutative semigroups, denoted by UWP, is defined as follows.

UWP.

Input: A set $\mathcal{R} = \{(U_1, V_1), \cdots, (U_s, V_s)\} \subseteq X^{\oplus} \times X^{\oplus}$ and two words $W, W' \in X$.

Question: Is $W \equiv_{\mathcal{R}} W'$?

The connection between MEMBER and UWP is expressed in the following.

FACT 2.1. $W \equiv_{\mathcal{R}} W'$ iff $W - W' \in \langle U_1 - V_1, \cdots, U_s - V_s\rangle$.

Proof. See [5] or [12]. □

Remark 2.2. In [12] the reader can find a combinatorial proof of the above fact, whereas [5] contains an algebraic one. Notice that in [5] the proof is carried out for ideals with an arbitrary commutative ring (with 1) as coefficient ring. Therefore,

[1] The reader is referred to [7] for complexity-theoretic notions.

Theorems 1 and 2 are valid even when $\mathbb{Q}$ is replaced by any commutative ring (with 1). On the other hand, several degree bound results for polynomial ideal problems require that the coefficient ring be a field satisfying certain conditions (cf. e.g. [14]). To simplify our discussions we choose, as in [12], $\mathbb{Q}$ as coefficient field.

Using this above equivalence between MEMBER and UWP, we prove Theorem 1 and Theorem 2 by constructing two log-space reductions from the satisfiability problem, which is well known to be NP-hard (cf. [7]), to MEMBER (1) and MEMBER (2), respectively.

Remark 2.3. We need to define the size of an input instance. We encode the inputs of our problems in a natural way. According to [12], the coefficients and exponents are encoded in binary notation without leading zeros. We will see that Theorem 1 depends on this encoding, whereas Theorem 2 holds even when the exponents are encoded in unary notation.

3. Proof of Theorem 1. Let SAT denote the satisfiability problem for Boolean formulas in conjunctive normal form. We construct a log-space reduction of SAT to MEMBER (1).

We reduce SAT to UWP so that the corresponding instance of UWP has only four generating symbols. This instance of UWP, formulated as an instance of MEMBER, is an instance of MEMBER (1).

Since $\{X_1, \cdots, X_4\}^{\oplus}$ is isomorphic to $\mathbb{N}^4$, we may illustrate the idea of the construction geometrically in the lattice $\mathbb{N}^4$, and elements of $\{X_1, \cdots, X_4\}^{\oplus}$ are regarded as points in $\mathbb{N}^4$. A defining relation is then a pair of points in $\mathbb{N}^4$.

Let $F = E_1 \wedge \cdots \wedge E_m$ be a Boolean formula in conjunctive normal form, where $\{y_1, \cdots, y_n\}$ is the set of Boolean variables occurring in F and $E_1, \cdots, E_m$ are clauses of literals.[2]

Given F we want to construct a defining relation system $\mathfrak{R} \subseteq \mathbb{N}^4 \times \mathbb{N}^4$ and two points $P, P' \in \mathbb{N}^4$ such that F is satisfiable iff $P \equiv_{\mathfrak{R}} P'$. Furthermore, the reduction should be computable on a log-space-bounded Turing machine.

Remark 3.1. The following constructions are technically quite complicated. This is so, because $\equiv_{\mathfrak{R}}$ is a symmetric relation. Note that the reduction to the word problem for commutative semi-Thue systems (instead of commutative Thue systems or equivalently commutative semigroups) is quite straightforward.

We first introduce some notation. Let b be an integer such that

$$b = 2^{\lceil \log_2(n+1) \rceil}.$$

(The choice of b will be clear later.)

For every literal $z \in \{y_1, \cdots, y_n, \bar{y}_1, \cdots, \bar{y}_n\}$ define

$$v_k(z) := \begin{cases} 1 & \text{if } z \text{ occurs in } E_k, \\ 0 & \text{otherwise,} \end{cases}$$

where $k = 1, \cdots, m$. ($v_k(z)$ encodes the occurrence of z in E_k.) To encode the occurrence of z in F define

$$w(z) := \sum_{k=1}^{m} v_k(z) \cdot b^k.$$

(To see how the encoding works, the reader may find it helpful to write out $w(z)$ and

[2] We assume w.l.o.g. that no variable and its negation occur simultaneously in a clause, since otherwise such a clause can be eliminated.

other integers occurring in the following constructions in b-ary notation. Notice that the integers do not have the b^0 terms. We do so for notational convenience only. When referring to a b^i-term of some b-ary integer, we mean $i>0$.)

In the following we give the construction of the finite set of defining relations $\mathcal{R}\subseteq\mathbb{N}^4\times\mathbb{N}^4$. The construction consists of two parts. Part 1 has two finite sets of defining relations: the second set is used to derive a point in $\mathbb{N}^4$ which represents a "nondeterministic" selection of Boolean values for the variables $y_1,\cdots,y_n$ (such information can be obtained when the second component of this point is written in b-ary notation), whereas the first set is used to select nondeterministically integer values for certain "normalization". These two nondeterministic selections are represented by a point in $\mathbb{N}^4$, say P'', that is derived from P by applying the defining relations in the first set and then those in the second one.

Part 2 of the construction provides two finite sets of defining relations: the first set corresponds to the second set of Part 1 and the second set corresponds to the first set of Part 1. These are used to verify whether the selection of Boolean values for $y_1,\cdots,y_n$ by applying relations in Part 1 provides an assignment that satisfies the formula F. Indeed, we will see that P'' (derived from P by using defining relations from Part 1) represents an assignment that satisfies F iff P'' derives P' using defining relations from Part 2. (Figures 1-4 show the effects of the defining relations.)

We need some abbreviations.

$$a := \sum_{k=1}^{m} b^k,$$

$$d_{k,j} := b^{(k-1)n+(j+1)}, \quad k=1,\cdots,m, \quad j=0,\cdots,n-1,$$

$$c_i := b^{mn+i}, \qquad i=1,\cdots,2n.$$

(The choice of these constants will be clear later.)

Part 1 of the construction. This part consists of two steps. As described above, the idea is as follows. Let P denote a point in $\mathbb{N}^4$ to be defined later. We will construct two finite sets of defining relations, in (3.1) and (3.2) respectively, such that a derivation starting at P that applies defining relations in (3.1) and (3.2) successively will reach a point (P'') that represents a selection of Boolean values for $y_1,\cdots,y_n$ (obtained from (3.2)) and a selection of an m-tuple of integers in $\{0,\cdots,n-1\}$ (obtained from (3.1)). These two selections are represented in the second component of P''. (Note that the construction is done in such a way that all the points congruent to P are incomparable.[3])

In the following we will use b^i-terms, $1\leqq i\leqq mn$, to encode a selection of an m-tuple of integers in $\{0,\cdots,n-1\}$, and b^i-terms, $mn+1\leqq i\leqq mn+2n$, to encode a selection of Boolean values for $y_1,\cdots,y_n$.

Let max be the integer

$$\max := \sum_{\substack{1\leqq k\leqq m\\ 0\leqq j\leqq n-1}} d_{k,j} + \sum_{i=1}^{2n} c_i = \sum_{j=1}^{mn+2n} b^j$$

and max′ be the integer

$$\max' := na = \sum_{k=1}^{m} nb^k.$$

[3] We define the partial order $\leqq$ on $\mathbb{N}^4$ componentwise.

Let P be the point $P=(\max, 0, \max', 0)\in\mathbb{N}^4$ and consider the line

$$L:=\{(p, q, \max', 0)\in\mathbb{N}^4 \mid p+q=\max\}.$$

We construct defining relations that are applicable at selected points on L. Furthermore, when projected on the first two components, the line that connects two points of a defining relation is parallel to L. (cf. Figs. 1-4).

Step 1 *of Part* 1 (cf. Fig. 1).

For all $k=1, \cdots, m$ and $j=0, \cdots, n-1$ define the relations:

$$((\max -p_k, q_k, \max', 0), (\max -p_k-d_{k,j}, q_k+d_{k,j}, \max', 0)), \tag{3.1}$$

where

$$p_k=\sum_{l=1}^{k-1} d_{l,n-1}, \quad q_k=\sum_{l=1}^{k-1} d_{l,0}, \quad \text{i.e. } p_k=\sum_{l=1}^{k-1} b^{ln} \text{ and } q_k=\sum_{l=1}^{k-1} b^{(l-1)n+1}.$$

Remark 3.2. (1) Notice that for a fixed k, the defining relations defined for $j=0, \cdots, n-1$ have $\max-p_k, q_k$ in the first and second components. Therefore, on L, the points that can derive another one by applying one of these defining relations must have first component $\geqq \max-p^k$ and second component $\geqq q_k$ (cf. Fig. 1).

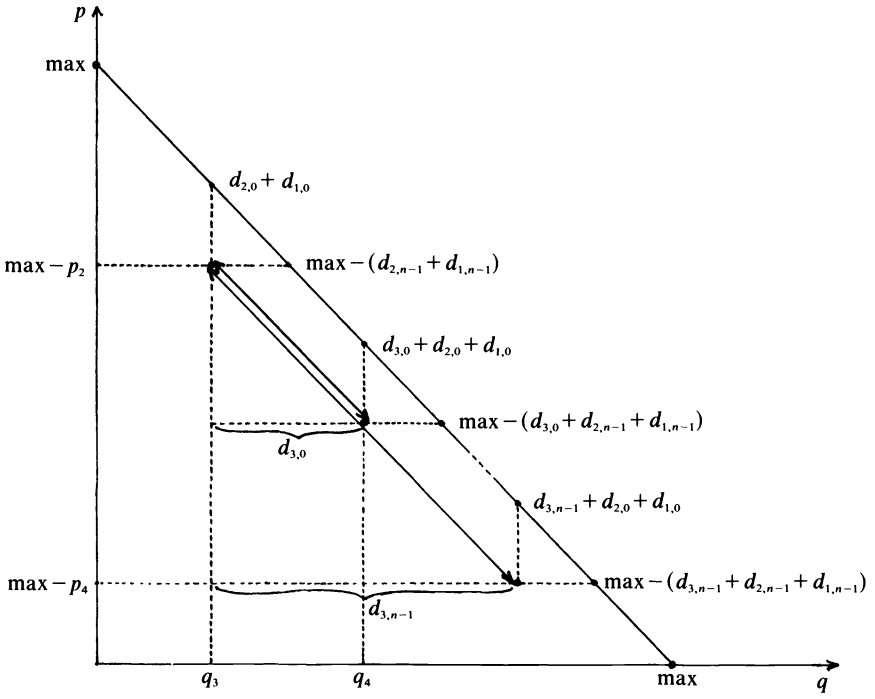


FIG. 1. *Illustration of* (3.1) *for* $k=3$.

(2) Consider a derivation starting at P that applies for $k=1, \cdots, m$ a defining relation defined by j_k, $0\leqq j_k\leqq n-1$, successively. Then the second component of the final point in this derivation has the b-ary representation

$$\sum_{k=1}^{m} d_{k,j_k}=\sum_{k=1}^{m} b^{(k-1)n+(j_k+1)}.$$

(This corresponds to a nondeterministic selection of the m-tuple $(j_1, j_2, \cdots, j_m)$, which

will be used in Part 2 to verify that each of the m clauses in F is satisfied. The values p_k and q_k ensure that exactly 1 value j_k is selected for each k.)

Step 2 *of Part* 1 (cf. Fig. 2). Let low, high denote the integers

$$\text{low} := \sum_{k=1}^{m} d_{k,0} = \sum_{k=1}^{m} b^{(k-1)n+1},$$

$$\text{high} := \sum_{k=1}^{m} d_{k,n-1} = \sum_{k=1}^{m} b^{kn}.$$

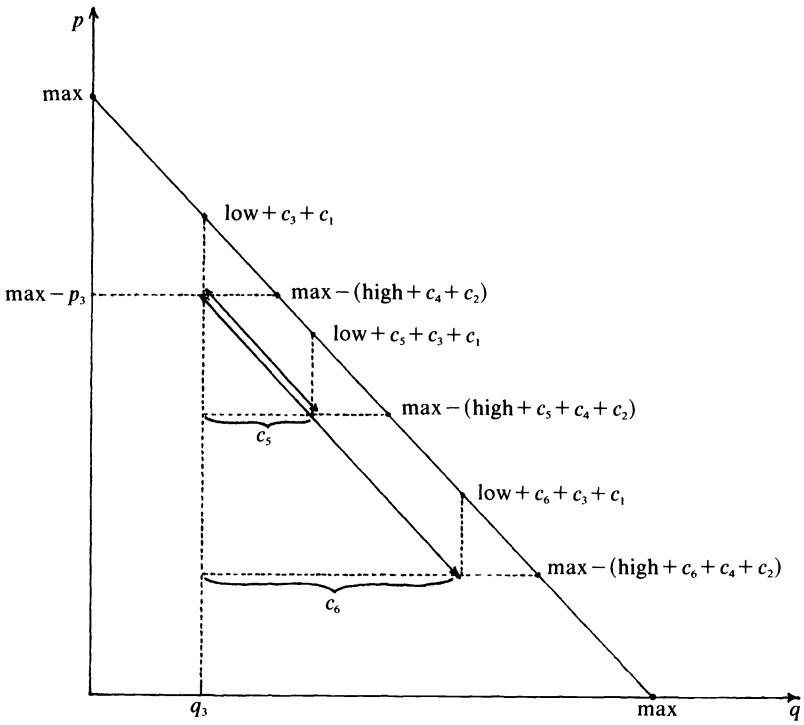


FIG. 2. *Illustration of* (3.2) *for* $i = 3$.

For $i = 1, \cdots, n$ define the following defining relations:

$$((\max - p_i, q_i, \max', 0), (\max - p_i - c_{2i-\delta}, q_i + c_{2i-\delta}, \max', 0)), \tag{3.2}$$

where

$$p_i = \text{high} + \sum_{j=1}^{i-1} c_{2j} = \sum_{k=1}^{m} b^{kn} + \sum_{j=1}^{i-1} b^{mn+2j},$$

$$q_i = \text{low} + \sum_{j=1}^{i-1} c_{2j-1} = \sum_{k=1}^{m} b^{(k-1)n+1} + \sum_{j=1}^{i-1} b^{mn+(2j-1)},$$

and $\delta \in \{0, 1\}$.

Remark 3.3. The idea of construction (3.2) is similar to the one in the previous construction. Consider the derivation in Remark 3.2 (2). If we continue that derivation by applying successively for $i = 1, \cdots, n$, one of the 2 defining relations defined for i, say the one that corresponds to $\delta_i \in \{0, 1\}$, then the second component of the final

point, denoted by P'', has the b-ary representation

$$\sum_{k=1}^{m} d_{k,j_k} + \sum_{i=1}^{n} c_{2i-\delta_i} = \sum_{k=1}^{m} b^{(k-1)n+(j_k+1)} + \sum_{i=1}^{n} b^{mn+(2i+\delta_i)},$$

whose second summand represents a selection of Boolean values for $y_1, \cdots, y_n$. These Boolean values are specified by $\delta_1, \cdots, \delta_n \in \{0, 1\}$.

Part 2 of the construction. This part consists of 2 steps corresponding to the two steps of part 1. The idea is as follows. Consider b-ary integers of the form $s_1 b^1 + \cdots + s_m b^m$. Let $\alpha = (\alpha_1, \cdots, \alpha_n)$ be an assignment that satisfies F. Then

$$\sum_{i=1}^{n} w(y_i^{\alpha_i}) = e_1 b^1 + \cdots + e_m b^m$$

has the property that $1 \leqq e_k \leqq n$ for all $k = 1, \cdots, m$, where $y^0 := \bar{y}$ and $y^1 := y$. If we add to the above number, for each $k = 1, \cdots, m$, the number $(n - e_k) b^k$, then we obtain the number $nb^1 + \cdots + nb^m = n.a.$ In other words, α is an assignment that satisfies F iff there is an m-tuple $(j_1, \cdots, j_m)$ of integers in $\{0, \cdots, n-1\}$ such that

$$(*) \qquad \sum_{i=1}^{n} w(y_i^{\alpha_i}) + \sum_{k=1}^{m} j_k b^k = \sum_{k=1}^{m} nb^k = na.$$

From Remarks 3.2 (2) and 3.3, we have seen that using the defining relations in (3.1) and (3.2), a derivation starting at P has a final point whose second component, written in b-ary notation, represents a selection of an assignment α and a selection of an m-tuple $(j_1, \cdots, j_m)$. Therefore, in Part 2 of the construction we define defining relations that check whether $(*)$ is fulfilled, i.e., whether α is an assignment that satisfies F. This will be done using the last two components.

The defining relations in this part are constructed so that the point

$$P' = (0, \max, 0, \max') \in \mathbb{N}^4$$

is congruent to P iff F is satisfiable.

Step 1 of Part 2 (cf. Fig. 3). For $i = n, n-1, \cdots, 1$ define the defining relations for $\delta = 0, 1$:

$$(3.3) \qquad ((\max - p_{i,\delta}, q_{i,\delta}, r_{i,\delta}, 0), (\max - p_i, q_i, 0, r_{i,\delta}))$$

where

$$p_{i,\delta} = \text{high} + \sum_{j=2i+1}^{2n} c_j + c_{2i-\delta} + \sum_{j=1}^{i-1} c_{2j},$$

$$q_{i,\delta} = \text{low} + \sum_{j=2i+1}^{2n} c_j + c_{2i-\delta} + \sum_{j=1}^{i-1} c_{2j-1},$$

$$r_{i,\delta} = w(y_i^{\delta}),$$

$$p_i = p_{i,\delta} + c_{2i-(1-\delta)} = \sum_{k=1}^{m} b^{kn} + \sum_{j=2i}^{2n} b^{mn+j} + \sum_{j=1}^{i-1} b^{mn+2j},$$

$$q_i = q_{i,\delta} + c_{2i-(1-\delta)} = \sum_{k=1}^{m} b^{(k-1)n+1} + \sum_{j=2i}^{2n} b^{mn+j} + \sum_{j=1}^{i-1} b^{mn+2j-1}.$$

Remark 3.4. Consider P'' defined in Remark 3.3. P'' represents the selection of Boolean values $\delta_1, \cdots, \delta_n \in \{0, 1\}$. If we apply for each $i = n, n-1, \cdots, 1$, the defining relation in (3.3) that corresponds to δ_i successively, starting at P'', then we reach a

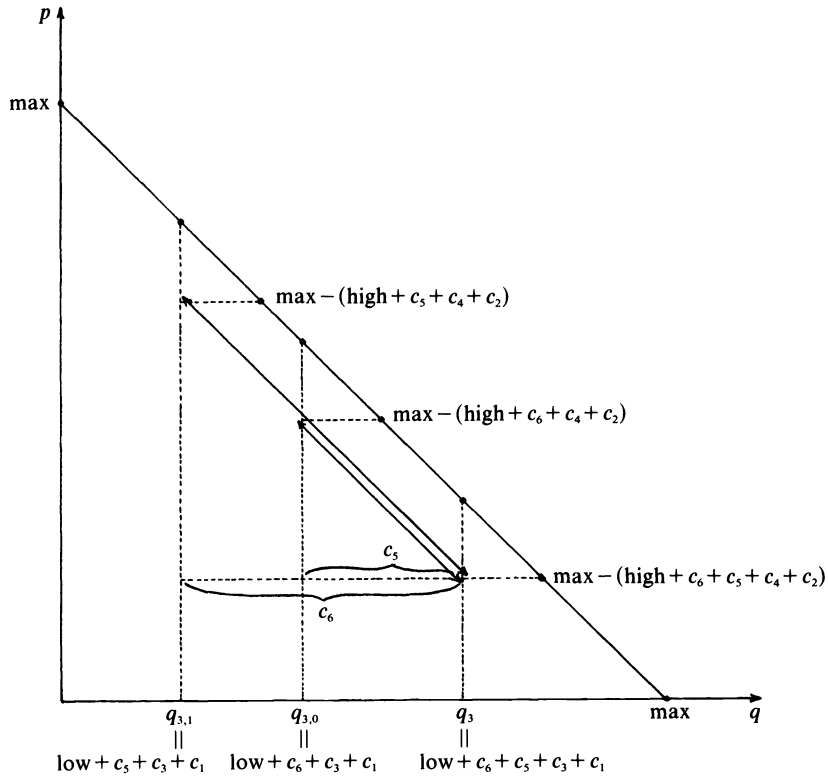


FIG. 3. *Illustration of* (3.3).

final point whose second component has b^i-terms with coefficients 1 for all $i = nm+1, \cdots, mn+2n$.

Step 2 *of Part* 2 (cf. Fig. 4). For $k = m, m-1, \cdots, 1$ and $j = n-1, \cdots, 0$ define the defining relations:

$$((\max - p_{k,j},\ q_{k,j},\ r_{k,j},\ 0),\ (\max - p_k,\ q_k,\ 0,\ r_{k,j})) \tag{3.4}$$

where

$$p_{k,j} = \sum_{j=1}^{2n} c_j + \sum_{l=k+1}^{m} \sum_{s=0}^{n-1} d_{l,s} + d_{k,j} + \sum_{l=1}^{k-1} d_{l,n-1},$$

$$q_{k,j} = \sum_{j=1}^{2n} c_j + \sum_{l=k+1}^{m} \sum_{s=0}^{n-1} k_{l,s} + d_{k,j} + \sum_{l=1}^{k-1} d_{l,0},$$

$$r_{k,j} = j \cdot b^k,$$

$$p_k = p_{k,j} + \sum_{\substack{s=0 \\ s \neq 1}}^{n-1} d_{k,s} = \sum_{l=1}^{k-1} b^{ln} + \sum_{l=k}^{m} \sum_{j=1}^{n} b^{(l-1)n+j} + \sum_{j=1}^{2n} c_j,$$

$$q_k = q_{k,j} + \sum_{\substack{s=0 \\ s \neq j}}^{n-1} d_{k,s} = \sum_{l=0}^{k-1} b^{(l-1)n+1} + \sum_{l=k}^{m} \sum_{j=1}^{n} b^{(l-1)n+j} + \sum_{j=1}^{2n} c_j.$$

Remark 3.5. Consider a continuation of the derivation in Remark 3.4 by applying the defining relation in (3.4) that corresponds to j_k for each $k = m, m-1, \cdots, 1$

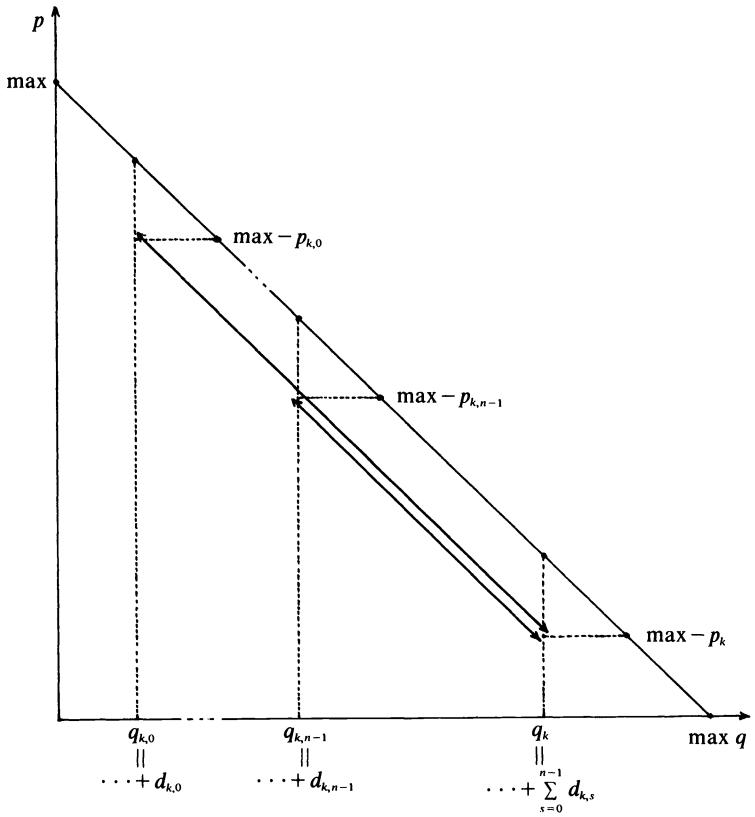


FIG. 4. *Illustration of* (3.4).

successively. Then the second component of the final point has the following b-ary representation:

$$\sum_{i=1}^{mn+2n} b^i = \max.$$

Observe that the above construction can be implemented on a log-space bounded Turing machine. We now proceed to prove the correctness of our construction.

Correctness of construction. Let $\mathcal{R}$ denote the set of relations constructed in (3.1), (3.2), (3.3) and (3.4). Further, as defined above, let

$$P = (\max, 0, \max', 0),$$

$$P' = (0, \max, 0, \max').$$

We want to show that

$$P \equiv_{\mathcal{R}} P' \text{ iff } F \text{ is satisfiable.}$$

LEMMA 3.1. *If F is satisfiable, then $P \equiv_{\mathcal{R}} P'$.*

Proof. Let $(\delta_1, \cdots, \delta_n) \in \{0, 1\}^n$ be an assignment for $y_1, \cdots, y_n$ that satisfies F. Then we have

$$\sum_{i=1}^{n} w(y_i^{\delta_i}) = e_1 b^1 + \cdots + e_m b^m$$

so that $e_k \geqq 1$ for all $k = 1, \cdots, m$. Furthermore, since no variable and its negation can occur simultaneously in a clause, we have that $1 \leqq e_k \leqq n$ for all $k = 1, \cdots, m$. Let $j_k := n - e_k$ for all $k = 1, \cdots, m$. Then it holds that

$$\sum_{i=1}^{n} w(y_i^{\delta_i}) + \sum_{k=1}^{m} j_k b^k = \sum_{k=1}^{m} nb^k = na = \max'.$$

Consider the derivation that starts at P and applies successively

- For each $k = 1, \cdots, m$, the defining relation in (3.1) that corresponds to j_k.
- For each $i = 1, \cdots, n$, the defining relation in (3.2) that corresponds to δ_i.
- For each $i = n, n-1, \cdots, 1$, the defining relation in (3.3) that corresponds to δ_i.
- For each $k = m, m-1, \cdots, 1$, the defining relation in (3.4) that corresponds to j_k.

By Remarks 3.2 (2), (3.3)-(3.5), the final point of this derivation is P'. Thus $P \equiv_{\mathcal{R}} P'$. This completes the proof of Lemma 3.1. □

We proceed to show the converse of Lemma 3.1. Because $\equiv_{\mathcal{R}}$ is symmetric, the proof of the converse of Lemma 3.1 needs some arguments.

First observe that the projection of $\mathcal{R}$ onto the first and second coordinates is a set of relations in $\mathbb{N}^2 \times \mathbb{N}^2$ which are parallel to the line $\{(p, q) \in \mathbb{N}^2 \mid p + q = \max\}$.

We divide "half" of the line L, defined above as

$$L = \{(p, q, \max', 0) \in \mathbb{N}^4 \mid p + q = \max\},$$

into "segments" $S_0, S_1, \cdots, S_m, S_{m+1}, \cdots, S_{m+n}$ as follows.

For $k = 0, \cdots, m$:

$$S_k = \{(p, q, \max', 0) \in \mathbb{N}^4 \mid p + q = \max,\ q_k \leqq q \leqq p_k\},$$

where p_k, q_k, as in (3.1), are defined by

$$p_k = \sum_{l=1}^{k} d_{l,n-1}, \qquad q_k = \sum_{l=1}^{k} d_{l,0}.$$

For $i = 1, \cdots, n$:

$$S_{m+i} = (p, q, \max', 0) \in \mathbb{N}^4 \mid p + q = \max,\ q_i \leqq q \leqq p_i\},$$

where p_i, q_i, as in (3.2), are defined by

$$p_i = \text{high} + \sum_{j=1}^{i-1} c_{2j}, \qquad q_i = \text{low} + \sum_{j=1}^{i-1} c_{2j-1}.$$

Further, define the following "segments"

$$W_{m+n}(= S_{m+n}),\ W_{m+n-1}, \cdots, W_{m+1}, W_m, \cdots, W_0$$

by:

For $i = n, n-1, \cdots, 1$:

$$W_{m+i-1} = \{(p, q, r, s) \in \mathbb{N}^4 \mid p + q = \max,\ r + s = \max',\ q_i \leqq q \leqq p_i\}$$

where q_i, p_i, as defined in (3.3), are:

$$p_i = \text{high} + \sum_{j=2i}^{2n} c_j + \sum_{j=1}^{i-1} c_{2j},$$

$$q_i = \text{low} + \sum_{j=2i}^{2n} c_j + \sum_{j=1}^{i-1} c_{2j-1}.$$

For $k = m, m-1, \cdots, 1$:

$$W_{k-1} = \{(p, q, r, s) \in \mathbb{N}^4 \mid p+q = \max,\ r+s = \max',\ q_k \leqq q \leqq p_k\}$$

where q_k, p_k, as defined in (3.4), are:

$$p_k = \sum_{j=1}^{2n} c_j + \sum_{l=k}^{m} \sum_{s=0}^{n-1} d_{l,s} + \sum_{l=1}^{k-1} d_{l,n-1},$$

$$q_k = \sum_{j=1}^{2n} c_j + \sum_{l=k}^{m} \sum_{s=0}^{n-1} d_{l,s} + \sum_{l=1}^{k-1} d_{l,0}.$$

FACT 3.2. (1) *For $k = 1, \cdots, m$, a point in S_{k-1} derives another point in S_k in one step only if exactly one of the relations in* (3.1) *is applied.*

(2) *A similar statement as* (1) *holds for S_{m+i-1}, S_{m+i}, $1 \leqq i \leqq n$, where the corresponding relation is in* (3.2).

(3) *A similar statement as* (1) *holds for W_{m+i+1}, W_{m+i}, $0 \leqq 1 \leqq n-1$, where the corresponding relation is in* (3.3).

(4) *A similar statement as* (1) *holds for W_{k+1}, W_k, $0 \leqq k \leqq m-1$, where the corresponding relation is in* (3.4).

Proof. This is clear from the construction of $\mathcal{R}$ (cf. Remark 3.2 (1)). □

We introduce some abbreviations. Consider a derivation starting at P. Such a derivation is said to cross a segment among $S_0, \cdots, S_{m+n}, W_{m+n-1}, \cdots, W_0$ if it contains a point that belongs to that segment. The segment $W_{m+n}(= S_{m+n})$ is called the critical segment.

LEMMA 3.3. *Let Q be a point in W_0. If $P \equiv_{\mathcal{R}} Q$, then there is a derivation from P to Q that crosses each segment in $S_0, \cdots, S_{m+n}, W_{m+n-1}, \cdots, W_0$ exactly once.*

Proof. Observe that in a derivation for $P \equiv_{\mathcal{R}} Q$, a subderivation from a point in S_i to a point in S_{i+k} and back to a point in S_i without crossing the critical segment must, by Fact 3.2 (1), (2) and the definition of defining relations in (3.1), (3.2), be a cycle. The same holds for a subderivation from a point in W_{i+k} to a point in W_i and back to a point in W_{i+k}.

Now consider a derivation D for $P \equiv_{\mathcal{R}} Q$. We may assume that D does not contain cycles within $S_0, \cdots, S_{m+n}$ or within $W_{m+n}, \cdots, W_0$. First observe that from the construction of $\mathcal{R}$, it follows that D must cross every segment. Assume that D cross some segment more than once. Since D does not contain cycles within $S_0, \cdots, S_{m+n}$ or within $W_{m+n}, \cdots, W_0$, it must be the case that D, starting at P, crosses the segments in the following order: $S_0, S_1, \cdots, S_i, \cdots, S_{m+n} = W_{m+n}, W_{m+n-1}, \cdots, W_k, W_{k+1}, \cdots, W_{m+n} = S_{m+n}, \cdots, S_i$ for some i, $k < m+n$. By the above discussion, the subderivation of D obtained when D crosses $S_i, \cdots, S_{m+n} = W_{m+n}, \cdots, W_k, W_{k+1}, \cdots, S_{m+n} = W_{m+n}, \cdots, S_i$, contains a cycle within $W_{m+1}, \cdots, W_0$: A contradiction to our assumption that D does not contain cycles within $S_0, \cdots, S_{m+n}$ or within $W_{m+n}, \cdots, W_0$. Thus D crosses each segment exactly once. This completes the proof of Lemma 3.3. □

We now show the converse of Lemma 3.1.

LEMMA 3.4. *If $P \equiv_{\mathcal{R}} P'$, then F is satisfiable.*

Proof. From Lemma 3.2, it follows that there is a derivation from P to P' which crosses each segment exactly once. Thus there is a unique point P'' in the critical segment such that there is a derivation of length $2(m+n)$ from P to P' that crosses the critical segment in P''.

Let the second entry of P'' be

$$\sum_{k=1}^{m} d_{k,j_k} + \sum_{i=1}^{n} c_{2i-\delta_i},$$

where $\delta_i \in \{0, 1\}$ and $j_k \in \{0, 1, \cdots, n-1\}$.

Since the fourth entries of P'' and P' are 0 and n.a, respectively, and $j_k \leqq n-1$, it follows that

$$\sum_{i=1}^{n} w(y_i^{\delta_i}) = \sum_{k=1}^{m} e_k b^k$$

with $e_k \geqq 1$ for all $k = 1, \cdots, m$.

Thus the assignment $(\delta_1, \cdots, \delta_n)$ satisfies F. □

From Lemmas 3.1 and 3.4, the correctness of the construction follows. This completes the proof of Theorem 1.

Remark 3.5. The reader should have noticed that the b^i-terms of the integers in the first and second components, that occur in the construction, have coefficients 0 or 1. Therefore, the basis b for these integers can be chosen to be 2.

Open question. Is the membership problem for polynomial ideals still NP-hard, when the number of variables is bounded by 3 or 2?

4. Proof of Theorem 2. Again we construct a log-space reduction from SAT to MEMBER (2). The idea of the construction is similar to the previous one.

As in the previous proof let $F = E_1 \wedge \cdots \wedge E_m$ be a Boolean formula in conjunctive normal form, where $\{y_1, \cdots, y_n\}$ is the set of Boolean variables occurring in F and $E_1, \cdots, E_m$ are clauses of literals.

We construct an instance of UWP, denoted by $\sigma(F)$, as follows. $\sigma(F)$ contains the symbols S, $Y_{1,0}$, $Y_{1,1}, \cdots, Y_{n,0}$, $Y_{n,1}$, $B_1, \cdots, B_n$, $C_1, \cdots, C_{m-1}$, $D_{1,l_1}, \cdots, D_{m,l_m}$, $U_1, \cdots, U_n$, $T_1, \cdots, T_m$, $V_1, \cdots, V_m$, $Z_1, \cdots, Z_m$, where $0 \leqq l_k \leqq n-1$ for $k = 1, \cdots, m$.

Again let $v_k(z) \in \{0, 1\}$ encode the occurrence of the literal z in E_k. The occurrence of z in F is encoded by

$$\gamma(z) = T_1^{v_1(z)} \cdots T_m^{v_m(z)} V_1^{1-v_1(z)} \cdots V_m^{1-v_m(z)}.$$

The set $\mathcal{R}$ of defining relations in $I(F)$ consists of:

(4.1) $(S, Y_{1,\delta}B_1)$, $\delta \in \{0, 1\}$,

(4.2) $(B_i, Y_{i+1,\delta}B_{i+1})$, $\delta \in \{0, 1\}$, $i = 1, \cdots, n-1$,

(4.3) $(B_n, D_{1,l_1}C_1)$, $0 \leqq l_1 \leqq n-1$,

(4.4) $(C_k, D_{k+1,l_{k+1}}C_{k+1})$ for $k = 1, \cdots, m-3$, $0 \leqq l_{k+1} \leqq n-1$,

(4.5) $(C_{m-2}, D_{m-1,l_{m-1}}D_{m,l_m})$, $0 \leqq l_{m-1}, l_m \leqq n-1$,

(4.6) $(Y_{i,\delta}, U_i\gamma(y_i^{\delta}))$ for $i = 1, \cdots, n$,

(4.7) $(D_{k,l_k}, T_k^{l_k} V_k^{n-1_k} Z_k)$ for $k = 1, \cdots, m$, $0 \leqq l_k \leqq n-1$.

Let $\mathcal{R}$ denote the set of the above relations. We prove:

LEMMA 4.1. $S \equiv_{\mathcal{R}} U_1 \cdots U_n T_1^n \cdots T_m^n V_1^n \cdots V_m^n Z_1 \cdots Z_m$ *if* F *is satisfiable.*

Proof. Let $\alpha = (\alpha_1, \cdots, \alpha_n)$ be an assignment such that $F(\alpha) = 1$. From the relations in (4.1) and (4.2) we have

$$S \equiv_{\mathcal{R}} Y_{1,\alpha_1} \cdots Y_{n,\alpha_n} B_n.$$

From the relations in (4.6) we have

$$Y_1, \alpha_1 \cdots Y_{n,\alpha_n} \equiv_{\mathcal{R}} U_1 \cdots U_n \gamma(y_1^{\alpha_1}) \cdots \gamma(y_n^{\alpha_n})$$
$$\equiv_{\mathcal{R}} U_1 \cdots U_n T_1^{e_1} \cdots T_m^{e_m} V_1^{f_1} \cdots V_m^{f_m}$$

where $1 \leqq e_k \leqq n$ for $k = 1, \cdots, m$, since $F(\alpha) = 1$.

Let $l_k := n - e_k$, $k = 1, \cdots, m$. Then $0 \leqq l_k \leqq n-1$. From the relations in (4.3), (4.4) and (4.5) we have

$$B_n \equiv_{\mathcal{R}} D_{1,l_1} \cdots D_{m,l_m}.$$

Now, by choosing appropriate relations in (4.7) we have

$$D_{1,l_1} \cdots D_{m,l_m} \equiv_{\mathcal{R}} T_1^{l_1} \cdots T_m^{l_m} V_1^{g_1} \cdots V_m^{g_m} Z_1 \cdots Z_m$$

with $f_k + g_k = n$ for all $k = 1, \cdots, m$. Thus we have

$$S \equiv_{\mathcal{R}} U_1 \cdots U_n T_1^n \cdots T_m^n V_1^n \cdots V_m^n Z_1 \cdots Z_m.$$

This completes the proof of Lemma 4.1. □

The proof of the converse of Lemma 4.1 needs some arguments. Let Σ denote the set of symbols in $\sigma(F)$. We define a partial order $\leqq$ on $\Sigma^{\oplus}$ by considering exponent vectors of words in $\Sigma^{\oplus}$ as usual.

Let $I \subseteq \{1, \cdots, n\}$ and $K \subseteq \{1, \cdots, m\}$. A word W in $\Sigma^{\oplus}$ is called an *(I, K)-word* if for each $i \in I$ exactly one $y_{i,\delta}$, $\delta = 0, 1$, occurs in W and for each $k \in K$ exactly one D_{k,l_k}, $0 \leqq l_k \leqq n-1$, occurs in W, and no other symbol occurs in W. A word $W \in \Sigma^{\oplus}$ is called *terminal* word if it contains only symbols in $\{U_1, \cdots, U_n, T_1, \cdots, T_m, V_1, \cdots, V_m, Z_1, \cdots, Z_m\}$. Further, a derivation step in which the left-hand side of some relation in (4.1)-(4.7) is replaced by the right-hand side is called an *expansion.* Otherwise, it is called a *contraction.*

LEMMA 4.2. *Let $I \subseteq \{1, \cdots, n\}$, $K \subseteq \{1, \cdots, m\}$. Let W_1', W_2' be two (I, K)-words. Further let W_1 and W_2 be two terminal words such that there is a derivation from W_l to W_l', $l = 1, 2$, which consists of only expansions. Then either $W_1' = W_2'$ or they are incomparable w.r.t. $\leqq$.*

Proof. Follows immediately from the construction of relations in (4.6) and (4.7). □

LEMMA 4.3. *If $S \equiv_{\mathcal{R}} W$, $W = U_1 \cdots U_n T_1^n \cdots T_m^n V_1^n \cdots V_m^n Z_1 \cdots Z_m$, then there is a derivation from S to W which consists of only expansions.*

Proof. Consider a derivation D from S to W. If D has some contractions, then there are some corresponding expansions. Using Lemma 2 this subderivation consisting of contractions followed by expansions can be deleted to shorten D. We omit the details. □

LEMMA 4.4. *Let W be as in Lemma 4.3. Then $S \equiv_{\mathcal{R}} W$ implies that F is satisfiable.*

Proof. By Lemma 4.3 there is a derivation consisting of only expansions from S to W. It follows that there is an (I, K)-word W' with $I = \{1, \cdots, n\}$, $K = \{1, \cdots, m\}$ such that there is a derivation consisting of only expansions leading from W' to W. From W' an assignment α for $\{y_1, \cdots, y_n\}$ can be obtained and $F(\alpha) = 1$. This completes the proof of Lemma 4.4. □

From Lemmas 4.1 and 4.4 the correctness of our reduction follows. On the other hand, our reduction can be implemented on a log-space bounded Turing machine. This observation completes the proof of Theorem 2.

Remark 4.4. Notice that MEMBER (2) is still NP-hard, even when the exponents of polynomials are encoded in unary.

5. Concluding remarks. In this paper we have considered the complexity of MEMBER (1) and MEMBER (2). Both problems have been shown to be NP-hard. As in [4], [12], our proofs employ the connection between the uniform word problem for commutative semigroups and the membership problem for polynomial ideals that has been briefly presented in § 2.

Concerning MEMBER (1), it is unlikely that a reduction that employs the connection between commutative semigroups and polynomial ideals can be constructed so that an essentially better lower bound for MEMBER (1) can be obtained. This is so, because the word problem for commutative semigroups with a bounded number of generating symbols is in symmetric linear space [8], [11], and hence in polynomial space. (With the same encoding of input instances as in this paper.) On the other hand, the subclass of polynomial ideals in MEMBER (2) represents a very restricted class of polynomial ideals when we consider the manifolds defined by polynomials of the form $Y - M$, where Y is a variable and M is a monomial. Although we do not have an upper bound for MEMBER (2) that is not much higher than NP, it seems unlikely that an essentially better lower bound for MEMBER (2) can be obtained. The evidence for this is that the context independence of the defining relations restricts their "simulation power" considerably.

Acknowledgments. The author wishes to thank the referees for many valuable suggestions for improving the presentation of this paper.

REFERENCES

[1] B. BUCHBERGER, *Ein Algorithmisches Kriterium für die Lösbarkeit eines Algebraischen Gleichungssystems*, Aequationes Mathematicae, 4 (1970), pp. 374–383.

[2] ———, *Gröbner bases: an algorithmic method in polynomial ideal theory*, to appear in Recent Trends in Multidimensional Systems Theory, ed. N. K. Bose.

[3] ———, *A note on the complexity of constructing Gröbner-bases*, Proc. EUROCAL'83, Lecture Notes in Computer Science 162, Springer-Verlag, New York, 1983, pp. 137–145.

[4] E. CARDOZA, R. LIPTON AND A. MEYER, *Exponential space complete problems for Petri nets and commutative semigroups*, Proc. 8th STOC, 1976, pp. 50–54.

[5] S. EILENBERG AND M. SCHÜTZENBERGER, *Rational sets in commutative monoids*, J. Algebra, 13 (1969), pp. 173–191.

[6] G. HERMANN, *Die Frage der Endlich Vielen Schritten in der Theorie der Polynomialideale*, Math. Ann., 95 (1926), pp. 736–788.

[7] J. HOPCROFT AND J. ULLMAN, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[8] D. HUYNH, *Complexity of the word problem for commutative semigroups of fixed dimension*, unpublished manuscript, 1983.

[9] ———, *A superexponential lower bound for Gröbner bases and Church-Rosser commutative Thue systems*, unpublished manuscript, 1984.

[10] D. LAZARD, *Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations*, Proc. EUROCAL '83, Lecture Notes in Computer Science 162, Springer-Verlag, New York, 1983, pp. 146–156.

[11] H. LEWIS AND C. PAPADIMITRIOU, *Symmetric space-bounded computation*, Theoret. Comput. Sci., 19 (1982), pp. 161–187.

[12] E. MAYR AND A. MEYER, *The complexity of the word problems for commutative semigroups and polynomial ideals*, Adv. Math., 46 (1982), pp. 305–329.

[13] M. MÖLLER AND F. MORA, *Upper and lower bounds for the degree of Gröbner bases*, Proc. EUROSAM'84, Lecture Notes in Computer Science 174, Springer-Verlag, New York, 1984.

[14] A. SEIDENBERG, *Constructions in algebra*, Trans. AMS, 197 (1974), pp. 273–313.

[15] F. WINKLER, *On the complexity of the Gröbner-bases algorithm over $K[x, y, z]$*, Proc. EUROSAM'84, Lecture Notes in Computer Science 174, Springer-Verlag, New York, 1984, pp. 184–194.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
021

NONCOMMUTATIVE BILINEAR ALGORITHMS FOR 3×3 MATRIX MULTIPLICATION*

RODNEY W. JOHNSON† AND AILEEN M. McLOUGHLIN‡

Abstract. New noncommutative bilinear algorithms for 3×3 matrix multiplication are presented. These have the same complexity, 23 essential multiplications, as the one discovered by Laderman, but are inequivalent to it. Equivalence here refers to a certain group of transformations all of which map noncommutative bilinear matrix-multiplication algorithms into other such algorithms; "inequivalent" means not related by a transformation in the group. This group has been studied by de Groote, who has shown for the case of 2×2 matrix multiplication with 7 essential multiplications that all such algorithms are equivalent to Strassen's. The new algorithms, by contrast, include infinitely many pairwise inequivalent algorithms. The computer search that led to the new algorithms is described.

Key words. matrix multiplication, computational complexity, Strassen's algorithm

1. Introduction. We here present new noncommutative bilinear algorithms for 3×3 matrix multiplication over the field of rational numbers—that is, we specify rational coefficients A_{ij}^r, B_{kl}^r, C_{mn}^r such that, with $N = 3$, the equation

$$\sum_{p=1}^{N} X_{np} Y_{pm} = \sum_{r=1}^{M} \left(\sum_{i,j=1}^{N} A_{ij}^r X_{ij} \right) \left(\sum_{k,l=1}^{N} B_{kl}^r Y_{kl} \right) C_{mn}^r \tag{1}$$

is an identity for $N \times N$ matrices X and Y.

The problem of finding such algorithms originally received attention because each such algorithm yields an upper bound $O(n^a)$ on the number of arithmetic operations needed for multiplication of $n \times n$ matrices, where $a = \log_N M$. Strassen's algorithm [1] ($N = 2$, $M = 7$) had been shown to be optimal for $N = 2$ [2], [3], and it seemed that one avenue to reducing the exponent a was to find algorithms for other small values of N and sufficiently small M, such as $N = 3$, $M = 21$. Recent reductions in the exponent by Pan [4], [5], [6], Bini et al. [7], [8], Schönhage [9], and Coppersmith and Winograd [10] have been achieved by rather different means with the help of several new ideas; for $N \geq 3$, it remains an interesting unsolved problem to determine the minimum complexity of noncommutative bilinear algorithms for $N \times N$ matrix multiplication.

Laderman, in [11], gave an algorithm with $M = 23$, the best value known for $N = 3$ with rational coefficients. (Unpublished work of Schönhage's [12] indicates that $M = 22$ is achievable with *complex* coefficients.) The algorithms we present here have the same complexity, $M = 23$ essential multiplications, as Laderman's but are inequivalent to it in the following sense.

Several sorts of transformations on families of coefficients A_{ij}^r, B_{kl}^r, C_{mn}^r have the property of mapping any (noncommutative, bilinear) algorithm for $N \times N$ matrix multiplication again into such an algorithm. We describe these using the notation A^r for the $N \times N$ matrix with elements A_{ij}^r, with similar definitions for B^r and C^r. The transformations include replacement of these coefficient matrices A^r, B^r, C^r by:

$$A^{\pi(r)}, \quad B^{\pi(r)}, \quad C^{\pi(r)}, \tag{2}$$

* Received by the editors October 3, 1984.

† Naval Research Laboratory, Washington, DC 20375.

‡ Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York 12180.

for some permutation π of the indices $1, \cdots, M$;

(3) $$C^r, \quad A^r, \quad B^r$$

(cyclic permutation);

(4) $$\tilde{C}^r, \quad \tilde{B}^r, \quad \tilde{A}^r,$$

where the tilde denotes transposition;

(5) $$a_r A^r, \quad b_r B^r, \quad c_r C^r,$$

where a_r, b_r, c_r are rational numbers such that $a_r b_r c_r = 1$ for $r = 1, \cdots, M$; and

(6) $$PA^rQ^{-1}, \quad QB^rR^{-1}, \quad RC^rP^{-1} \tag{6}$$

where P, Q, R are invertible $N \times N$ matrices. Two algorithms will be called *equivalent* if the coefficients of one are mapped to those of the other by a sequence of transformations of the forms (2)-(6); otherwise they are *inequivalent.*

De Groote [13], [14], [15] has studied the group generated by such transformations and has shown [14] that Strassen's algorithm is essentially unique: any algorithm with $N = 2$ and $M = 7$ is equivalent to Strassen's. (De Groote worked over an arbitrary field K, not assumed to be the rational numbers. Pan had stated the result without proof in [16] for the case of the rational numbers. Hopcroft and Musinski [17] had treated the case of the integers (mod 2).)

The algorithms presented here show that, in contrast with Strassen's algorithm, Laderman's is not essentially unique; in fact they form a 3-parameter family and a 1-parameter family that contain infinitely many pairwise inequivalent algorithms with $N = 3$ and $M = 23$. The coefficients of the first family of algorithms are shown in Table 1; the 3 parameters are x, y, and z. The coefficients for the second family are in Table 2; the parameter is x. Only for 5 values of r in each table do the entries actually depend on the parameters. We discovered the new algorithms with the help of a computer search, which we describe in the next section. In the third section we discuss the question of the algorithms' inequivalence.

2. Search procedure. A necessary and sufficient condition for (1) to hold identically in X and Y is that the coefficients satisfy

(7) $$\sum_{r=1}^{M} A_{ij}^r B_{kl}^r C_{mn}^r = \delta_{ni}\delta_{jk}\delta_{lm}.$$

Solutions of (7) correspond to zeros of

(8) $$\sum_{i,j,k,l,m,n=1}^{N} \left(\sum_{r=1}^{M} A_{ij}^r B_{kl}^r C_{mn}^r - \delta_{ni}\delta_{jk}\delta_{lm} \right)^2,$$

which is a nonnegative function of the coefficients. We sought solutions of (7) by trying to minimize (8). Although (8) is a sixth-degree polynomial, it is only quadratic in the A_{ij}^r when the other coefficients are held fixed; likewise it is quadratic as a function of the B_{kl}^r alone or of the C_{mn}^r alone. Since the minimum of a quadratic polynomial can be obtained from the solution of a set of linear equations, it is straightforward to minimize (8) with respect to any one of the three sets of coefficients separately. We wrote a program that assigns random starting values to the matrices A^r and B^r and, holding these fixed, determines values for the C^r that minimize (8). Next it determines new values for the B^r that minimize (8) with the A^r and C^r held fixed. It then minimizes with respect to the A^r with the new B^r and the C^r held fixed, minimizes with respect

TABLE 1

Coefficients for three-parameter algorithm family.

r	A^r			B^r			C^r		
	1	0	−1	1	0	0	1	0	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	0	0	0	0
2	1	1	−1	1	1	0	1	1	0
	0	0	0	0	0	0	0	0	0
	1	−1	−1	0	1	−1	0	0	0
3	−1	1	1	0	0	0	1	0	0
	1/3	1	1	0	0	0	0	0	0
	0	$1/(1+xy)$	0	0	0	0	1	z	$-(1-y)(1-z)$
4	0	0	0	1	0	x	−1	$-z$	$(1-y)(1-z)$
	0	0	0	0	0	0	y	yz	$-2(1-y)(1-z)$
	0	−1	0	0	−1	1	0	0	1
5	0	1	0	−1	−1	0	1	0	−1
	1/3	1	0	0	0	0	0	0	2
	0	0	1	0	1	−1	0	1	0
6	0	0	−1	0	0	0	0	−1	0
	0	0	−1	0	1	1	3/2	3/2	0
	0	0	0	0	0	0	0	0	2
7	0	0	0	0	1	1/2	0	0	−2
	0	−1	1	0	0	0	0	0	2
	1	0	0	0	0	−1	0	0	0
8	−1	0	0	0	0	0	−1	0	0
	0	0	0	0	0	0	−1	0	0
	0	0	1	1	0	0	1	1	0
9	0	0	0	0	0	0	−1	−1	0
	0	0	0	1	0	0	1	1	0
	0	$1/(1+xy)$	0	0	0	0	x	xz	$-(1+x)(1-z)$
10	0	0	0	y	0	−1	$-x$	$-xz$	$(1+x)(1-z)$
	0	0	0	0	0	0	−1	$-z$	$-2(1+x)(1-z)$
	0	0	0	−1	−1/3	1/3	0	3/2	−1
11	−1	0	1	−2/3	−2/3	0	−3/2	−3/2	0
	0	0	1	−1	−1	0	0	0	0
	0	0	0	0	0	0	0	0	−2
12	0	0	0	1	1	0	−1	0	1
	−1/3	−1	1	0	0	0	0	0	−2
	0	0	−2/3	1	1	−1	0	3/2	0
13	−1/3	0	1	0	0	0	−3/2	−3/2	0
	0	0	1	1	1	0	3	3	0
	0	0	0	0	0	0	0	−1	1
14	1	0	−1	1	1	0	0	0	0
	0	0	0	1	1	0	0	0	0
	0	0	−1/2	1	1/2	−1/2	0	0	0
15	0	0	3/2	0	0	0	0	0	0
	0	0	3/2	1	1/2	−1/2	−2	−2	0
	0	0	0	−1	−1/3	1/3	0	0	−1
16	1	0	−1	1/3	1/3	0	0	0	0
	1	0	−1	0	0	0	0	0	0

TABLE 1—continued

r	A^r			B^r			C^r		
	0	$-z$	0	0	0	0	0	-1	-1
17	0	1	0	-1	0	1/2	0	1	1
	0	0	1	0	0	$-1/2$	0	0	-2
	0	0	0	-1	-1	0	0	0	0
18	1	0	0	0	0	0	-1	-1	0
	0	0	0	-1	-1	0	-1	-1	0
	0	$-z$	0	0	0	0	0	$-1/2$	1/2
19	0	1	0	0	0	-1	0	1/2	$-1/2$
	0	0	0	0	0	-1	0	-1	1
	0	0	0	0	0	0	0	1	-1
20	0	0	0	0	-1	$-1/2$	0	-1	1
	0	0	-1	0	-1	$-1/2$	0	0	0
	0	0	0	0	-1	$-1/2$	0	0	0
21	0	0	0	0	0	0	0	0	$-2/3$
	1	0	0	0	0	0	0	0	$-2/3$
	0	z	-1	0	0	0	0	-1	0
22	0	-1	1	0	0	0	0	1	0
	0	0	0	0	0	-1	0	-1	0
	0	1	0	0	-1	1	0	0	1
23	0	-1	0	0	-1	-1	0	0	-1
	0	-1	0	0	0	0	0	0	2

to the C^r with the new A^r and B^r held fixed, and continues thus cyclically. This results in a decreasing sequence of values for (8); the search is considered to be successful if these appear to be converging to 0 while the coefficients converge to finite limits.

One difficulty that proved troublesome in practice was "zeros at infinity." For some searches, some of the coefficients seemed to be tending to infinity in such a way that (8) was tending to zero. This phenomenon is undoubtedly due to the existence of arbitrary-precision approximating (APA) algorithms in the sense of Bini et al. [7], which implies that there are sequences of values for the coefficients that behave as described. APA algorithms for 3×3 matrix multiplication are known to exist; indeed Schönhage [9] has constructed such an algorithm with $M=21$.

The difficulty was countered with a modification of the expression the programs were attempting to minimize; a term

$$\varepsilon \sum_{rij} ((A_{ij}^r)^2+(B_{ij}^r)^2+(C_{ij}^r)^2) \tag{9}$$

was added to (8). The coefficient ε was adjusted by trial and error, interactively, so that, if possible, the magnitudes of the coefficients would stay bounded or decrease at the same time that the value of (8) was decreasing. If a suitable value for ε could not be found, new random starting values were chosen for the coefficients and the search was begun again.

The procedure just described typically yields tables of rather arbitrary-looking floating-point numbers as values for the coefficients and a small but nonzero positive number, such as 10^{-6}, as the value of (8) computed to machine precision. One would like coefficient values that can be expressed exactly and that yield 0 as the exact value of (8). Simple rational coefficient values are therefore desirable—that is, ratios of small

TABLE 2
Coefficients for one-parameter algorithm family.

r	A^r			B^r			C^r		
	1	0	0	1	0	0	1	0	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	0	0	0	0
2	0	1	0	−1	−1	0	1	−1	0
	0	0	0	0	0	0	0	0	0
	0	−1	1	0	0	0	0	−1	−1
3	0	−1	0	1	0	0	0	1	1
	0	0	0	0	−1	1	1	0	1
	0	0	0	0	0	0	0	−1	0
4	−1	0	−1	0	0	0	0	0	0
	0	0	0	1	1	0	0	0	0
	$-x$	$x-1$	0	0	$1-2/x$	−1	0	0	0
5	0	$x-1$	0	0	1	−1	1/2	0	1/2
	0	1	0	0	1	−1	1/2	0	−1/2
	0	0	0	1	0	0	0	1	0
6	1	0	0	0	0	0	0	−1	−1
	−1/2	0	1/2	−1	−1	0	0	0	1
	−1	1	0	0	$1-x$	$x-1$	0	0	0
7	0	1	0	0	$-x$	x	0	0	0
	0	0	0	0	$-x$	x	1	0	0
	0	0	0	−1	−1	0	0	0	0
8	−1	0	0	0	0	0	0	1	1
	0	0	0	1	1	0	0	0	−1
	0	0	0	0	1	1	0	0	0
9	0	0	0	0	0	0	0	0	0
	1	−1	0	0	−1	1	0	0	1
	0	0	0	0	0	0	0	1	1
10	0	0	1	0	0	0	0	−1	−1
	0	0	0	0	0	−1	0	−1	−1
	−1	$1-1/x$	0	0	1	0	0	0	0
11	−1	$1-1/x$	0	0	0	0	0	0	1
	1	$1/x-1$	0	0	0	0	0	0	−1
	0	0	0	−1	0	0	0	1/2	1/2
12	0	0	0	0	0	0	0	−1/2	−1/2
	−1	0	1	1	0	1	0	0	1/2
	$-x$	$x-1$	0	0	1	−1	0	0	0
13	0	$x-1$	0	0	1	−1	−1/2	0	−1/2
	0	0	0	0	1	−1	1/2	0	1/2
	0	1	−1	0	0	0	1	−1	0
14	0	0	0	1	0	0	−1	1	0
	0	0	0	0	0	0	0	0	0
	0	−1	1	0	0	0	0	0	1
15	0	−1	1	0	0	0	0	0	−1
	0	1	−1	0	1	0	0	0	−1
	0	0	0	0	0	0	0	0	0
16	0	1	0	0	0	−1	0	0	0
	0	0	0	0	0	0	1	−1	0

TABLE 2—*continued*

r	A^r			B^r			C^r		
	0	1	0	0	$1-1/x$	0	0	0	0
17	0	1	0	0	1	0	1	0	0
	0	−1	0	0	1	0	1	0	0
	0	0	0	0	0	1	0	0	0
18	−1	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	−1	0
	0	0	1	0	0	0	1	0	0
19	0	0	0	1	0	0	0	0	0
	0	0	0	1	0	0	0	0	0
	0	−1	1	0	0	0	0	0	1
20	0	−1	0	1	0	0	−1	0	−1
	0	1	0	0	−1	0	−1	0	−1
	0	0	0	1	0	0	0	0	−1/2
21	0	0	0	0	0	0	0	0	0
	−1	0	−1	1	1	0	0	0	0
	0	1	−1	0	0	0	0	−1	−1
22	0	1	−1	0	0	0	0	1	1
	−1/2	0	1/2	0	−1	1	0	0	1
	0	0	0	0	−1/2	−1/2	0	0	0
23	0	0	0	−1	−1/2	−1/2	−1	0	−1
	0	1	0	0	1/2	−1/2	−1	0	−1

integers. (Ideally, one would like the coefficient values to be confined to 0 and ±1, as indeed they are for Strassen's and Laderman's algorithms.)

The minimization procedure is unlikely to lead to a simple rational solution of (7), even if one exists; with any such solution, transformations (5) and (6) associate an infinite family of equivalent solutions, most of which do not consist of simple rational numbers. We therefore wrote procedures for performing transformations of the forms (5) and (6). When the minimization procedure appeared to be converging to a zero of (8), we used these procedures in an attempt to transform the solution to a simple rational form—if possible, consisting of 1's, 0's, and −1's. We would arbitrarily choose one of the coefficient matrices—for example B^1; we would then find values for the matrices Q and R in (6) and the scalars a_1, b_1, and c_1 in (5) that would transform B^1 to diagonal form with 1's and 0's on the diagonal, as in Table 1. Enough freedom remained in the choice of P, Q, R, and the scalars to permit further simplifications, such as the transformation of A^1 and C^1 and the coefficients for $r=2$ to the form shown in the table. When that freedom had been exhausted, the coefficients for several values of r had attained the values shown in the table to within $\pm 10^{-3}$. We altered these coefficients, replacing the approximate rational values by exact rational values (to machine precision). For other values of r, the coefficients were not close to any obvious simple rational values. In Table 1 these included $r=4$, 10, 17, 19, 22: the 5 rows of the table that show dependence on the parameters x, y, and z.

We wrote a version of the search procedure that would minimize (8) with respect to the coefficients for selected values of r only, holding all coefficients fixed for the remaining r. We held fixed the coefficients that we had set to rational values and found that the search still seemed to be converging to a zero of (7). Moreover, by arbitrarily

perturbing some of the nonfixed coefficients, we found it possible to guide the search so that additional coefficients tended toward rational values. We could thus simplify additional rows of the table, replace approximate rational values by exact, and add to the list of values of r for which the coefficients were held fixed. The 5 parameter-dependent rows were the last to be simplified. We found that we could vary certain of the coefficients in these rows practically at will and still obtain good numerical solutions to (7) with the coefficients in the other rows held fixed. This observation led to the parametrized family explicitly presented in Table 1. The same procedure, with different random starting values for the search, led to the family shown in Table 2.

Similar, earlier computer searches for minima of (8) were undertaken by Brent [18], Ungar [19], and Lafon [20], all of whom report successes with $M=7$, $N=2$: computer runs for which (8) became small and appeared to be converging to 0. Except for the ε term (9), the minimization procedure we have described is basically identical to Brent's [18]. Brent reported apparent success with $N=3$, $M=25$, but did not try to simplify the solution by equivalence transformations such as (6), (7). Schönhage [12], by admitting complex coefficients, has obtained numerical solutions (unsimplified) of (7) with $N=3$, $M=22$.

3. Inequivalence. For comparison with Tables 1 and 2, the coefficients of Laderman's algorithm are shown in Table 3.

We have claimed that none of the new algorithms is equivalent to Laderman's algorithm. To prove this, we point out that, except for permutations, the transformations (2)-(6) leave the ranks of the matrices A^r, B^r, and C^r unchanged. Six matrices in Table 3 (A^1 and B^3, for instance) have rank 3. But regardless of the parameters, all the matrices in Table 1 have rank 1 or 2, and just one matrix (B^{23}) in Table 2 has rank 3. Therefore, no combination of transformations (2)-(6) can change the coefficients in Table 3 into those given in Table 1 or Table 2. That is, the algorithm presented in Table 3 is not equivalent to any algorithm of the families presented in Tables 1 and 2.

Similarly, the algorithms of the family in Table 1 are inequivalent to those of Table 2.

Next we consider whether distinct algorithms within a family are equivalent. Let (x', y', z') and (x'', y'', z'') be distinct triples of values for the parameters in Table 1. If

$$x' = -1/y'', \quad y' = -1/x'', \quad z' = z'', \tag{10}$$

then the corresponding algorithms are equivalent by a permutation (2) that interchanges $r=4$ and $r=10$ together with an obvious scaling (5). Otherwise the algorithms are inequivalent. In Table 2, distinct values x' and x'' of the parameter always correspond to inequivalent algorithms.

The proof of these statements is entirely elementary, and too tedious to give in full. We merely sketch the ideas involved. For definiteness, consider the family of Table 2, and write $A(x)$, $B(x)$, $C(x)$ to show the dependence of the coefficients on the parameter.

Let x' and x'' be values of the parameter, and suppose the algorithm with coefficients $A(x')$, $B(x')$, $C(x')$ is equivalent to that with coefficients $A(x'')$, $B(x'')$, $C(x'')$; we need to show that $x'=x''$. For some permutation π, numbers a_r, b_r, c_r, and invertible matrices P, Q, R, one of six sets of equations holds:

$$\begin{aligned} A^r(x') &= a_r P A^{\pi(r)}(x'') Q^{-1}, \\ B^r(x') &= b_r Q B^{\pi(r)}(x'') R^{-1}, \\ C^r(x') &= c_r R C^{\pi(r)}(x'') P^{-1}, \end{aligned} \tag{11}$$

TABLE 3
Coefficients for Laderman's algorithm.

r	A^r	B^r	C^r	r	A^r	B^r	C^r
	1 1 1	0 0 0	0 0 0		0 0 1	0 0 0	0 0 1
1	−1 −1 0	0 1 0	1 0 0	13	0 0 0	0 1 0	0 0 1
	0 −1 −1	0 0 0	0 0 0		0 0 −1	0 −1 0	0 0 0
	1 0 0	0 −1 0	0 1 0		0 0 1	0 0 0	1 1 1
2	−1 0 0	0 1 0	0 1 0	14	0 0 0	0 0 0	1 0 1
	0 0 0	0 0 0	0 0 0		0 0 0	1 0 0	1 1 0
	0 0 0	−1 1 0	0 1 0		0 0 0	0 0 0	0 0 0
3	0 1 0	1 −1 −1	0 0 0	15	0 0 0	0 0 0	1 0 1
	0 0 0	−1 0 1	0 0 0		0 1 1	−1 1 0	0 0 0
	−1 0 0	1 −1 0	0 1 0		0 0 −1	0 0 0	0 1 0
4	1 1 0	0 1 0	1 1 0	16	0 1 1	0 0 1	0 0 0
	0 0 0	0 0 0	0 0 0		0 0 0	1 0 −1	1 1 0
	0 0 0	−1 1 0	0 0 0		0 0 1	0 0 0	0 1 0
5	1 1 0	0 0 0	1 1 0	17	0 0 −1	0 0 1	0 0 0
	0 0 0	0 0 0	0 0 0		0 0 0	0 0 −1	0 1 0
	1 0 0	1 0 0	1 1 1		0 0 0	0 0 0	0 0 0
6	0 0 0	0 0 0	1 1 0	18	0 1 1	0 0 0	0 0 0
	0 0 0	0 0 0	1 0 1		0 0 0	−1 0 1	1 1 0
	−1 0 0	1 0 −1	0 0 1		0 1 0	0 0 0	1 0 0
7	0 0 0	0 0 1	0 0 0	19	0 0 0	1 0 0	0 0 0
	1 1 0	0 0 0	1 0 1		0 0 0	0 0 0	0 0 0
	−1 0 0	0 0 1	0 0 1		0 0 0	0 0 0	0 0 0
8	0 0 0	0 0 −1	0 0 0	20	0 0 1	0 0 0	0 1 0
	1 0 0	0 0 0	0 0 1		0 0 0	0 1 0	0 0 0
	0 0 0	−1 0 1	0 0 0		0 0 0	0 0 1	0 0 0
9	0 0 0	0 0 0	0 0 0	21	1 0 0	0 0 0	0 0 0
	1 1 0	0 0 0	1 0 1		0 0 0	0 0 0	0 1 0
	1 1 1	0 0 0	0 0 0		0 0 0	0 1 0	0 0 0
10	0 −1 −1	0 0 1	0 0 0	22	0 0 0	0 0 0	0 0 1
	−1 −1 0	0 0 0	1 0 0		1 0 0	0 0 0	0 0 0
	0 0 0	−1 0 1	0 0 1		0 0 0	0 0 0	0 0 0
11	0 0 0	1 −1 −1	0 0 0	23	0 0 0	0 0 0	0 0 0
	0 1 0	−1 1 0	0 0 0		0 0 1	0 0 1	0 0 1
	0 0 −1	0 0 0	0 0 1				
12	0 0 0	0 1 0	1 0 1				
	0 1 1	1 −1 0	0 0 0				

or one of five similar sets obtained from these by permutations of $A(x'')$, $B(x'')$, $C(x'')$, on the right, possibly with transposition (cf. (3), (4)). The steps of the proof are to show that (11) holds; that P, Q, and R are scalar multiples of the identity matrix; that $\pi(r) = r$ for all r; and finally that $x' = x''$.

Inspection of a tabulation of the ranks rk A^r, rk B^r, rk C^r of the matrices in Table 2 suffices for a proof that (11) holds.

Consideration of the tabulated ranks also permits a proof, for a few particular values of r, that $\pi(r) = r$. We obtain further information by considering inclusion relations between row spaces or between column spaces of matrices. For instance we

have col $C^9(x') \subseteq$ col $C^{22}(x')$, which by (11) implies col $C^{\pi(9)}(x'') \subseteq$ col $C^{\pi(22)}(x'')$. From a listing of such inclusion relations, we can show for a number of additional values of r that $\pi(r) = r$. Now when r is such that $\pi(r) = r$ and $C^r(x') = C^r(x'')$, it follows from (11) that the column space col $C^r(x')$ is an invariant subspace for R. It is possible to identify in this way enough invariant subspaces for R to find 4 eigenvectors of R, no 3 of which are linearly dependent. From this it follows that R is a scalar multiple of the identity. A similar analysis shows that P and Q are likewise scalar multiples of the identity.

It now follows from (11) that $A^{\pi(r)}(x'')$, $B^{\pi(r)}(x'')$, and $C^{\pi(r)}(x'')$ are scalar multiples of $A^r(x')$, $B^r(x')$, and $C^r(x')$. For all r this implies $\pi(r) = r$. Then $A^5(x'')$ is a scalar multiple of $A^5(x')$; it follows that $x' = x''$.

REFERENCES

[1] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354-356.

[2] J. E. HOPCROFT AND L. R. KERR, *On minimizing the number of multiplications necessary for matrix multiplication*, SIAM J. Appl. Math., 20 (1971), pp. 30-36.

[3] S. WINOGRAD, *On multiplication of* 2×2 *matrices*, Linear Algebra and Appl., 4 (1971), pp. 381-388.

[4] V. YA. PAN, *Strassen's algorithm is not optimal—trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations*, Proc. 19th Annual Symposium on Foundations of Computer Science, Oct. 1978, pp. 166-176.

[5] ——, *Field extension and trilinear aggregating, uniting and canceling for the acceleration of matrix multiplications*, Proc. 20th Annual Symposium on Foundations of Computer Science, 1979.

[6] ———, *New fast algorithms for matrix operations*, this Journal, 9 (1980), pp. 321-342.

[7] D. BINI, M. CAPOVANI, F. ROMANI AND G. LOTTI, $O(n^{2.7799})$ *complexity for* $n \times n$ *approximate matrix multiplication*, Inform. Proc. Lett., 8 (1979), pp. 234-235.

[8] D. BINI, G. LOTTI AND F. ROMANI, *Approximate solutions for the bilinear form computational problem*, this Journal, 9 (1980), pp. 692-697.

[9] A. SCHÖNHAGE, *Partial and total matrix multiplication*, this Journal, 10 (1981), pp. 434-455.

[10] D. COPPERSMITH AND S. WINOGRAD, *On the asymptotic complexity of matrix multiplication*, this Journal, 11 (1982), pp. 472-492.

[11] J. LADERMAN, *A noncommutative algorithm for multiplying* 3×3 *matrices using* 23 *multiplications*, Bull. Amer. Math. Soc., 82 (1976), pp. 126-128.

[12] A. SCHÖNHAGE, private communication.

[13] H. F. DE GROOTE, *On varieties of optimal algorithms for the computation of bilinear mappings.* I: *The isotropy group of a bilinear mapping*, Theoret. Comp. Sci., 7 (1978), pp. 1-24.

[14] ———, *On varieties of optimal algorithms for the computation of bilinear mappings.* II: *Optimal algorithms for* 2×2 *matrix multiplication*, Theoret. Comp. Sci., 7 (1978), pp. 127-148.

[15] ———, *On varieties of optimal algorithms for the computation of bilinear mappings.* III: *Optimal algorithms for the computation of xy and yx where* $x, y \in M_2(K)$, Theoret. Comp. Sci., 7 (1978), pp. 239-249.

[16] V. YA. PAN, *On schemes for computation of the product of matrices and the inverse of a matrix*, Uspekhi Mat. Nauk, 27, 5 (1972), pp. 249-250. (In Russian.)

[17] J. HOPCROFT AND J. MUSINSKI, *Duality applied to the complexity of matrix multiplication and other bilinear forms*, this Journal, 2 (1973), pp. 159-173.

[18] R. P. BRENT, *Algorithms for matrix multiplication*, Tech. Rep. CS 157, Computer Science Dept., Stanford Univ., Stanford, CA, March 1970.

[19] P. UNGAR, private communication.

[20] J. C. LAFON, Ph.D. dissertation, Univ. of Grenoble.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
022

COLLECTIONS OF FUNCTIONS FOR PERFECT HASHING*

FRANCINE BERMAN†, MARY ELLEN BOCK‡, ERIC DITTERT†, MICHAEL J. O'DONNELL† AND DARRELL PLANK§

Abstract. Hashing techniques for accessing a table without searching it are usually designed to perform efficiently on the average over all possible contents of the table. If the table contents are known in advance, we might be able to choose a hashing function with guaranteed efficient (worst-case) performance. Such a technique has been called "perfect hashing" by Sprugnoli and others. In this paper, we address the question of whether perfect hashing is feasible in principle as a general technique, or whether it must rely on special qualities of the table contents. We approach the question by counting the number of functions which must be searched to be sure of finding a perfect hashing function. We present upper and lower bounds on the size of this search space, with attention to the tradeoff between the size of the search space and the size of the hash table.

Key words. hashing, perfect hashing

1. Introduction.

Often a computer program needs to accept as all or part of its input a sequence of character strings and decide, for each string, whether that string is a member of some finite set of known strings. The set of known strings may be nonempty when the program starts and may change as the program receives input. The strings, both known and otherwise, are generally referred to as *keys.* Testing a key for membership in the set of known keys is called a *search,* adding a key to the set of known keys is called an *insertion,* and removing a key from the set is a *deletion.*

Many different schemes have been developed to handle this computational task. These include linear search of an unordered table, binary search of an ordered table, *B*-trees, tries, various forms of string pattern matching, and hashing. The choice of one scheme over another for a certain application generally depends on the size of the set of known keys, and the relative numbers and order of searches, insertions, and deletions, since each scheme is efficient in some situations, and inefficient or inapplicable in others.

Hashing refers to schemes that use some simple arithmetic function of a key as the location in the table at which the key should be stored. The locations in the table are referred to as *buckets.* A search for a key is performed by computing the same function on the key to be searched for, and then comparing the key with whatever, if anything, is currently stored in the indicated bucket. A *collision* occurs when two keys to be inserted are mapped by the function to the same bucket.

In the general case in which search operations and insertion operations may be intermingled in an arbitrary manner, collisions are inevitable, and any hashing scheme designed to work in this case must include a method of resolving them. However, if the insertion operations are guaranteed to precede all the search operations, then all the keys which will ever be in the table are known in advance. In this case one might try to find a function mapping these keys, without collisions, into a table not much larger than the set of keys. In particular, we have the following definitions from Sprugnoli [Spr77].

* Received by the editors July 14, 1982, and in revised form March 13, 1985. This work was supported by the National Science Foundation under grants MCS 78-01812, MCS 80-05387, and MCS 81-01670.

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

‡ Department of Statistics, Purdue University, West Lafayette, Indiana 47907.

§ Bell Laboratories, Indian Hill, Illinois.

DEFINITION 1.1. A hashing function is a *perfect hashing function* (or *phf*) for a set of keys iff the function is 1-1 on that set of keys, i.e., there are no collisions on those keys.

DEFINITION 1.2. A hashing function is *a minimal phf* for a set of keys iff the function maps the keys 1-1 onto the buckets $0, 1, \cdots, k-1$, where k is the number of keys in the set.

Thus if one has a minimal *phf* for a set of keys, the hash table need have only as many entries as there are keys in the set. A *nearly minimal phf* is a function which uses a table not much larger than the set of keys, where "not much larger than" might be interpreted, for example, as "not larger than a constant factor times".

Every set of keys has, of course, several minimal *phf*s; the problem is that we may not know how to compute any of them in constant time, as we would like for a hashing function. Several articles [Spr77], [Cic80], [Jae81] recount procedures which take as input a set of keys and try to adjust numeric parameters in an arithmetic function in order to make it a *phf* for the given set of keys. Jaeschke's work [Jae81] is unique among these in that his method succeeds in producing a *phf* for every set of keys given as input. The problem with his method is that the parameters needed to specify the *phf* may grow very rapidly with the size of set of keys.

Alternate approaches to the general problem include that of Tarjan and Yao [Tar79] which produces minimal *phf*s computable in $O(\log_k P)$ time, where P is the size of the universal set from which k keys are drawn. Recently, Harry Mairson [private communication] has studied the tradeoff between program complexity and the number of probes of a table needed to determine whether a key is in the table. Perfect hashing, using but one probe, is at one extreme, while binary search, using $\log k$ probes is at the other. Carter and Wegman [Car77] study classes of functions with the property that given a set of keys, S, a function chosen at random from the class will be, on the average, a good hashing function for S. And Comer and O'Donnell [Com82] give an algorithm for producing *phf*s that are linear combinations of possibly nonperfect hashing functions; the *phf*s produced, though, may have very large ranges (table sizes).

This paper addresses the question of how large a collection of functions must be in order to be *perfect*, that is, to contain at least one *phf* for each set of keys. This is important because when this number is very large, then the large-parameter problem of Jaeschke's method is, in fact, inherent; when it is not, then it is likely that a refinement of Jaeschke's method, or some different scheme can overcome the parameter problem. We also investigate what types of functions make up minimal-sized perfect collections.

2. Notation. In order to study these questions, we must first make them precise. Since the internal structure of keys and bucket addresses is irrelevant to hashing, we shall take the set of possible keys and the set of buckets each to be an initial segment of the natural numbers. In addition, we will specify the size of the sets of keys for which *phf*s are to be found. We shall use the following symbols:

P the number of possible keys, those being $\{0, 1, \cdots, P-1\}$,

b the number of buckets, those being $\{0, 1, \cdots, b-1\}$,

k the number of keys in a key set,

$b^P = \{f \mid f\colon \{0, 1, \cdots, P-1\} \to \{0, 1, \cdots, b-1\}\}$.

Also, $\mathbf{R}$ will denote the real numbers, $\mathbf{R}^+$ the nonnegative real numbers, and $\mathbf{N}$ the natural numbers. And we shall need the following definitions.

DEFINITION 2.1. A (k, P)-*set* is a k-element subset of $\{0, 1, \cdots, P-1\}$.

DEFINITION 2.2. A function $f \in b^P$ *covers* a (k, P)-set, S, iff f is a *phf* for S. Also, $G \subseteq b^P$ *covers* a (k, P)-set, S, iff for at least one $g \in G$, g covers S.

DEFINITION 2.3. A collection of functions $F \subseteq b^P$ is *k-perfect* iff F covers every (k, P)-set.

By definition, then, the collection of functions produced by an algorithm generating a *phf* for each (k, P)-set must be k-perfect. Thus, the question which we wish to answer is

> Given P, b, and k, what is the smallest integer $Minf(k, b, P)$ such that there exists a k-perfect collection of $Minf(k, b, P)$ functions in b^P?

The answer will yield information about the feasibility of using parameterized *phf*s.

We have used two distinct approaches to finding $Minf(k, b, P)$. The first starts by posing the following question

> Given P, b, k and an integer n, what is the maximum number, $Maxs(n, k, b, P)$, of (k, P)-sets which can be covered by n functions in b^P?

Then to find *Minf* we observe that $Minf(k, b, P)$ is in fact the smallest n such that $Maxs(n, k, b, P)$ equals $\binom{P}{k}$, the total number of (k, P)-sets. Section 3 introduces an approach which reduces a collection of functions to an array of integers, called the key distribution, which still contains enough information to determine the number of (k, P)-sets covered. Sections 4 and 7 use key distributions to obtain some information about *Maxs* and *Minf*.

The second approach is to construct a k-perfect collection of functions, thus providing an upper bound on *Minf*. In §§ 5 and 6 we discuss some results obtained using this approach.

3. An abstraction of collections of functions.

DEFINITION 3.1. Given $F = \{f_1, \cdots, f_n\} \subseteq b^P$, $cover_k(F)$ is the number of (k, P)-sets covered by F.

In this section we define for each collection of functions, F, a structure called the *key distribution* of F, and show that it contains the information necessary to compute $cover_k(F)$. If $|F| = n$, then the key distribution of F, denoted $A(F)$, is an n-dimensional array of integers, and we develop a formula for a function, cov_k, of the key distribution such that $cov_k(A(F)) = cover_k(F)$.

If we have just one function $f \in b^P$, then we can count the number of (k, P)-sets covered by $\{f\}$ in the following way. First, let "simple bucket-set i" denote $f^{-1}(i)$, the set of keys mapped by f to i. Now select any k *distinct* buckets $i_1, \cdots, i_k$, and the corresponding simple bucket-sets. If we form a (k, P)-set by picking one key from each of the selected simple bucket-sets, that (k, P)-set is covered by f. For a particular selection of k buckets, $I = \{i_1, \cdots, i_k\}$, the number of (k, P)-sets that we can form in this way is just the product of the cardinalities of the corresponding simple bucket-sets:

$$\prod_{j=1}^{k} |\text{bucket-set } i_j| = \prod_{i \in I} |f^{-1}(i)|.$$

If we then sum over all ways of picking k buckets, we will have counted exactly once each (k, P)-set that f covers:

$$cover_k(\{f\}) = \sum_{\substack{i \subseteq \{0,1,\cdots,b-1\} \\ |I|=k}} \prod_{i \in I} |f^{-1}(i)|.$$

To illustrate the reasoning for collections containing more than one function, we will use a simple example involving two functions with four buckets acting on seven keys ($n = 2$, $b = 4$, $P = 7$). These functions are defined in Fig. 1. (For clarity, we have

x	q	r	s	t	u	v	w
$f(x)$	0	1	2	3	3	1	2
$g(x)$	1	0	2	3	1	0	3

FIG. 1. *Definition of f and g.*

i	$f^{-1}(i)$	$g^{-1}(i)$
0	$\{q\}$	$\{r, v\}$
1	$\{r, v\}$	$\{q, u\}$
2	$\{s, w\}$	$\{s\}$
3	$\{t, u\}$	$\{t, w\}$

FIG. 2. *Simple bucket-sets of f and g.*

named the keys by letters rather than integers.) Also, in this example we will take the number of keys per set to be three ($k=3$).

For collections $F=\{f_1, \cdots, f_n\} \subseteq b^P$ containing more than one function it will not do to just sum the preceding formula for $cover_k(\{f\})$ over all the functions in the collection, since a (k, P)-set could be covered by more than one function, but should only be counted once. (For instance, in our example both f and g cover $\{q, r, s\}$.) One way to overcome this difficulty is to generalize our notion of bucket-sets. Each generalized bucket-set is the intersection of n simple bucket-sets, one from each function. We name a generalized bucket-set by a vector, $\mu=\langle\mu(1), \cdots, \mu(n)\rangle$, which indicates from which simple bucket-sets it was derived. More precisely, the generalized bucket-set Z_μ is given by

$$Z_\mu(F)=Z_{\mu(1),\cdots,\mu(n)}(F)=\bigcap_{j=1}^{n} \text{bucket-set } \mu(j) \text{ from function } j$$

$$=\bigcap_{j=1}^{n} f_j^{-1}(\mu(j)).$$

Note that in general each key will be a member of exactly one generalized bucket set, so that the array of generalized bucket sets is partition of the keys.

In our example, the generalized bucket-sets of $\{f, g\}$ are named by ordered pairs (since $n=2$) and hence can be thought of as the elements of a square matrix, as illustrated in Fig. 3. For instance, $Z_{\langle 3,1\rangle}(\{f, g\})=f^{-1}(3) \cap g_{(1)}^{-1}=\{t, u\} \cap \{q, u\}=\{u\}$.

f \ g	0	1	2	3
0	$\varnothing$	$\{q\}$	$\varnothing$	$\varnothing$
1	$\{r, v\}$	$\varnothing$	$\varnothing$	$\varnothing$
2	$\varnothing$	$\varnothing$	$\{s\}$	$\{w\}$
3	$\varnothing$	$\{u\}$	$\varnothing$	$\{t\}$

FIG. 3. *Generalized bucket-sets of* $\{f, g\}$.

Unfortunately, it is not the case that picking one key from each of k distinct generalized bucket-sets will guarantee that the (k, P)-set so formed is covered by F. The essence of the complication is that for F to cover a (k, P)-set, some *single* function in F must cover that (k, P)-set. (For instance, s, w, and t come from distinct elements of $Z(\{f, g\})$, but neither f nor g covers $\{s, w, t\}$.) If we select k generalized bucket-sets, $Z_{\mu_1}, \cdots, Z_{\mu_k}$, then the (k, P)-sets formed by picking one key from each Z_{μ_i} will be covered by F just if there is some $f \in F$ such that $Z_{\mu_1}, \cdots, Z_{\mu_k}$ are subsets of k distinct simple bucket-sets of f. One can visualize this condition as follows. Suppose that S is a (k, P)-set formed by picking one key from each of $Z_{\mu_1}, \cdots, Z_{\mu_k}$. Align the names of the generalized buckets sets $\mu_1, \cdots, \mu_k$ one beneath the other:

$$\begin{array}{c} \langle \mu_1(1) \cdots \mu_1(n) \rangle \\ \langle \mu_2(1) \cdots \mu_2(n) \rangle \\ \vdots \qquad\quad \vdots \\ \langle \mu_k(1) \cdots \mu_k(n) \rangle \end{array}$$

Then $f_i \in F$ covers S if and only if column i of the display above comprises k distinct entries. Thus, F covers S if and only if at least one column of the display comprises k distinct entries. To facilitate using this notion in formulae, we introduce

$$\mathit{diff}_k(\mu_1, \cdots, \mu_k) = |\{j\colon \mu_1(j), \cdots, \mu_k(j) \text{ are pairwise distinct}\}|.$$

Thus $\mathit{diff}_k(\mu_1, \cdots, \mu_k)$ is exactly the number of functions in F which cover each (k, P)-set formed by picking one key from each of $Z_{\mu_1}, \cdots, Z_{\mu_k}$.

In our example, $\mathit{diff}_3(\langle 2, 2\rangle, \langle 2, 3\rangle, \langle 3, 3\rangle) = 0$, $\mathit{diff}_3(\langle 2, 2\rangle, \langle 2, 3\rangle, \langle 3, 1\rangle) = 1$, and $\mathit{diff}_3(\langle 1, 0\rangle, \langle 2, 2\rangle, \langle 3, 3\rangle) = 2$. Thus, respectively, $\{s, w, t\}$ is covered by 0 functions, $\{s, t, u\}$ is covered by 1 function (g), and $\{r, s, t\}$ is covered by 2 functions.

So to compute $\mathit{cover}_k(F)$ for $|F| > 1$, we take, as in the case of one function, products of cardinalities in sequences of k bucket-sets. But we sum only over the sequences of bucket-sets having the property described above:

$$\mathit{cover}_k(F) = \frac{1}{k!} \sum_{\mathit{diff}_k(\mu_1, \cdots, \mu_k) \geq 1} \prod_{j=1}^{k} |Z_{\mu_j}(F)|.$$

The factor $1/k!$ reflects the fact that each collection of k bucket-sets, $Z_{\xi_1}, \cdots, Z_{\xi_k}$, can be ordered in $k!$ different ways to appear as $Z_{\mu_1}, \cdots, Z_{\mu_k}$ in the formula.

Since apparently only the cardinalities of the various Z_μ are important we define

$$a_\mu(F) = |Z_\mu(F)| = \left| \bigcap_{j=1}^{n} f_j^{-1}(\mu(j)) \right|.$$

Then $A(F) = [a_\mu(F)]$ will be an n-dimensional $(b \times b \times \cdots \times b)$ array of nonnegative integers with $\sum_\mu a_\mu = P$ (since each key is counted in exactly one a_μ). We refer to A as the *key distribution* of F. And, as promised, we can define a function of the key distribution

$$\mathit{cov}_k(A) = \frac{1}{k!} \sum_{\mathit{diff}_k(\mu_1, \cdots, \mu_k) \geq 1} \prod_{j=1}^{k} a_{\mu_j}.$$

so that $\mathit{cov}_k(A(F)) = \mathit{cover}_k(F)$.

Figure 4 shows the key distribution $A(\{f, g\})$. We find the number of sets of size three covered by $\{f, g\}$ by computing $\mathit{cov}_3(A(\{f, g\})) = 10 \cdot (2 \cdot 1 \cdot 1) + 8 \cdot (1 \cdot 1 \cdot 1) = 28$.

f \ g	0	1	2	3
0	0	1	0	0
1	2	0	0	0
2	0	0	1	1
3	0	1	0	1

FIG. 4. *Key distribution*: $A(\{f, g\})$.

Our goal is to translate our questions about collections of functions into questions about key distributions. We must first discover what the set of all key distributions is.

DEFINITION 3.2a. For $n \geqq 1$, $b \geqq 1$, and $P > 0$, let $\Omega^N_{n,b,P}$ denote the space of all n-dimensional, $b \times b \times \cdots \times b$ arrays, A, with entries which are nonnegative integers such that $\sum_\mu a_\mu = P$.

Our observations to this point show that for every collection of n functions, $F \subseteq b^P$, the key distribution $A(F) \in \Omega^N_{n,b,P}$. We now observe that in fact every $A \in \Omega^N_{n,b,P}$ is the key distribution of some collection of n functions in b^P. Thus we can find $Maxs(n, k, b, P)$ if we can find the maxima of $cov_k(A)$ for $A \in \Omega^N_{n,b,P}$. In § 4 we show that in the case of one function ($n = 1$) and in the case of two keys per set ($k = 2$), cov is of a special form which facilitates finding its maxima.

In § 7 we try to obtain at least an approximate answer for the general case (arbitrary n, k, b, and P) by extending the domain of $cov_k(A)$ to arrays of real numbers.

DEFINITION 3.2b. For $n \geqq 1$, $b \geqq 1$, and $P > 0$, let $\Omega_{n,b,P}$ denote the space of all n-dimensional, $b \times b \times \cdots \times b$ arrays, A, with entries which are real numbers such that $\sum_\mu a_\mu = P$.

DEFINITION 3.2c. For $n \geqq 1$, $b \geqq 1$, and $P > 0$, let $\Omega^+_{n,b,P}$ denote the space of all n-dimensional, $b \times b \times \cdots \times b$ arrays, A, with entries which are nonnegative real numbers such that $\sum_\mu a_\mu = P$.

Section 7 concerns finding a maximum of $cov_k(A)$ for $A \in \Omega^+_{n,b,P}$. Such a maximum would certainly be an upper bound on the maximum of $cov_k(A)$, $A \in \Omega^N_{n,b,P}$ and thus would also bound $Maxs(n, k, b, P)$ above, yielding a lower bound for $Minf$.

4. Results for two special cases: one function ($n = 1$); and two keys per set ($k = 2$). Formally, the function $cov_k(A)$ is the sum of products, each product having k distinct elements of A as factors. In general, cov_k does not include *all* such products. However, in two special cases it does. In this section we first discuss some properties of the function, denoted by $\Gamma_k(A)$, that is the sum of *all* products of k distinct elements of a key distribution A, and then exhibit the results about those special cases that we can deduce from the properties of Γ_k.

DEFINITION 4.1. For $n \geqq 1$, $k \geqq 2$, $b \geqq k$, $P > 0$, for $A \in \Omega_{n,b,P}$

$$\Gamma_k(A) = \sum_{\substack{M \subseteq b^n \\ |M| = k}} \prod_{\mu \in M} a_\mu.$$

In finding the maxima of cov in the case involving one function ($n = 1$) both Berman, Bock, and Plank [Ber81] and Anderson and Sprugnoli [And79] used the idea of comparing $cov(A)$ with $cov(A')$, where A' is obtained from A by perturbing just two elements of A. By studying how the value of Γ_k changes under such transformations,

we can show that Γ_k is maximized on arrays in which all entries have the same value [Ber82, Dit82].

DEFINITION 4.2. For each real constant c let $[c]_{n,b}$ denote the n-dimensional $b \times b \times \cdots \times b$ array, all the entries of which have the value c.

THEOREM 4.3. *For $A \in \Omega^{+}_{n,b,P}$ the value of $\Gamma_k(A)$ is maximized at $A = [P/b^n]_{n,b}$.*

A similar result holds when we restrict our attention to key distributions.

DEFINITION 4.4. $A \sim_N [c]_{n,b}$ (read A approximates $[c]_{n,b}$ with integers) if and only if the sum of the elements of A is $c \cdot b^n$ and each element of A is equal to either $\lfloor c \rfloor$ or $\lceil c \rceil$.

THEOREM 4.5. *For $A \in \Omega^{N}_{n,b,P}$ the value of $\Gamma_k(A)$ is maximized if and only if $A \sim_N [P/b^n]_{n,b}$ (provided that $P \geqq k$).*

The first use we make of these findings is to generalize the results of Anderson and Sprugnoli [And79] and Berman, Bock, and Plank [Ber81] for the case of one function ($n = 1$). In this case, the key distribution is a vector, $A = [a_1, \cdots, a_b]$, where a_i is the number of keys mapped by f to bucket i, and

$$cov_k(A) = \sum_{\substack{I \subseteq b \\ |I| = k}} \prod_{i \in I} a_i.$$

Thus, for $A \in \Omega_{1,b,P}$, $cov_k(A)$ is the sum of *all* products of k distinct elements of A, so in this case $cov_k(A)$ is identically $\Gamma_k(A)$. It follows immediately from Theorem 4.5 that for one function ($A \in \Omega^{N}_{1,b,P}$) $cov_k(A)$ is maximized by key distributions which have, as nearly as possible, all elements equal ($A \sim_N [P/b]_{1,b}$). This allows an exact calculation of $Maxs(1, k, b, P)$.

THEOREM 4.6. *$cov_K(A)$, $A \in \Omega^{N}_{1,b,P}$ is maximized if and only if $A \sim_N [P/b]_{1,b}$. Thus*

$$Maxs(1, k, b, P) = \sum_{r=0}^{k} \binom{P - P \bmod b}{r} \cdot \lfloor P/b \rfloor^r \cdot \binom{P \bmod b}{k-r} \cdot \lceil P/b \rceil^{k-r} \approx \binom{b}{k} (P/b)^k.$$

Our second application of the properties of Γ_k is to the case of two keys per set ($k = 2$). We establish a complete characterization, in terms of key distributions, of the collections of functions from b^P that cover all $(2, P)$-sets, and show that the minimal size for such a collection is exactly $\lceil \log_b P \rceil$. Examining the form of cov_2, we find that

$$cov_{n,b,2}(A) = \frac{1}{2} \sum_{diff_2(\mu,\xi) \geqq 1} a_\mu a_\xi = \frac{1}{2} \sum_{\mu} \sum_{\xi \neq \mu} a_\mu a_\xi,$$

because $diff_2(\mu, \xi) \geqq 1$ if and only if $\mu \neq \xi$. So cov_2 is just the sum of all products of pairs of elements of A, which is Γ_2, and the following is immediate from Theorem 4.5.

THEOREM 4.7. *$cov_2(A)$, $A \in \Omega^{N}_{n,b,P}$ is maximized if and only if $A \sim_N [P/b^n]_{n,b}$. Thus*

$$Maxs(n, 2, b, P) = \frac{1}{2}\left[P^2 - P \bmod b^n - \frac{P^2 - (P \bmod b^n)^2}{b^n} \right].$$

Our next two results are then easy corollaries, or they can be proved independently [Ber82].

PROPOSITION 4.8. *$F = \{f_1, \cdots, f_n\} \subseteq b^P$ covers all $(2, P)$-sets iff its keys distribution, $A(F)$ comprises solely 0s and 1s.*

THEOREM 4.9. *$Minf(2, b, P) = \lceil \log_b P \rceil$.*

The results in these special cases seem to indicate that there is one particular kind of function that is especially interesting if we are trying to construct perfect collections.

DEFINITION 4.10. A function $f \in b^P$ is an *even-sprinkling function* iff its key distribution, A, is such that $A \sim_N [P/b]_{1,b}$. Thus an even-sprinkling function is one which maps, as nearly as possible, the same number of keys to each bucket.

Theorem 4.6 tells us that each even-sprinkling function covers the maximal number of (k, P)-sets, and that every other function covers fewer. This is consistent with our intuition about which functions are good hashing functions: for example, *mod* is an even-sprinkling function.

In the case of two keys per set, there is always a minimal size perfect collection comprising only even-sprinkling functions [Ber81]. A good example of a minimal-sized collection, $F \subseteq b^P$, with a (0, 1) key distribution is $F = \{f_i: 0 \leqq i \leqq \lceil \log_b P \rceil - 1\}$, where

$$f_i(x) = i\text{th digit in the base } b \text{ representation of } x.$$

This also illustrates the point that given a collection G, if one considers, for each key x, the vector $\langle g_1(x), \cdots, g_n(x)\rangle$ to be a representation of x, then a (0, 1) key distribution for G corresponds to each key having a unique representation.

In fact, for each n there is a set of n even-sprinkling functions covering the maximal number of $(2, P)$-sets ($Maxs(n, 2, b, P)$). Also, in some cases (when P/b is near an integer), every collection of n functions covering $Maxs(n, 2, b, P)$ $(2, P)$-sets must comprise only even-sprinkling functions. It remains an open question whether for arbitrary k, for every n, there is a collection of n even-sprinkling functions covering $Maxs(n, k, b, P)$ (k, P)-sets.

5. A collection of functions which covers all sets of size 3. In this section we prove an upper bound on the minimal size of a collection of functions which covers all $(3, P)$-sets ($Minf(3, b, P)$), by constructing such a collection for each b and P.

THEOREM 5.1. *For* $b \neq 6$, $Minf(3, b, P) \leqq (\lceil \log_b P \rceil)^2$; *and* $Minf(3, 6, P) \leqq (\lceil 1.2 \cdot \log_n P \rceil]^2$.

Proof. By Proposition 4.9 we can find $\lceil \log_b P \rceil$ functions in b^P such that for any two keys (elements of $\{0, 1, \cdots, P-1\}$), at least one of the functions will map the two keys to different buckets. For example, we could use the functions f_i defined by

$$f_i(x) = i\text{th digit in the base } b \text{ representation of } x$$

for $i = 0, \cdots, \lceil \log_b P \rceil - 1$. Let F denote this set of functions.

Note that this set of functions does not cover all $(3, P)$-sets, since there are a number of sets of three distinct elements which do not all differ in the *same* digit (for example 00, 01 and 11). However, for each $(3, P)$-set, $S = \{s_1, s_2, s_3\}$, either (a) some $f_i \in F$ covers S; or (b) there are two functions f_l and f_m such that f_l distinguishes s_1 from s_2 and s_3, and f_m distinguishes s_2 from s_1 and s_3 (subject, perhaps, to a renumbering of s_1, s_2, s_3). In particular, in case (b) f_l and f_m could be diagrammed as follows:

s	$f_l(s)$	$f_m(s)$
s_1	x	z
s_2	y	w
s_3	y	z

where $x \neq y$ and $w \neq z$.

We can obtain a collection of functions which covers all the $(3, P)$-sets by adding to F a collection of functions which covers the $(3, P)$-sets in case (b). We do this by constructing, from each pair of functions $f_l, f_m \in F$, two new functions g and h such that, if f_l and f_m satisfy the conditions of case (b) for S, then either g or h covers S. The construction uses the concept of orthogonal latin squares.

DEFINITION 5.2. A *latin square* of order r is an $r \times r$ square matrix having entries chosen from a set of r elements in such a way that each element appears exactly once in each column and exactly once in each row. In a latin square L, $L(i,j)$ denotes the entry in row i and column j $(0 \leqq i, j \leqq r-1)$.

DEFINITION 5.3. Two latin squares L and M of order r are *orthogonal* if every ordered pair of symbols occurs exactly once among the r^2 pairs $\{\langle L(i,j), M(i,j)\rangle: 0 \leqq i, j \leqq r-1\}$. If $r \neq 6$, then such a pair of latin squares always exists [Bos60].

Given $b \neq 6$ and a collection of functions F as above, let L and M be orthogonal latin squares of order b with entries from $\{0, 1, \cdots, b-1\}$. For each pair of functions $f_l, f_m \in F$ $(0 \leqq l < m \leqq \lceil \log_b P \rceil - 1)$ let

$$g_{l,m}(x) = L(f_l(x), f_m(x)),$$

and

$$h_{l,m}(x) = M(f_l(x), f_m(x)).$$

And let $E = F \cup \{g_{l,m}, h_{l,m}: 0 \leqq l < m \leqq \lceil \log_b P \rceil - 1\}$.

We claim that E covers all $(3, P)$-sets. To see this, consider an arbitrary $(3, P)$-set, $S = \{s_1, s_2, s_3\}$. If one of the $f_i \in F$ covers S, then we are done. If not, then case (b) above pertains and for some l and m $f_l, f_m, g_{l,m}, h_{l,m}$ act on $\{s_1, s_2, s_3\}$ as follows:

s	$f_l(s)$	$f_m(s)$	$g_{l,m}(s)$	$h_{l,m}(s)$
s_1	x	z	$L(x, z)$	$M(x, z)$
s_2	y	w	$L(y, w)$	$M(y, w)$
s_3	y	z	$L(y, z)$	$M(y, z)$

where $x \neq y$ and $w \neq z$. Since the same value does not appear twice in any row or column of a latin square $g_{l,m}(s_1) \neq g_{l,m}(s_3)$, $g_{l,m}(s_2) \neq g_{l,m}(s_3)$, $h_{l,m}(s_1) \neq h_{l,m}(s_3)$, and $h_{l,m}(s_2) \neq h_{l,m}(s_3)$. Furthermore, the orthogonality of L and M guarantees that either $g_{l,m}(s_1) \neq g_{l,m}(s_2)$ or $h_{l,m}(s_1) \neq h_{l,m}(s_2)$, for if not then $\langle L(x, z), M(x, z)\rangle = \langle L(y, w), M(y, w)\rangle$, which contradicts the orthogonality of L and M. Thus either $g_{l,m}$ or $h_{l,m}$ covers S. Since S was arbitrary, this shows that E covers all $(3, P)$-sets. It remains to count the number of functions in E.

We know that $|F| = \lceil \log_b P \rceil$. The added collections $\{g_{l,m}\}$ and $\{h_{l,m}\}$ each contain at most one function for each unordered pair (l, m) of indices from $\{0, \cdots, \lceil \log_b P \rceil - 1\}$. Thus

$$|E| \leqq \lceil \log_b P \rceil + 2 \cdot \tfrac{1}{2} \cdot (\lceil \log_b P \rceil)(\lceil \log_b P - 1 \rceil) = (\lceil \log_b P \rceil)^2.$$

Recall that we cannot guarantee the existence of orthogonal latin squares when $b = 6$. However, in this case we obtain an only slightly larger bound by observing that $Minf(3, 6, P) \leqq Minf(3, 5, P)$, and we have just shown that $Minf(3, 5, P) \leqq (\lceil \log_5 P \rceil)^2$. Thus

$$Minf(3, 6, P) \leqq (\lceil \log_5 P \rceil)^2 = (\lceil \log_5 6 \cdot \log_6 P \rceil)^2 < (\lceil 1.2 \cdot \log_6 P \rceil)^2.$$

6. Some k-perfect collections of minimal *phf*s. In this section we exhibit a construction producing for each key-set k and each universal set size P a k-perfect collection $F_{k,P}$ of functions using k buckets. The size of $F_{k,P}$ is then an upper bound on

Minf(k, k, P). The construction is derived from one given in [Yao81]. For clarity of presentation, we assume that P is a power of 2. If P is not a power of 2, the results are substantially the same, but the notation and proofs are more complicated ([Dit82]).

The central idea of the construction is to split the domain, P, in two, and then form a k-perfect collection from combinations of pairs of functions: one from an i-perfect collection for one half of the domain, the other from a $(k-i)$-perfect collection for the other half. The basis for this recursion uses the identity function if the domain becomes small enough ($P < k$), or a function mapping everything to 0 if $k = 1$. To give the details, we need a convenient method for denoting renamings of the elements in the domain and/or the range of a function.

DEFINITION 6.1. Suppose f: $D \to R$, $D = \{d_1, \cdots, d_\delta\}$, $R = \{r_1, \cdots, r_\rho\}$, and we have two other sets: $D' = \{d'_1, \cdots, d'_\delta\}$, and $R' = \{r'_1, \cdots, r'_\rho\}$. Then $f[D'; R']$ is defined by

$$f[D'; R'](d'_i) = r'_j \Leftrightarrow f(d_i) = r_j.$$

Similarly, for a collection of functions, F, with common domain D and range R, $F[D'; R'] = \{f[D'; R'] \mid f \in F\}$.

Note that if F is k-perfect, then so is $F[D'; R']$.

Construction 6.2. Our construction of $F_{k,P}$ goes by recursion on k. The case ($k \geqq 2$) goes by recursion on P.

$$(k = 1)\text{: } F_{1,P} = \{f\}, \quad \text{where } f(x) = 0, \quad x = 0, \cdots, P-1,$$

$$(k \geqq 2, 2 \leqq P \leqq k)\text{: } F_{k,P} = \{f\}, \quad \text{where } f(x) = x, \quad x = 0, \cdots, P-1,$$

$$(k \geqq 2, P > k)\text{: Let } P_1 = \{0, \cdots, P/2-1\} \quad \text{and } P_2 = \{P/2, \cdots, P-1\}.$$

$$\text{Let } G_{k,P} = \bigcup_{1 \leqq i \leqq k-1} \{f \cup g[P_2; \{i, \cdots, k-1\}] \mid f \in F_{i,P/2}, g \in F_{k-i,P/2}\}$$

$$\text{and } H_{k,P} = \{f \cup f[P_2; k] \mid f \in F_{k,P/2}\}.$$

$$\text{Then } F_{k,P} = G_{k,P} \cup H_{k,P}.$$

Motivation for the last step of the construction can be found in the proof that $F_{k,P}$ is k-perfect.

PROPOSITION 6.3. *$F_{k,P}$ is k-perfect.*

Proof. For the cases ($k = 1$) and ($k \geqq 2$, $P \leqq k$), this is obvious from the construction. These form the bases for inductions on k and P, respectively. To complete the proof, we need to show that $F_{k,P}$ is k-perfect when $k \geqq 2$ and $P > k$. Our induction hypotheses are that $F_{m,Q}$ is m-perfect for $m < k$ and that $F_{k,Q}$ is k-perfect for $Q < P$.

Now pick an arbitrary (k, P)-set, $S = \{s_1, \cdots, s_k\}$, and imagine the domain, $\{0, 1, \cdots, P-1\}$, being split into two halves: $P_1 = \{0, \cdots, P/2-1\}$ and $P_2 = \{P/2, \cdots, P-1\}$. This induces a splitting of S as well. The first possibility is that the splitting of S is trivial; that is, either $S \subseteq P_1$ or $S \subseteq P_2$. In this case, one of the functions in $H_{k,P}$ covers S. In particular, by the induction hypothesis $F_{k,P/2}$ is k-perfect; so if $S \subseteq P_1$, then there is an $f \in F_{k,P/2}$ that covers S, and if $S \subseteq P_2$, then there is an $f \in F_{k,P/2}$ such that $f[P_2; k]$ covers S. Furthermore, the function $f \cup f[P_2; k]$ covers every (k, P)-set covered by either f or $f[P_2; k]$. (The point of gluing the functions together being just to reduce the size of $H_{k,P}$.) Thus, since $F_{k,P} \supseteq H_{k,P}$, $F_{k,P}$ covers S in this case.

If the splitting of S is nontrivial, then one of the functions in $G_{k,P}$ covers S. Let $S_1 = S \cap P_1$, $S_2 = S \cap P_2$, $i = |S_1|$. Note that $1 \leqq i \leqq k-1$. By our induction hypothesis, $F_{i,P/2}$ is i-perfect, and $F_{k-i,P/2}$ is $(k-i)$-perfect. Hence, there is an $f \in F_{i,P/2}$ that covers S_1, and there is a $g \in F_{k-i,P/2}[P_2; \{i, \cdots, k-1\}]$ that is 1-1 on S_2. Given these, it is

clear that $f \cup g$ covers S. Since we have carefully included all functions of this sort in $G_{k,P}$, and so in $F_{k,P}$, it follows that $F_{k,P}$ covers S.

Next we derive an upper bound on the size of $F_{k,P}$, which is then also an upper bound on $Minf(k, k, P)$. The steps in the inductive proof follow the steps in the recursive construction of $F_{k,P}$.

THEOREM 6.4. $Minf(k, k, P) \leqq |F_{k,P}| \leqq k^{k-1}(\log_2 P)^{k-1}$.

Proof. Here we only prove this for P a power of 2, that is, $Minf(k, k, 2^\alpha) \leqq (k\alpha)^{k-1}$. The same result holds in general [Dit82], the proof being similar, but more complicated.

Since $F_{k,P}$ are k-perfect collections, it suffices to show that $|F_{k,2^\alpha}| \leqq (k\alpha)^{k-1}$. We prove this by induction on k and α. For the cases $(k = 1)$ and $(k \geqq 2, 2 \leqq 2^\alpha \leqq k)$

$$|F_{k,2^\alpha}| = 1 \leqq (k\alpha)^{k-1}.$$

For $(k \geqq 2, 2^\alpha > k)$, from our construction of $F_{k,P}$ we see that

$$|F_{k,2^\alpha}| \leqq |G_{k,2^\alpha}| + |H_{k,2^\alpha}| \leqq \sum_{i=1}^{k-1} |F_{i,2^{\alpha-1}}| \cdot |F_{k-i,2^{\alpha-1}}| + |F_{k,2^{\alpha-1}}|.$$

Then by our induction hypotheses

$$\begin{aligned}
|F_{k,2^\alpha}| &\leqq \sum_{i=1}^{k-1} (i(\alpha-1))^{i-1} \cdot ((k-i)(\alpha-1))^{k-i-1} + (k(\alpha-1))^{k-1} \\
&= (\alpha-1)^{k-2} \sum_{i=1}^{k-1} i^{i-1}(k-i)^{k-i-1} + (k(\alpha-1))^{k-1} \\
&\leqq (\alpha-1)^{k-2} \sum_{i=1}^{k-1} (k-1)^{k-2} + (k(\alpha-1))^{k-1} \\
&= (k-1)^{k-1}(\alpha-1)^{k-2} + k^{k-1}(\alpha-1)^{k-1} \\
&\leqq k^{k-1}(\alpha-1)^{k-2}[1+\alpha-1] \\
&\leqq k^{k-1}\alpha^{k-1}.
\end{aligned}$$

By comparison, the construction in [Yao81] splits the domain into k parts, and the derived bound on the size is $4^{k^2}(\log P)^{k-1}$.

7. Results obtained using arrays of real numbers. Despite our success in the case of one function and the case of two keys per set, we do not known in general how to find the maxima of cov_k over integer arrays ($\Omega^N_{n,b,P}$, the key distributions). However, if we could find the maxima of cov_k over arrays with nonnegative real numbers as entries, then that maximal value would at least be an upper bound on the maximal value of cov_k over integer arrays. In this section we present the results we have obtained using this approach.

Recalling our observations in previous sections that for $A \in \Omega_{1,b,P}$, $cov_k(A)$ is identical to $\Gamma_k(A)$, and that in all cases cov_2 is identical to Γ_2, the following are consequences of Theorem 4.3.

PROPOSITION 7.1. $cov_k(A)$, $A \in \Omega^+_{1,b,P}$ *is maximized at* $A = [P/b]_{1,b}$.

PROPOSITION 7.2. $cov_2(A)$, $A \in \Omega^+_{n,b,P}$ *is maximized at* $A = [P/b^n]_{n,b}$.

Considering this evidence, we make the following conjecture.

Conjecture 7.3. $cov_k(A)$, $A \in \Omega^+_{n,b,P}$ *is maximized at* $A = [P/b^n]_{n,b}$.

The following theorem represents the extent of our progress in proving this conjecture correct.

THEOREM 7.4. $\forall n \geq 1, \forall k \geq 2, \forall b \geq k, \forall P > 0$, $cov_k(A)$, $A \in \Omega^+_{n,b,P}$ *has a strict local maximum at* $A = [P/b^n]_{n,b}$.

Proof. We present here an outline of the proof; details are available in [Dit82]. We prove this theorem by showing that on lines in $\Omega^+_{n,b,P}$ which pass through $[P/b^n]_{n,b}$, cov_k has a strict *local* maximum at the point $A = [P/b^n]_{n,b}$. (Note that a proof that $[P/b^n]_{n,b}$ is a *global* maximum on the portion of each line comprising arrays with nonnegative entries would be a proof of our conjecture.)

Each line in $\Omega^+_{n,b,P}$ can be characterized as a set of arrays, $\{(\rho \cdot D + [P/b^n]_{n,b}) \mid \rho \in \mathbf{R}\}$, where D is an n-dimensional $b \times, \cdots, \times b$ array, the elements of which sum to 0 ($D \in \Omega_{n,b,0}$), but which has nonzero elements ($D \neq [0]_{n,b}$). Thinking of D as fixed, we see that $cov_k(\rho D + [P/b^n]_{n,b})$ is a function of the single variable ρ. Our goal, then, is to show that $cov_k(\rho D + [P/b^n]_{n,b})$ has a strict local maximum at $\rho = 0$.

The first step is to show that $cov_k(\rho D + [P/b^n]_{n,b})$ is a polynomial in ρ of degree k, i.e., there are coefficients $c_0, \cdots, c_k$ such that

$$cov_k(\rho D + [P/b^n]_{n,b}) = \sum_{r=0}^{k} c_r \cdot \rho^r.$$

Of course, the coefficients depend on the choice of D, so we write

$$cov_k(\rho D + [P/b^n]_{n,b}) = \sum_{r=0}^{k} c_r(D) \cdot \rho^r. \tag{1}$$

Next we establish some facts about the coefficients $c_0(D)$, $c_1(D)$, and $c_2(D)$. In particular, $\forall D \in \Omega_{n,b,0} \backslash \{[0]_{n,b}\}$ (i.e., for every line)

$$c_0(D) = cov_k([P/b^n]_{n,b}).$$

This is easily shown by substituting $\rho = 0$ in (1). Secondly, $\forall D \in \Omega_{n,b,0} \backslash \{[0]_{n,b}\}$

$$c_1(D) = 0.$$

To prove this requires a little algebra, but basically it follows from the fact that the elements of D sum to 0. Finally, we can show that $\forall D \in \Omega_{n,b,0} \backslash \{[0]_{n,b}\}$

$$c_2(D) < 0.$$

The proof of this is long and seems to have no easy intuition behind it.

We can use these facts about the coefficients to rewrite (1):

$$cov_k(\rho D + [P/b^n]_{n,b}) = cov_k([P/b^n]_{n,b}) + \rho^2 c_2(D) + \sum_{r=3}^{k} c_r(D) \rho^r.$$

Now for sufficiently small ρ, $\rho^2 c_2(D)$ will dominate $\sum_{r=3}^{k} c_r(D) \rho^r$. Therefore, since $c_2(D)$ is always negative, we have that for sufficiently small ρ ($\rho \neq 0$)

$$cov_k(\rho D + [P/b^n]_{n,b}) < cov_k([P/b^n]_{n,b}).$$

Hence, $cov_k([P/b^n]_{n,b})$ is a local maximum on every line through $[P/b^n]_{n,b}$.

This result does not give us any certain information about $Maxs(n, k, b, P)$, the number of (k, P)-sets that can be covered by n functions from b^P. However, if $A = [P/b^n]_{n,b}$ is *global* maximum for $cov_k(A)$, $A \in \Omega^+_{n,b,P}$, then $cov_k([P/b^n]_{n,b})$ is an upper bound on $Maxs(n, k, b, P)$. In particular

$$cov_k([P/b^n]_{n,b}) = \frac{1}{k!} \sum_{diff_k(\mu_1, \cdots, \mu_k) \geq 1} \prod_{j=1}^{k} P/b^n = \frac{1}{k!} [P/b^n]^k \sum_{diff_k(\mu_1, \cdots, \mu_k) \geq 1} 1.$$

We can show [Dit82] that

$$\sum_{diff_k(\mu_1,\cdots,\mu_k)\geq 1} 1 = b^{nk} \cdot \left[1-\left(1-\frac{b!}{(b-k)!\,b^k}\right)^n\right].$$

Hence

$$\begin{aligned} cov_k([P/b^n]_{n,b}) &= \frac{1}{k!}[P/b^n]^k b^{nk} \cdot \left[1-\left(1-\frac{b!}{(b-k)!\,b^k}\right)^n\right] \\ &= \frac{P^k}{k!}\left[1-\left(1-\frac{b!}{(b-k)!\,b^k}\right)^n\right]. \end{aligned} \tag{2}$$

If our conjecture is correct, then (2) is an upper bound on $Maxs(n, k, b, P)$, and solving

$$cov_k([P/b^n]_{n,b}) = \text{total } \# \text{ of } (k, P)\text{-sets} = \binom{P}{k}$$

for n would produce a lower bound on $Minf(k, b, P)$. The outcome of this calculation is

$$n = \frac{\log\left[1-P!/(P-k)!\,P^k\right]}{\log\left[1-b!/(b-k)!\,b^k\right]}.$$

However, this may not be a tight bound since the key distribution of a collection of functions which covers all (k, P)-sets contains many zeros, and hence is on the boundary of $\Omega^+_{n,b,P}$, whereas $[P/b^n]_{n,b}$ is right in the middle of $\Omega^+_{n,b,P}$.

8. Conclusion. The preceding sections of this paper contain a variety of results which we have obtained pursuing various approaches to finding the minimal size for k-perfect collections of functions. This minimal size is significant for any perfect hashing scheme employing parametrized functions, because the parameters must be large enough to uniquely identify each function of some k-perfect collection. And the time complexity of actually evaluating the *phf* must include time to do computations involving the parameter (although the size of the parmater is not necessarily a lower bound on the time complexity evaluating the *phf*, since the entire parameter may not be used during every evaluation). In this section we discuss the bounds we can derive on the minimal size of k-perfect collections, using the results from previous sections, and we summarize the consequences of these results for perfect hashing schemes that employ parameterized functions.

In § 4 we found an exact expression for $Maxs(1, k, b, P)$, the maximum number of sets that one hashing function can cover. However, this expression is rather cumbersome, and Proposition 9.1 provides a reasonably tight and relatively simple upper bound:

$$Maxs(1, k, b, P) \leqq cov_k([P/b]_{1,b}) = \binom{b}{k}(P/b)^k.$$

We can use this to calculate a lower bound on $Minf(k, b, P)$, the minimum size of a k-perfect collection:

$$Minf(k, b, P) \geqq \frac{\text{total } \# \text{ of } (k, P)\text{-sets}}{Maxs(1, k, b, P)} \geqq \frac{\binom{P}{k}}{\binom{b}{k}(P/b)^k}.$$

For *minimal* perfect hashing, when we have the same number of buckets as keys per set ($b = k$), this bound becomes

$$Minf(k, k, P) \geqq \binom{P}{k}\binom{k}{P}^k.$$

Applying Stirling's approximation for factorials and doing some algebra yields

$$Minf(k, k, P) \geqq .89\frac{(f(k, P))^k}{\sqrt{k}}, \quad \text{where } f(k, P) = \left(1+\frac{k}{P-k}\right)(P-k)/k.$$

The quantity $f(k, P)$ increases toward the constant e as the ratio $(P-k)/k$ becomes large. Table 1 indicates the rate at which this convergence occurs. Since in applications of perfect hashing $(P-k)/k$ is likely to be very large indeed, for practical purposes we can consider our lower bound to be $.89e^k/\sqrt{k}$. Thus, the size in bits of the parameters specifying minimal *phf*s will be at least $\log_2(.89e^k/\sqrt{k}) \geqq 1.44k - \frac{1}{2}\log_2 k - .16$.

TABLE 1

$(P-k)/k$	$f(k, P)$
1	2.000
2	2.250
3	2.370
4	2.441
5	2.488
10	2.594
20	2.653
100	2.705
1,000	2.717
10,000	2.718

To see how much allowing more buckets than keys might reduce the parameter size, we apply similar reasoning to our lower bound for $Minf(k, b, P)$ for arbitrary b. We find that

$$Minf(k, b, P) \geqq \frac{.8}{\sqrt{k+1}}\left(\frac{f(k, P)}{f(k, b)}\right)^k,$$

where f is as above. The minimum parameter size is then $k \cdot \log_2(f(k, P)/f(k, b)) - \frac{1}{2}\log_2(k+1) - .32$. Hence, increasing the number of buckets may substantially reduce the coefficient of k. For example, if we assume $P \gg k$, so that $f(k, P) \approx e$, and make $b = 2k$, then the coefficient of k is $\log_2(e/f(k, 2k)) \approx .44$. However, so long as b depends linearly on k, the ratio $(b-k)/k$ is constant, and so the coefficient of k is constant. Hence, the number of bits in each parameter will be at least linear in k.

On the other hand, the results in § 6 indicate that the situation may not be much worse than this. The upper bound $Minf(k, k, P) \leqq (k \log_2 P)^{k-1}$ implies that the parameter size theoretically need be no greater than $(k-1)(\log_2 k + \log_2 \log_2 P)$, and there are at least two reasons to suspect that this bound is overly large. First, it was derived for minimal perfect hashing ($b = k$), and so takes no account of the advantage gained by allowing more buckets than there are keys per set. Second, many of the functions in the k-perfect collection produced by the construction in § 6 are far from being optimal with respect to the number of sets they cover individually. For instance, some of the functions map as many as $P/2$ keys to a single bucket, whereas we know from

§ 4 that a function covering the maximal number of sets distributes keys evenly among the buckets.

Thus our analysis gives hope that even minimal perfect hashing can be practical for key sets of limited size, and that increasing the number of buckets used can substantially increase the size limit. However, it also shows that if the entire parameter of a parameterized *phf* must be used in evaluating the function, then that form of *phf* can be useful (i.e., better than binary search) *only* for limited-size key sets.

Acknowledgments. We would like to thank Mark Wegman and Tom Leighton for helpful discussions on parts of the material.

Afterword. Since the submission of this paper, many other researchers have reported findings on various aspects of perfect hashing. Among those with results bearing on the issue of the minimal size for a perfect collection are Fredman, Komlos, and Szemeredi [Fre84] and Mehlhorn [Meh82]. In terms of the notation we have used, Fredman et al. mention that R. Graham has obtained an upper bound on $Minf(k, k, P)$ that is approximately e^k. Mehlhorn proved a similar but more general result: $Minf(k, b, P) \leqq k \cdot \log P \cdot e^{k^2/b}$. This seems to pretty completely pin down *Minf*.

The main contribution of Fredman et al. is to the practical side of the issue: they give a way to construct perfect hashing functions with $b < 3k$. The parameters used to describe the function occupy, at worst, a total of $(k+1) \log P$ bits. Significantly, though, only $2 \log P$ bits need to be examined during any one function evaluation. Hence, their scheme has a worst-case running time which, with respect to k, is $O(1)$.

REFERENCES

[And79] M. R. ANDERSON AND R. SPRUGNOLI, *unpublished notes*, 1979.

[Ber82] F. BERMAN, M. E. BOCK, E. DITTERT, M. J. O'DONNELL AND D. PLANK, *Collections of functions for perfect hashing*, Tech. Rept. CSD-TR-408, Dept. Computer Sciences, Purdue University, West Lafayette, IN, 1982.

[Ber81] F. BERMAN, M. E. BOCK AND D. PLANK, *unpublished notes*, 1981.

[Bos60] R. C. BOSE, S. S. SHRIKHANDE AND E. T. PARKER, *Further results on the construction of mutually orthogonal latin squares and the falsity of Euler's conjecture*, Canad. J. Math., 12 (1960), pp. 189-203.

[Car77] J. L. CARTER AND M. N. WEGMAN, *Universal classes of hash functions*, Proc. Ninth Annual Symposium on the Theory of Computing, May 1977, pp. 106-112.

[Cic80] R. CICHELLI, *Minimal perfect hash functions made simple*, Comm. ACM, 23 (1980), pp. 17-19.

[Com82] D. COMER AND M. J. O'DONNELL, *Geometric problems with application to hashing*, this Journal, 11 (1982), pp. 217-226.

[Dit82] E. DITTERT, *On the feasibility of a method for recognizing elements of a set of data keys*, Ph.D. dissertation, Dept. Computer Sciences, Purdue Universtiy, West Lafayette, IN, 1982.

[Fre84] M. L. FREDMAN, J. KOMLOS AND E. SZEMEREDI, *Storing a sparse table with $O(1)$ worst case access time*, J. Assoc. Comput. Mach., 31 (1984), pp. 538-544.

[Jae81] G. JAESCHKE, *Reciprocal hashing: A method for generating minimal perfect hashing functions*, Comm. ACM, 24 (1981), pp. 829-833.

[Meh82] K. MEHLHORN, *On the program size of perfect and universal hash functions*, Proc. 23rd Annual Symposium on the Foundations of Computer Science, November 1982, pp. 170-175.

[Spr77] R. SPRUGNOLI, *Perfect hashing functions: A single probe retrieving method for static sets*, Comm. ACM, 20 (1977), pp. 841-850.

[Tar79] R. E. TARJAN AND A. C. YAO, *Storing a sparse table*, Comm. ACM, 22 (1979), pp. 606-611.

[Yao81] A. C. YAO, *Should tables be sorted?*, J. Assoc. Comput. Mach., 28 (1981), pp. 615-628.

SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
023

RECOGNIZING COMPOSITE GRAPHS IS EQUIVALENT TO TESTING GRAPH ISOMORPHISM*

JOAN FEIGENBAUM† AND ALEJANDRO A. SCHÄFFER†

Abstract. We consider *composition*, a graph multiplication operator defined by Harary and Sabidussi, from a complexity theoretic point of view. If G and H are undirected graphs without self-loops, then the composite graph $G[H]$ has vertex set $V(G)\times V(H)$ and edge set $\{(g_1,h_1)$—(g_2,h_2): g_1—$g_2\in E(G)$ or $g_1=g_2$ and h_1—$h_2\in E(H)\}$. We show that the complexity of testing whether an arbitrary graph can be written nontrivially as the composition of two smaller graphs is the same, to within polynomial factors, as the complexity of testing whether two graphs are isomorphic.

Key words. graph isomorphism, graph products, graph composition, graph algorithms, computational complexity.

AMS Subject Classifications. 02E10: Algorithms, 05C99: Graph Theory, 68A10: Algorithms, 68A20: Computational Complexity and Efficiency.

In this paper, we investigate one of the graph products defined in [6] and [9] from a complexity theoretic point of view. We refer to this product as *composition* and call a graph that cannot be expressed as the composition of two smaller graphs *irreducible*. In [5], Garey and Johnson used composition to study approximation algorithms for graph coloring, and, in [8], Papadimitriou and Yannakakis used it to show that EXACT CLIQUE is complete for a class called D^P, but they did not address the question of how difficult it is to decide whether a graph is irreducible. Our main result is a proof that testing a connected graph for irreducibility is polynomial-time equivalent to testing whether two connected graphs are isomorphic. In the proof, we provide an algorithm that uses a polynomial number of calls to a graph isomorphism subroutine and either determines that the input graph is irreducible or yields a factorization. This distinguishes the graph-factoring problem from the integer-factoring problem, where factoring and primality testing are not known to have the same complexity. In the following discussion, all graphs are undirected and are assumed to be connected unless otherwise specified.

DEFINITION. Given two graphs G_1 and G_2, construct the *composition* $G = G_1[G_2]$ as follows: For each node in G_1, insert a copy of G_2. If two copies correspond to nodes that are adjacent in G_1, then draw in all possible edges x — y such that x is in one copy and y is in the other. We refer to these as *adjacent copies* and call G_1 and G_2 the *left* and *right factors* of G, respectively. We prefer the notation x — y to

* This work was done while the authors were summer employees of AT&T Bell Laboratories. During the academic year, the first author is supported by a Xerox Corporation Fellowship and the second by a National Science Foundation Graduate Fellowship and NSFDCR 83-00984. The preparation of the Figures was paid for by NSFDCR 83-08109. The text of this paper was typeset at the Stanford University Computer Science Department using TEX. 'TEX' is a trademark of the American Mathematical Society.

† Computer Science Department, Bldg. 460, Stanford University, Stanford, California 94305.

the more common (x, y) because (x, y) has many other meanings, x — y emphasizes that the graph is undirected, and x — y readily extends to x —— y to denote a path from x to y.

EXAMPLE. Figure 1 shows the composition of a three-node chain and a triangle.

Observe that, unlike the more common graph products of [6], [9], and [10], composition is not commutative. The smallest counterexample to commutativity is $G_1[G_2] \ncong G_2[G_1]$, where G_1 is the two-node graph with one edge and G_2 is the two-node graph with no edge. For graphs G_1 and G_2 on more than one node, $G_1[G_2] \cong G_2[G_1]$ if and only if $G_1 \cong G_2$ or G_1 and G_2 are both complete or both empty [6].

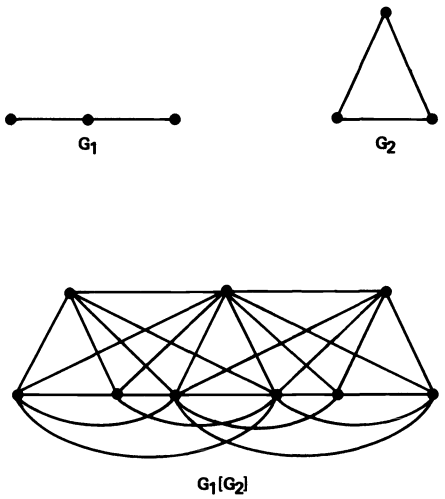


FIG. 1. *The composition of a three-node chain and a triangle.*

Having defined the composition, we naturally ask how difficult it is to test a graph for irreducibility. In Theorems 1 and 2, we show that the complexity of irreducibility testing is the same to within polynomial factors as that of testing graph isomorphism. We first show that certain useful restrictions on the graph isomorphism problem do not reduce its complexity.

LEMMA 1. *Consider instances of the graph isomorphism problem in which p, the number of nodes in each graph, is an odd prime, both graphs are connected, and no node in either graph has degree greater than* $\frac{p}{2}$. *This restricted version is at least as hard as the general graph isomorphism problem.*

Proof. We reduce an instance $G_1? \cong G_2$ of the general graph isomorphism problem to an instance $H_1? \cong H_2$ of the restricted problem. Suppose that each of G_1 and G_2 has n nodes. We know that for any positive integer n, there is at least one odd prime p in the interval $[2n + 3, 4n + 6)$ [7]. The first step in our reduction is to

find the smallest such p; we can do this deterministically by trial division of successive odd integers $2n+3$, $2n+5$, etc., each in time polynomial in n. Next we transform G_1 into a graph H_1 on p nodes as follows: Call G_1's nodes $x_1, x_2, \ldots, x_n$ and create a set of $p-n$ new nodes $x_{n+1}, \ldots, x_p$. Add an edge x_i — x_{n+1} for each $i \leq n$ and an edge x_i — x_{i+1} for each i, $n+1 \leq i \leq p-1$. Use the same procedure to transform G_2 into a graph H_2 on p nodes; name H_2's nodes y_1 through y_p. Figure 2 shows the results of applying the transformation to two four-node graphs. Note that H_1 and H_2 will always be connected.

We can perform this reduction in time polynomial in n. The maximum degree nodes in the resulting graphs are x_{n+1} and y_{n+1}, which have degree $n+1$; because $p \geq 2n+3$, this means that no node has degree greater than $\frac{p}{2}$. It is clear that if $G_1 \cong G_2$, then $H_1 \cong H_2$. Suppose conversely that $H_1 \cong H_2$. In each of H_1 and H_2, there is a unique set X of n nodes, precisely those nodes that induce G_1 and G_2, satisfying the following conditions: there is exactly one node $x \notin X$ such that x is adjacent to every node in X, and no node in X is adjacent to any node not in X except x. In H_1, $X = X_1 = \{x_1, \ldots, x_n\}$ and in H_2, $X = X_2 = \{y_1, \ldots, y_n\}$. Hence, any isomorphism from H_1 onto H_2 must map the subgraph induced by X_1 onto the subgraph induced by X_2, and these subgraphs are G_1 and G_2 respectively. □

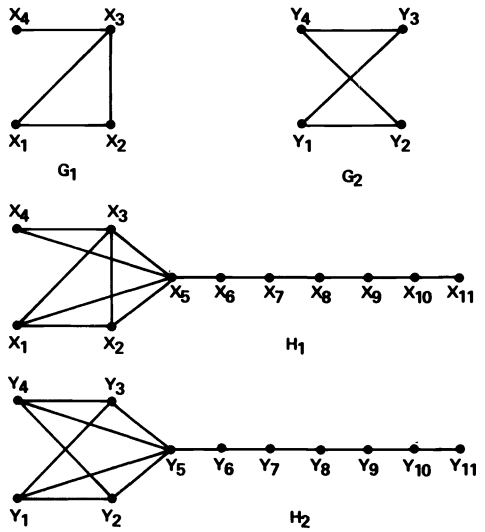


FIG. 2. *The reduction in the proof of Lemma* 1 *transforms* G_1 *and* G_2 *into* H_1 *and* H_2.

Armed with the fact that the restrictions of Lemma 1 do not reduce the complexity of the graph isomorphism problem, we proceed to our analysis of the complexity of irreducibility testing.

THEOREM 1. *Testing a connected graph G for irreducibility is at least as difficult as graph isomorphism.*

Proof. It suffices to reduce an instance $G_1? \cong G_2$ of graph isomorphism satisfying the conditions of Lemma 1 to an instance of irreducibility testing such that $G_1 \cong G_2$ if and only if the resulting graph is composite. To do this, form a graph G that

resembles $G_1[G_1]$ as follows: Replace $p-1$ of the nodes in G_1 with copies of G_1, and replace the remaining node with a copy of G_2. If adjacent nodes v and w in G_1 have been replaced by graphs V and W, then draw in an edge a — b of G for each $a \in V$ and $b \in W$. If $G_1 \cong G_2$, then $G \cong G_1[G_1]$, so G is composite.

To prove the converse, suppose that G is composite. Because G is connected, we know that if $G = H_1[H_2]$, then its left factor H_1 is also connected. Assume first that H_2 is also connected; we will show that under these assumptions, $H_1 \cong H_2 \cong G_1 \cong G_2$. Observe that G has p^2 nodes, where p is prime; so if it factors nontrivially its factors must have p nodes each.

Let x be an arbitrary node in G. We will give a polynomial-time algorithm to identify a unique set of p nodes, H_2^x, the copy of the right factor that contains x. H_2 is connected; hence, it suffices to show that we can distinguish between neighbors of x that belong to the copy H_2^x and neighbors that belong to other copies. We denote by $N(x)$ the set of all neighbors of x.

Let the degree of x in G be d. Because of the way we constructed G, we know that $d = d_1 p + d_2$, where $0 < d_1, d_2 < \frac{p}{2}$. For $G = H_1[H_2]$ with p nodes in the right factor H_2, that means x has d_2 neighbors in its copy H_2^x and that H_2^x is adjacent to d_1 other copies. For all x, both d_1 and d_2 are strictly less than $\frac{p}{2}$; hence all nodes in H_1 and H_2 have degree less than $\frac{p}{2}$.

If y is a neighbor of x in H_2^x and H_2^z is a copy of H_2 that is adjacent to H_2^x, then both x and y are adjacent to each node in H_2^z. In this case x and y have at least $d_1 p$ common neighbors. On the other hand, if y is adjacent to x but $H_2^x \neq H_2^y$, then H_2^x and H_2^y can *both* be adjacent to at most $d_1 - 1$ other copies of H_2 in G; hence other copies contribute at most $(d_1 - 1)p$ nodes to the set $N(x) \cap N(y)$. In addition, x and y are both adjacent to all nodes in $N(x) \cap H_2^x$ and all nodes in $N(y) \cap H_2^y$. There are fewer than p of these nodes because all nodes in H_2 have degree less than $\frac{p}{2}$.

Thus for each edge x — y, we determine in polynomial time that x and y are in the same copy if $|N(x) \cap N(y)| \geq d_1 p$ and that x and y are in different copies if $|N(x) \cap N(y)| < d_1 p$. Because H_2 is connected, we can find at least one node in H_2^x adjacent to x. Because x is arbitrary, we can do a similar common neighbor set computation to identify the remaining nodes in H_2^x. By construction, p nodes comprise a copy of G_2 and the remaining $(p-1)p$ nodes can be partitioned into copies of G_1; so if $G = H_1[H_2]$ and H_2 is connected, then $G_1 \cong G_2 \cong H_1 \cong H_2$.

If we drop the assumption that H_2 is connected, then we can use the same procedure to construct, for each $x \in G$, the connected component of H_2^x that contains x. If that component had fewer than p nodes, it would correspond to a component of G_1 or G_2 with fewer than p nodes, which is impossible because G_1 and G_2 are connected. Therefore, if $G = H_1[H_2]$, H_2 must be connected and $H_1 \cong H_2 \cong G_1 \cong G_2$. □

Several readers, including one of the referees, suggested that Theorem 1 could be proved with a much simpler argument as follows. Let $G_1 ? \cong G_2$ be an instance of graph isomorphism and reduce it to an instance of composite testing such that G has two connected components, G_1 and G_2. Then G is composite if and only if $G_1 \cong G_2$, in which case G is the composition of the empty graph on two nodes with G_1. We did not use this reduction because the G that it produces is not connected. It is important to show that recognizing composite graphs is difficult even for *connected* graphs, because this distinguishes composition from another definition of graph multiplication called cartesian product. In [4] and [12], Feigenbaum, Hershberger, Schäffer, and

Winkler show that connected cartesian-product graphs can be recognized in polynomial time, whereas recognizing disconnected cartesian-product graphs is at least as difficult as testing graph isomorphism. The proof for disconnected cartesian-product graphs is precisely the reduction just suggested.

Theorem 1 implies that if there were a polynomial-time algorithm for testing irreducibility, there would be one for testing graph isomorphism. Theorem 2 gives the reverse implication: Given an instance G of irreducibility testing, we can produce in polynomial time a polynomial number of graph isomorphism problems (whose sizes are polynomial in $|G|$) such that the answers indicate whether G is irreducible. The reduction is similar to the proof of sufficiency in Theorem 1 in that it entails taking an arbitrary node x in G and finding the nodes that would have to constitute G_2^x if G were the composition $G_1[G_2]$. Without the restrictions on G_1 and G_2 that we imposed in Theorem 1 that process is more complicated.

If n, the number of nodes in G, is not the square of a prime, then we must try all possible nontrivial factorizations of n as candidate sizes of the factors G_1 and G_2. This alone does not preclude a polynomial-time reduction, because there are only polynomially many different factorizations $n = n_1 n_2$. Observe that there may also be more than one way to partition the nodes of G into subgraphs G_2^x so that $G = G_1[G_2]$. Figure 3 shows two ways of doing so in the composition of a two-node chain with a four-node star. Note that the nodes A and B of the stars violate the degree condition of Lemma 1. In our proof of Theorem 2, we provide an algorithm for making consistent choices and discovering that such a graph is composite. Dörfler, Imrich, Coppersmith, and Feigenbaum characterize all the finite graphs G that have two inequivalent factorizations $G \cong G_1[G_2] \cong H_1[H_2]$ and show that for a fixed factorization $n_1 n_2 = |V(G)|$, there is at most one pair of graphs G_1, G_2 such that $G_1[G_2] \cong G$, $|V(G_1)| = n_1$, and $|V(G_2)| = n_2$ [2, 3].

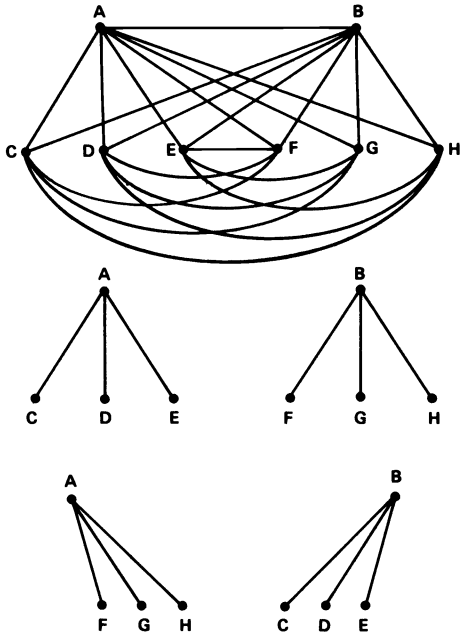


FIG. 3. *The composition of a two-node chain and a four-node star. There are two ways of partitioning the nodes into copies of the right factor.*

THEOREM 2. *Testing an n-node graph G for irreducibility is no more difficult than solving a polynomial number of graph isomorphism problems, each of whose size is polynomial in n.*

Proof. For each factorization $n = n_1 n_2$ such that $1 < n_1, n_2 < n$, we attempt to place each node x of G in a copy G_2^x of size n_2. If we succeed, we will have produced n_1 graphs of size n_2 and we can test whether they are pairwise isomorphic. We can also examine which copies are adjacent and construct a candidate for G_1; then we form $G_1[G_2]$ and test whether it is isomorphic to our original graph G. So in what follows, assume that we have fixed a factorization $n = n_1 n_2$. If the possible factorizations have been exhausted, then G is irreducible.

In the following algorithm, we start with no information about any edge x — y in G. For each edge x — y, we perform two polynomial-time tests, and in some cases the tests determine whether x and y are in the same copy of G_2. If the tests are conclusive, then we record the result, maintaining sets of nodes that can be subsets of copies of G_2 and pairs of nodes that cannot be in the same copy. There are three conditions under which we abandon the current factorization attempt and go on to the next possible factorization $n_1 n_2$: We create a copy with more than n_2 nodes; our tests tell us that some node x is in G_2^y but y is not in G_2^x; we find a node z that is in both G_2^x and G_2^y where $G_2^x \neq G_2^y$. We refer to all of these conditions as *contradictory states.*

Before presenting the algorithm, we show that we may assume without loss of generality that we are attempting to factor a connected graph G and that if $G = G_1[G_2]$, then both G_1 and G_2 are connected. First consider composite graphs $G = G_1[G_2]$ in which G_1 is not connected. For a fixed n_2, the right factor G_2 is unique up to isomorphism. If G_1 has m isolated nodes, $m \geq 1$, then G has a non-empty subgraph G' in which every component has at most n_2 nodes; otherwise all components of G have size greater than n_2. We divide G' into copies of G_2 as follows: Place the components in isomorphism classes, and say that there are p classes, with representatives $C_1, \ldots, C_p$. If the current values of m and n_2 are to yield a proper factorization, then the number of components in the i^{th} class must be $k_i m$, for some integer $k_i \geq 1$, and G_2 must be the disjoint union, for i between 1 and p, of k_i copies of C_i.

The subgraph $G \backslash G'$ is the part that corresponds to the non-isolated nodes of G_1. Here there is a one-to-one correspondence between the components of G and those of G_1, regardless of the number of components in G_2. Therefore, we can factor each component of G separately and, if each one factors nontrivially, check whether all of the right factors are isomorphic to each other and to the right factor of G'. If they are, then G is composite; otherwise we proceed to the next factorization $n = n_1 n_2$. Hence we can assume that G and G_1 are connected.

If $G = G_1[G_2]$ is connected but G_2 is not connected, then for each node $x \in G$ our factoring algorithm produces not the entire copy G_2^x but the connected component C_x of G_2^x that contains x. After placing all the nodes in components, we can put the components into isomorphism classes with a polynomial number of isomorphism tests. The number of components in each class must be divisible by n_1; once again, if a class has kn_1 members, then each copy of the right factor G_2 contains k of them. After putting components into isomorphism classes, we can use the following polynomial-time test to put the correct number of members of each class into each copy G_2^x and then check whether the copies have n_2 nodes each: If $G_2^x = G_2^y$ but x and y are in

different connected components of G_2^x, then $C_x \cap C_y = \emptyset$ and $N(x) \setminus C_x = N(y) \setminus C_y$. Conversely, if $C_x \cap C_y = \emptyset$ and $N(x) \setminus C_x = N(y) \setminus C_y$, then C_x and C_y are distinct components of copies G_2^x and G_2^y that have the same neighbor copies in G_1 and hence play interchangeable roles in G_1; so we can place C_x and C_y in the same copy without loss of generality. Thus we assume for simplicity of exposition of our algorithm that G is connected and that if it factors its right factor is also connected; so we can construct copies G_2^x by distinguishing between neighbors of x in the same copy and those in different copies.

In the first stage of the algorithm, we check whether $G = G_1[G_2]$ where G_1 is a complete graph. This special case can be recognized more easily than the general case and in fact our algorithm for the general case uses it as a subroutine. Let K_m denote the complete graph on m nodes. If $G = K_n$, then G is composite if and only if n is composite. Suppose that $G \neq K_n$ and consider the possibility that $G \cong K_{n_1}[G_2]$. If x is a node in such a graph, then all nodes *not* adjacent to x must be in G_2^x. Let G^c be the complement graph of G, that is, the graph with the same nodes as G that contains edge x — y if and only if G does not contain edge x — y. If $G \cong K_{n_1}[G_2]$, then G^c can be partitioned into n_1 copies of G_2^c such that no two nodes in different copies are adjacent. Therefore, we attempt to factor G by forming copies of G_2^c. This approach is inspired by Spinrad's algorithm for modular decomposition [11].

Find the connected components of G^c and place them in isomorphism classes; this can be done with a polynomial number of isomorphism tests. If $G \cong K_{n_1}[G_2]$, then each isomorphism class of components in G^c must have kn_1 members for some integer k, and there must be k of these members in each copy of G_2^c. It does not matter which k members are placed in a particular copy, because each node in a connected component of G^c is adjacent to every node of G outside that component. Hence we can finish this stage of the algorithm by checking whether the size of each isomorphism class is a multiple of n_1 and if so dividing the members evenly among the copies of G_2^c. If this gives us n_1 copies each with n_2 nodes, then $G \cong K_{n_1}[G_2]$.

In the remainder of the algorithm, we use the subroutine *TEST* to decide whether nodes can be in the same copy.

```
TEST(x, y)
{
    N := N(x) \ {y};
    G' := the subgraph of G induced by the vertices
        that remain after removing N from G;
    C := the connected component of G' that contains x;
    return(C);
}
```

Note that $TEST(x, y)$ is not necessarily the same as $TEST(y, x)$. Here is the final stage of the algorithm:

```
For each node x in G
{
    For each y in N(x)
        If |TEST(x, y)| > n2 or |TEST(y, x)| > n2
            Then mark x and y as members of different copies;
    If fewer than n2 neighbors of x remain unmarked
        Then mark all the remaining ones as members of G_2^x;
```

Else
{
$Y :=$ all unmarked neighbors of x;
$H := \bigcup_{y \in Y} y \cup TEST(x, y)$;
$m := |H|/n_2$;
Partition H into copies of G_2 using the algorithm for
graphs of the form $K_m[G_2]$;
}
If a contradictory state has been reached, stop and go on to the next $n_1 n_2$;
}

It remains to show that the main loop of our algorithm assigns nodes to copies correctly in the case that G_1 is not a complete graph. If $G_2^x = G_2^y$, then neither $TEST(x, y)$ nor $TEST(y, x)$ contains any nodes not in G_2^x; hence they are both of cardinality at most n_2, and x and y are not marked as members of different copies. If $G_2^x \neq G_2^y$ and G_2^y has at least one neighbor copy G_2^z that is not adjacent to G_2^x in G_1, then $TEST(x, y)$ contains at least x, y, and all the nodes in G_2^z; thus it has cardinality greater than n_2, our algorithm marks x and y correctly, and the set Y in the "Else" clause will not contain any of these nodes. Similarly, we mark x and y correctly in the case that $G_2^x \neq G_2^y$ and G_2^x has at least one neighbor copy that is not adjacent to G_2^y in G_1. Therefore, the only neighbors of x that are unmarked after the execution of the inner loop of the algorithm are those in G_2^x or in copies that, together with G_2^x, form a clique of G_1; that is, if y and z are unmarked neighbors of x, neither is in G_2^x, and $G_2^y \neq G_2^z$, then G_2^y and G_2^z must be adjacent copies. For any unmarked neighbor y, $TEST(x, y)$ contains x, y, and all the nodes of G_2^x that are *not* adjacent to x. The reason for taking the union in the second statement of the "Else" clause is that Y contains all nodes in copies of G_2 in this clique *except* nodes in G_2^x that are not adjacent to x. Hence H is of the form $K_m[G_2]$ if G is of the form $G_1[G_2]$. □

Thus we can reduce an instance of irreducibility testing to a polynomial number of graph isomorphism problems each of size polynomial in the size of the original problem. We include the separate stage of the reduction algorithm that checks for composite graphs of the form $K_{n_1}[G_2]$ primarily for expository reasons, because the final stage reduces to that case if G is such a graph. For a given factorization $n = n_1 n_2$ the reduction takes polynomial time because the body of the inner loop is executed twice for each edge, the entire main loop is executed once for each vertex, and both involve only connected component, set-union, and various bookkeeping operations all of which can be done in polynomial time [1].

Finally, we remark that if G_2 is in a class of graphs that can be tested for isomorphism in polynomial time, for example, the class of trees or the class of bounded-degree graphs, then G can be tested for irreducibility in polynomial time.

Acknowledgements. We would like to thank our referees and Mihalis Yannakakis, Dave Johnson, and Jeff Lagarias of AT&T Bell Laboratories for uncovering mistakes and unnecessary complication in earlier versions of our proofs. We also thank David Fuchs of Stanford for invaluable help with TEX.

REFERENCES

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

[2] D. COPPERSMITH AND J. FEIGENBAUM, *Finite Graphs with Two Inequivalent Factorizations under the Composition Operator*, IBM Research Report RC 11149, Yorktown Heights, 1985.

[3] W. DÖRFLER AND W. IMRICH, *Das lexikographische Produkt gerichteter Graphen*, Monatshefte für Math, 76 (1972), pp. 21–30.

[4] J. FEIGENBAUM, J. HERSHBERGER AND A. A. SCHÄFFER, *A Polynomial Time Algorithm for Finding the Prime Factors of Cartesian Product Graphs*, Discrete Appl. Math., 12 (1985), pp. 123–138.

[5] M. R. GAREY AND D. S. JOHNSON, *The Complexity of Near-Optimal Graph Coloring*, J. Assoc. Comput. Mach., 23 (1976), pp. 43–49.

[6] F. HARARY, *On the Group of the Composition of Two Graphs*, Duke Math J., 26 (1959), pp. 29–34.

[7] I. NIVEN AND H. S. ZUCKERMAN, *An Introduction to the Theory of Numbers*, 3rd ed., John Wiley, New York, 1972.

[8] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *The Complexity of Facets (and Some Facets of Complexity)*, J. Comput. System Sci., 28 (1984), pp. 244–259.

[9] G. SABIDUSSI, *The Composition of Graphs*, Duke Math. J., 26 (1959), pp. 693–696.

[10] ———. *Graph Multiplication*, Math. Z., 72 (1960), pp. 446–457.

[11] J. SPINRAD, *Transitive Orientation in $O(n^2)$ Time*, Proc. 15fth Annual STOC, Boston, 1983, pp. 457–465.

[12] P. WINKLER, *Factoring a Graph in Polynomial Time*, submitted for publication.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics

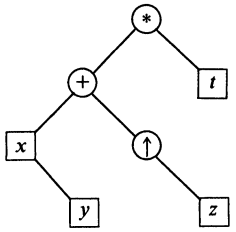
REGISTER ALLOCATION FOR UNARY–BINARY TREES*

P. FLAJOLET† AND H. PRODINGER‡

Abstract. We study the number of registers required for evaluating arithmetic expressions formed with any set of unary and binary operators. Our approach consists in a singularity analysis of intervening generating functions combined with a use of (complex) Mellin inversion. We illustrate it first by rederiving the known results about binary trees and then extend it to the fully general case of unary-binary trees. The method used, as mentioned in the conclusion, is applicable to a wide class of combinatorial sums.

Key words. analysis of algorithms, register allocation, random trees, Mellin transform

1. Introduction. An arithmetic expression with only binary operations may be described as a *binary tree.* For instance, $(x+y \uparrow z) * t$ corresponds to



The problem of *register allocation* consists in finding an *evaluation strategy* for arithmetic expressions using only binary operations applied to elements of an array called *registers.* For the above expression with registers being an array $R[0], R[1], \cdots$ a possible evaluation strategy is

$$\begin{aligned} &R[0] \leftarrow x \\ &R[1] \leftarrow y \\ &R[2] \leftarrow z \\ &R[1] \leftarrow R[1] \uparrow R[2] \\ &R[0] \leftarrow R[0] + R[1] \\ &R[1] \leftarrow t \\ &R[0] \leftarrow R[0] * R[1] \end{aligned}$$

There is an optimal strategy with respect to the number of registers used. That strategy has been found by Ershov as early as 1958 [5] and is described by Sethi and Ullman in [23]. The minimal number of registers necessary to keep intermediate results is called the register function of the tree t, and is denoted by Reg (t). This function may be defined recursively as follows:

$$\text{Reg}(\square) = 0,$$

$$\text{Reg}\left(\begin{array}{ccc} & \circ & \\ t_1 & & t_2 \end{array}\right) = \begin{cases} 1 + \text{Reg}(t_1) & \text{if Reg}(t_1) = \text{Reg}(t_2), \\ \max\{\text{Reg}(t_1), \text{Reg}(t_2)\} & \text{otherwise.} \end{cases}$$

The average number D_n of registers needed to evaluate a binary tree of size n (i.e. n internal nodes) assuming that all binary trees of size n are equally likely is a

* Received by the editors January 5, 1984, and in revised form February 19, 1985.
† INRIA, Rocquencourt, 78150 Le Chesnay, France.
‡ Technical University Vienna, Gußhausstraße 27-29, A-1040 Vienna, Austria.

well studied quantity [7], [12], [16]. It satisfies

$$D_n = \log_4 n + D(\log_4 n) + O\left(\frac{\log^* n}{\sqrt{n}}\right),$$

where D is a periodic function with period 1 and known Fourier coefficients and $\log^* n$ denotes an unspecified power of $\log n$ (usually different powers in different situations).

The aim of the present paper is twofold. Firstly, we give an alternative proof of this result, which is based on an analytic technique "à la Odlyzko" that has proved to be very helpful in tree enumeration problems (see [9], [17]); this alternative proof permits us if needed to derive asymptotic expansions of D_n to any order.

Then we show that this approach extends easily to more general classes of trees: assume that unary operations like $-$, sin, exp, log, etc, $\cdots$, are also permitted. There we have to deal with unary-binary trees, possibly with weights, according to the number of unary and binary operations allowed. (See § 3 for precise definitions.)

The register function is also defined on *unary-binary trees* in an obvious way: it is clear that unary nodes do not affect the register function. More precisely, for a unary-binary tree t, the register function $\mathrm{Reg}(t)$ is defined inductively by:

$$\mathrm{Reg}(\square) = 0,$$

$$\mathrm{Reg}\left(\begin{array}{c} \circ \\ | \\ t \end{array}\right) = \mathrm{Reg}(t),$$

$$\mathrm{Reg}\left(\begin{array}{ccc} & \circ & \\ t_1 & & t_2 \end{array}\right) = \begin{cases} 1 + \mathrm{Reg}(t_1) & \text{if } \mathrm{Reg}(t_1) = \mathrm{Reg}(t_2), \\ \max\{\mathrm{Reg}(t_1), \mathrm{Reg}(t_2)\} & \text{otherwise.} \end{cases}$$

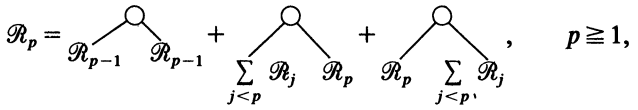
In § 3 we consider the average number of registers needed to evaluate a unary-binary tree. The analysis that we develop for binary trees (§ 2) can be translated to this more general case since the unary-binary trees are obtained from the binary trees by a simple substitution operation. As a consequence, all the generating functions needed for the analysis are obtained from the corresponding ones for binary trees via a simple substitution.

The *singularity analysis* that we are going to use in this paper is based on an extension to complex arguments of the *Mellin transform* inversion theorem. It can be applied to several problems in the analysis of algorithms. We mention height of trees [2], register allocation [7], [12], [16] and odd-even merge [8], [19]. The advantage is that asymptotic expansions to any order can be derived rather simply, as the generating functions are usually much easier to approximate in a neighbourhood of their singularities than their Taylor coefficients; also Mellin transform techniques constitute a rather powerful tool when dealing with number theoretic functions (here the dyadic valuation).

2. The register function of binary trees revisited. In order to rederive the formula for D_n, we need several generating functions, which can be most easily obtained by a simple translation from so-called symbolic equations [6]: If A and B are families of trees, then we write $\begin{array}{ccc} & \circ & \\ A & & B \end{array}$ for the set of all trees consisting of a root, a left subtree $t_1 \in A$ and a right subtree $t_2 \in B$. The family $\mathcal{B}$ of binary trees is then described by the symbolic equation

$$\mathcal{B} = \square + \begin{array}{ccc} & \circ & \\ \mathcal{B} & & \mathcal{B} \end{array}.$$

If we define the family $\mathcal{R}_p$ to be the family of all binary trees t with $\mathrm{Reg}(t) = p$, then the definition of the register function easily carries over to:



$\mathcal{R}_0 = \square$.

Let $R_p(z)$ denote the generating function of the family $\mathcal{R}_p$, i.e.

$$R_p(z) = \sum_{t \in \mathcal{R}_p} z^{\mathrm{size}(t)}.$$

It is known [7], [12], [16] that

$$R_p(z) = \frac{z^{2^p-1}}{F_{2^{p+1}}(z)},$$

where $F_i(z)$ is the ith Fibonacci polynomial:

$$F_i(z) = \frac{y^i - \bar{y}^i}{y - \bar{y}}, \quad \text{with } y = \frac{1+r}{2}, \quad \bar{y} = \frac{1-r}{2}, \quad r = r(z) = \sqrt{1-4z}.$$

The generating function of the cumulated register values is

$$E(z) = \sum_{p \geqq 1} p \cdot R_p(z).$$

The sought average D_n is then

$$D_n = \frac{[z^n]E(z)}{[z^n]B(z)},$$

where

$$B(z) = \sum_{t \in \mathcal{B}} z^{\mathrm{size}(t)}$$

is the generating function of all binary trees and $[z^n]f$ denotes the nth Taylor coefficient of the power series f.

From the defining equation for $\mathcal{B}$ one obtains immediately

$$B(z) = 1 + z(B(z))^2, \quad \text{or } B = (1 - r(z))/2z.$$

Using the substitution [2]

$$z = \frac{u}{(1+u)^2} \leftrightarrow u = \frac{1-r}{1+r},$$

we easily find

$$E(z) = \frac{1-u^2}{u} \sum_{p \geqq 1} p \frac{u^{2^p}}{1-u^{2^{p+1}}} = \frac{1-u^2}{u} \sum_{k \geqq 1} v_2(k) u^k,$$

where $v_2(k)$ is the dyadic valuation of k, defined as

$$v_2(k) = \max\{i \mid 2^i \text{ divides } k\}.$$

We want to extract $[z^n]E(z)$ by means of Cauchy's formula, viz.

$$[z^n]E(z) = \frac{1}{2\pi i} \int_\Gamma E(z) \frac{dz}{z^{n+1}}, \tag{1}$$

where Γ is a path as depicted in Fig. 1.

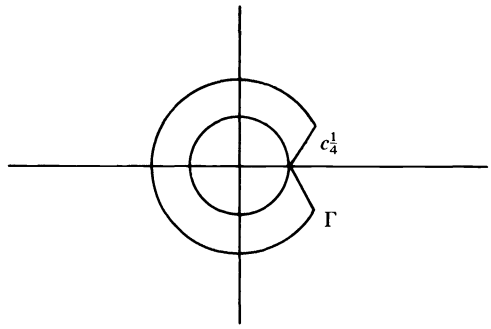


FIG. 1

To be more precise, let $0<\theta<\pi/2$, $\omega>0$ and $\rho>\frac{1}{4}$. Then $\Gamma=\Gamma_0\cup\Gamma_1\cup\Gamma_2$ with

$$\Gamma_0=\{z:\ |z-\tfrac{1}{4}|=\omega,\ |\mathrm{Arg}\,(z-\tfrac{1}{4})|>\theta\},$$
$$\Gamma_1=\{z:\ |z-\tfrac{1}{4}|\geqq\omega,\ |z|<\rho,\ |\mathrm{Arg}\,(z-\tfrac{1}{4})|=\theta\},$$
$$\Gamma_2=\{z:\ |z|=\rho,\ |\mathrm{Arg}\,(z-\tfrac{1}{4})|\geqq\theta\}.$$

For this, we have to show that $E(z)$ has an appropriate analytic continuation in a domain which properly contains Γ.

Following the general strategy developed in [9], we can, provided we have an approximation of $E(z)$ about $\frac{1}{4}$, "translate" it to an approximation of the coefficients $[z^n]E(z)$ given by the Cauchy integral (1). This is a fairly straightforward process once the approximation of $E(z)$ is known. So our task is reduced to the problem of obtaining an expansion of the form:

$$E(z)\sim\alpha\cdot r\cdot\log r+\beta\cdot r+\cdots,$$

where $r=\sqrt{1-4z}$, in a sector about $\frac{1}{4}$ which contains the line segment of Γ; the contribution of the Cauchy integral (1) of the part of the circle with radius $>\frac{1}{4}$ is negligible.

Now, since

$$\sum_{k\geqq1}v_2(k)u^k=\frac{u^2}{1-u^2}+\frac{u^4}{1-u^4}+\frac{u^8}{1-u^8}+\cdots,$$

the unit circle $|u|=1$ is a natural boundary of this function. The nature of the mapping $z=z(u)$ is such that the boundary of the unit circle in the u-plane is mapped on the halfray $\mathrm{Re}\,(z)\geqq\frac{1}{4}$, $\mathrm{Im}\,(z)=0$, and this halfray thus constitutes a *natural boundary* for $E(z)$. From the preceding remark we are free to choose any contour that simply encircles the origin without crossing the halfray and in particular we can take the contour Γ of Fig. 1.

What remains to do is thus to find a local expansion of $E(z)$ about $z=\frac{1}{4}$. This will be done by the use of the Mellin transform. (See [4], [20] for more information about the Mellin transform and [6], [18] for some applications in Computer Science.)

We set $u=e^{-t}$ and

$$V(t)=\sum_{k\geqq1}v_2(k)\,e^{-kt},\qquad V^*(s)=\int_0^\infty x^{s-1}V(x)\,dx.$$

Since $v_2(2k) = 1 + v_2(k)$ and $v_2(2k+1) = 0$, we easily find

$$\sum_{k \geq 1} v_2(k) k^{-s} = \frac{\zeta(s)}{2^s - 1},$$

and so

$$V^*(s) = \frac{\Gamma(s)\zeta(s)}{2^s - 1}, \qquad \mathrm{Re}\,(s) > 1.$$

The Mellin inversion formula gives

$$V(t) = \frac{1}{2\pi i} \int_{2-i\infty}^{2+i\infty} V^*(s) t^{-s}\, ds,$$

and we can shift the line of integration to the left as far as we please if we only take the residues into account.

The reader might be puzzled that we use the Mellin transform of functions of a complex variable. But we actually do not need more than

$$e^{-t} = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} \Gamma(s) t^{-s}\, ds, \qquad \mathrm{Re}\,(t) > 0,\ c > 0;$$

a reference for this is for example [1, p. 91].

Thus we find an asymptotic series for $V(t)$ via

$$V(t) \sim \sum_{\mathrm{Re}\,(s) \leq 1} \mathrm{Res}\,(V^*(s) t^{-s}).$$

The main contributions come from $s = 1$, $s = 0$, $s = 2k\pi i/\log 2$, $(k \neq 0)$. The residue at $s = 1$ is easily found to be

$$\frac{1}{t}.$$

By using local expansions of $\Gamma(s)$, $\zeta(s)$, $(2^s - 1)^{-1}$ and t^{-s} we find the residue at $s = 0$, resp. at $s = \chi_k := 2k\pi i/\log 2$:

$$\frac{1}{2}\log_2 t - \frac{1}{2}\log_2 2\pi + \frac{1}{4} + \frac{\gamma}{2\log 2}$$

and the residue at $s = \chi_k$.

$$c_k t^{-\chi_k} \quad \text{with} \quad c_k = \frac{1}{\log 2}\Gamma(\chi_k)\zeta(\chi_k).$$

Putting things together, we find $(z \to 1/4$, i.e. $t \to 0)$

$$E(z) = t \cdot \log_2 t + K \cdot t + \sum_{k \neq 0} 2c_k t^{1-\chi_k} + 2 + O(t^2)$$

with

$$K = -\log_2 2\pi + \frac{1}{2} + \frac{\gamma}{\log 2}.$$

Now $e^{-t} = u$ and $u = (1-r)/(1+r)$ with $r = \sqrt{1-4z}$; thus

$$t = -\log\frac{1-r}{1+r} = 2r + O(r^3),$$

yielding

$$E(z)=2r\log_2 r+2(K+1)r+4\sum_{k\neq 0}c_k r^{1-\chi_k}+2+O(r^2).$$

So we find an asymptotic expansion for $[z^n]E(z)$, as announced earlier, by looking at $[z^n]2r\log_2 r$, $[z^n]2(K+1)r$ and so on. For this, we refer to [9], [10], [11], [13]:

$$[z^n](1-z)^\alpha=\frac{n^{-\alpha-1}}{\Gamma(-\alpha)}\left(1+O\left(\frac{1}{n}\right)\right),\qquad \alpha\neq 0,1,2,\cdots,$$

$$[z^n]\log(1-z)\cdot(1-z)^\alpha=\frac{-n^{-\alpha-1}\log n}{\Gamma(-\alpha)}+\frac{n^{-\alpha-1}}{(\Gamma(-\alpha))^2}\Gamma'(-\alpha)+O\left(\frac{\log^* n}{n^{\alpha+2}}\right).$$

Using known values of $\Gamma(-\frac{1}{2})$ and $\Gamma'(-\frac{1}{2})$ this gives us

$$[z^n]\log(1-z)\cdot(1-z)^{1/2}=\frac{n^{-3/2}\log n}{2\sqrt{\pi}}+\frac{n^{-3/2}}{2\sqrt{\pi}}(\gamma+2\log 2-2)+O\left(\frac{\log^* n}{n^{5/2}}\right).$$

Hence

$$D_n=\frac{\log n}{2\log}+\frac{1}{\log 2}\left(\frac{\gamma}{2}+\log 2-1\right)-(K+1)+4\sqrt{\pi}\sum_{k\neq 0}\frac{c_k n^{\chi_k/2}}{\Gamma((\chi_k-1)/2)}+O\left(\frac{\log^* n}{n}\right).$$

Using the duplication formula for the gamma function [21], we can simplify:

$$\frac{4\sqrt{\pi}c_k}{\Gamma((\chi_k-1)/2)}=\frac{\zeta(\chi_k)\Gamma(\chi_k/2)(\chi_k-1)}{\log 2}.$$

Finally we notice that $n^{\chi_k/2}=e^{2k\pi i\cdot\log_4 n}$ and state [7], [12]:

THEOREM 1. *The average number of registers to evaluate a binary tree with n nodes is given by*

$$D_n=\log_4 n+D(\log_4 n)+O\left(\frac{\log^* n}{n}\right),$$

where $D(x)$ is a periodic function with period 1. This function can be expanded as a convergent Fourier series $D(x)=\sum_{k\in Z}d_k e^{2k\pi ix}$, and

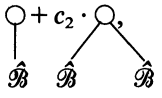
$$d_0=-\frac{1}{2}-\frac{\gamma}{2\log 2}-\frac{1}{\log 2}+\log_2 2\pi,$$

$$d_k=\frac{1}{\log 2}\zeta(\chi_k)\Gamma\left(\frac{\chi_k}{2}\right)(\chi_k-1),\qquad k\neq 0,\quad \chi_k=\frac{2k\pi i}{\log 2}.$$

Remark that the constant d_0 was erroneously stated in [7].

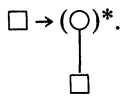
3. The register function of unary–binary trees. The symbolic equation:

$$\hat{\mathcal{B}}=c_0\cdot\square+c_1\cdot\bigcirc+c_2\cdot\bigcirc,$$

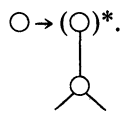


describes a family of unary-binary trees, where the weights fulfill $c_0>0$, $c_1\geqq 0$, $c_2>0$. The interpretation is that we have c_0 different types of nullary nodes, c_1 unary nodes and c_2 binary nodes. For example, if the set of operators and variables is $\{x, y, t, \pi; \sqrt{}, \log, \sin; +, \times\}$, then $c_0=4$, $c_1=3$ and $c_2=2$.

We can obtain $\hat{\mathcal{B}}$ from the family $\mathcal{B}$ of binary trees by means of the following substitution process: Above each leaf insert a sequence of unary nodes, viz.



Above each binary node insert a sequence of unary nodes, viz.



For plain binary trees, $yB(yz)$ is the series enumerating $\mathcal{B}$, where y marks a leaf and z marks an internal node. Thus the corresponding series for $\hat{\mathcal{B}}$ is obtained by the substitutions

$$y \to \frac{c_0 y}{1-c_1 z},$$

(2)

$$z \to \frac{c_2 z}{1-c_1 z}.$$

Let $\hat{B}$ be the generating function of the trees in $\hat{\mathcal{B}}$ and $\hat{R}_p$ be the generating function of the trees in $\hat{\mathcal{R}}_p$, i.e. the trees in $\hat{\mathcal{B}}$ with register function $= p$. Since the substitutions do not change the register function of the involved trees, we can find $\hat{\mathcal{R}}_p$ from $\mathcal{R}_p$ by the substitutions (2).

We can define the size of a tree in $\hat{\mathcal{B}}$ in two ways:

(1) we count leaves and internal nodes,

(2) we only count internal nodes.

In terms of generating functions (1) corresponds to the transformation

$$f(z) \to \hat{f}(z) = \frac{c_0 z}{1-c_1 z} f\left(\frac{c_0 c_2 z^2}{(1-c_1 z)^2}\right),$$

while (2) corresponds to:

$$f(z) \to \hat{f}(z) = \frac{c_0}{1-c_1 z} f\left(\frac{c_0 c_2 z}{(1-c_1 z)^2}\right).$$

We can treat both cases together by considering the more general transformation:

$$f(z) \to \hat{f}(z) = \frac{c_0 z + c_0'}{1-c_1 z} f\left(\frac{(c_0 z + c_0') c_2 z}{(1-c_1 z)^2}\right), \tag{3}$$

where $c_0,\ c_0' \geqq 0$ and $c_0 \neq 0 \Leftrightarrow c_0' = 0$. So all we have to do in order to compute the average register function $\hat{D}_n$ of all trees of size n is to perform the transformation in the expansion

$$E(z) = 2r \log_2 r + 2(K+1)r + 4 \sum_{k \neq 0} c_k r^{1-\chi_k} + 2 + O(r^2)$$

and in

$$B(z) = 2 - 2r + O(r^2).$$

We are interested in

$$\hat{D}_n = \frac{[z^n]\hat{E}(z)}{[z^n]\hat{B}(z)}.$$

Since the factor $(c_0z+c_0')/(1-c_1z)$ appears both in the numerator and the denominator and is regular at the dominant singularity of $\hat{\mathcal{B}}$, we can write

$$\hat{D}_n=\left[[z^n]E\left(\frac{(c_0z+c_0')c_2z}{(1-c_1z)^2}\right)\right]\Big/\left[[z^n]B\left(\frac{(c_0z+c_0')c_2z}{(1-c_1z)^2}\right)\right]\left(1+O\left(\frac{1}{n}\right)\right).$$

Let $\varphi(z)=(c_0z+c_0')c_2z/(1-c_1z)^2$. We have to express $r(\varphi(z))$ in terms of $\hat{r}=(1-z/\sigma)^{1/2}$, where σ is the singularity of $r(\varphi(z))$ nearest to the origin; σ plays the role that $\frac{1}{4}$ plays in the case of binary trees:

$$r(\varphi(z))=\frac{1}{1-c_1z}\sqrt{1-(2c_1+4c_0'c_2)z+(c_1^2-4c_0c_2)z^2};$$

σ is one of the solutions s_1, s_2 of

$$(c_1^2-4c_0c_2)z^2-(2c_1+4c_0'c_2)z+1=0, \qquad \text{i.e.}$$

$$s_{1,2}=\frac{c_1+2c_0'c_2\pm 2\sqrt{c_2}\cdot\sqrt{c_0'c_1+c_0'^2c_2+c_0}}{c_1^2-4c_0c_2}.$$

(a) We assume first that $c_1^2\neq 4c_0c_2$ and $c_1+2c_0'c_2>0$. Then $s_1\neq -s_2$. We set $\sigma=s_2$ and $\bar{\sigma}=s_1$. If $c_1^2<4c_0c_2$, then σ is the singularity closest to the origin and $|\bar{\sigma}|>|\sigma|$. If $c_1^2>4c_0c_2$, this is also true, because

$$\begin{aligned}&c_1+2c_0'c_2-2\sqrt{c_2}\sqrt{c_0'c_1+c_0'^2c_2+c_0}>0\\ &\Leftrightarrow c_1^2+4c_0'c_1c_2+4c_0'^2c_2^2>4c_0'c_1c_2+4c_0'^2c_2^2+4c_0c_2\\ &\Leftrightarrow c_1^2>4c_0c_2.\end{aligned}$$

So we have

$$(c_1^2-4c_0c_2)z^2-(2c_1+4c_0'c_2)z+1=(c_1^2-4c_0c_2)(z-\sigma)(z-\bar{\sigma})$$

and thus as $z\to\sigma$

$$\begin{aligned}(c_1^2-4c_0c_2)z^2-(2c_1+4c_0'c_2)z+1&\sim(c_1^2-4c_0c_2)(z-\sigma)(\sigma-\bar{\sigma})\\ &=4\sqrt{c_2}\sqrt{c_0'c_1+c_0'^2c_2+c_0}\cdot\sigma\cdot\left(1-\frac{z}{\sigma}\right).\end{aligned}$$

Hence

$$r(\varphi(z))\sim\frac{1}{1-c_1\sigma}\cdot 2\sqrt{\sigma}\cdot c_2^{1/4}(c_0'c_1+c_0'^2c_2+c_0)^{1/4}\sqrt{1-\frac{z}{\sigma}}=:A\cdot\sqrt{1-\frac{z}{\sigma}}.$$

(b) If $c_1^2=4c_0c_2$, then

$$r(\varphi(z))=\frac{1}{1-c_1\sigma}\cdot\sqrt{1-(2c_1+4c_0'c_2)z}=A\cdot\sqrt{1-\frac{z}{\sigma}}$$

with

$$\sigma=\frac{1}{2c_1+4c_0'c_2}\quad\text{and}\quad A=\frac{1}{1-c_1\sigma}.$$

(c) If $c_1+2c_0'c_2=0$, we have $\bar{\sigma}=-\sigma$. This means $c_1=0$ and $c_0'=0$, so that we have to consider

$$\frac{[z^n]c_0zE(c_0c_2z^2)}{[z^n]c_0zB(c_0c_2z^2)}=\frac{[z^{n-1}]E(c_0c_2z^2)}{[z^{n-1}]B(c_0c_2z^2)}.$$

In this case n has to be odd, $n=2N+1$, and we substitute $z^2=w$ and have to consider

$$\frac{[w^N]E(c_0c_2w)}{[w^N]B(c_0c_2w)},$$

which is as in the other cases.

In order to compute $\hat{D}_n$ up to a relative error of $O(1/n)$, we can use

$$\begin{aligned}
&\left[[z^n]\left(2A\sqrt{1-\frac{z}{\sigma}}\log_2\left(A\sqrt{1-\frac{z}{\sigma}}\right)+2(K+1)A\sqrt{1-\frac{z}{\sigma}}\right.\right.\\
&\qquad\left.\left.+4\sum_{k\neq 0}c_k\left(A\sqrt{1-\frac{z}{\sigma}}\right)^{1-\chi_k}\right)\right]\Bigg/\\
&\cdot\left[[z^n]-2A\sqrt{1-\frac{z}{\sigma}}\right]\\
&=\left[[z^n]\frac{1}{2\log 2}\sqrt{1-\frac{z}{\sigma}}\log\left(1-\frac{z}{\sigma}\right)+(K+1+\log_2 A)\sqrt{1-\frac{z}{\sigma}}\right.\\
&\qquad\left.+2\sum_{k\neq 0}c_k\left(\sqrt{1-\frac{z}{\sigma}}\right)^{1-\chi_k}A^{-\chi_k}\right]\Bigg/\\
&\cdot\left[[z^n]-\sqrt{1-\frac{z}{\sigma}}\right]\\
&=\log_4 n+D(\log_4 n-\log_2 A)-\log_2 A.
\end{aligned}$$

If we consider (according to case (c))

$$\log_4\frac{n-1}{2}+D\left(\log_4\frac{n-1}{2}-\log_2 A\right)-\log_2 A,$$

this is, up to a relative error of $O(1/n)$, equal to

$$\log_4 n+D(\log_4 n-\tfrac{1}{2}-\log_2 A)-\tfrac{1}{2}-\log_2 A.$$

This leads us to our main theorem.

THEOREM 2. *Given a family $\hat{\mathcal{B}}$ of unary-binary trees:*

$$\hat{\mathcal{B}}=c_0\cdot\square+c_1\cdot\bigcirc+c_2\cdot\bigcirc\!\!\bigcirc,\qquad c_0>0,\quad c_2>0,\quad c_1\geqq 0,$$

the average register function $\hat{D}_n$, where all trees of size n are equally likely (if the size is measured by the number of internal nodes and leaves, we set $c_0'=0$; if the size is just the number of internal nodes, we set $c_0':=c_0$ and $c_0:=0$), is given by:

(a) *If $c_1^2\neq 4c_0c_2$ and $c_1+2c_0'c_2>0$, set*

$$\sigma=\frac{c_1+2c_0'c_2-2\sqrt{c_2}\sqrt{c_0'c_1+c_0'^2c_2+c_0}}{c_1^2-4c_0c_2}$$

and

$$A=\frac{1}{1-c_1\sigma}2\sqrt{\sigma}\cdot c_2^{1/4}(c_0'c_1+c_0'^2c_2+c_0)^{1/4},$$

then

$$\hat{D}_n = \log_4 n + D(\log_4 n - \log_2 A) - \log_2 A + O\left(\frac{\log^* n}{n}\right), \qquad (n \to \infty).$$

(b) *If* $c_1^2 = 4c_0c_2$, *set*

$$\sigma = \frac{1}{2c_1 + 4c_0'c_2} \quad \text{and} \quad A = \frac{1}{1 - c_1\sigma}.$$

Then

$$\hat{D}_n = \log_4 n + D(\log_4 n - \log_2 A) - \log_2 A + O\left(\frac{\log^* n}{n}\right), \qquad (n \to \infty).$$

(c) *If* $c_1 + 2c_0'c_2 = 0$, *then for odd* n, *we have with* σ, A *defined as in* (a):

$$\hat{D}_n = \log_4 n + D\left(\log_4 n - \log_2 A - \frac{1}{2}\right) - \log_2 A - \frac{1}{2} + O\left(\frac{\log^* n}{n}\right), \qquad (n \to \infty).$$

Example. Let us consider the Motzkin trees, defined by:

$$\mathcal{M} = \square + \circ\!\!-\!\!\mathcal{M} + \begin{array}{c} \circ \\ \mathcal{M}\ \ \mathcal{M} \end{array}.$$

Let a leaf contribute to the size. The generating function of the numbers of Motzkin trees satisfies $M(z) = z(1 + M(z) + M(z)^2)$, whence

$$M(z) = \frac{1 - z - \sqrt{1 - 2z - 3z^2}}{2z},$$

$c_0 = 1$, $c_0' = 0$, $c_1 = 1$, $c_2 = 1$, $\sigma = 1/3$, $A = \sqrt{3}$. The average number $\hat{D}_n$ of registers needed to evaluate a Motzkin tree of size n is then

$$\hat{D}_n = \log_4 n + D\left(\log_4 n - \frac{1}{2}\log_2 3\right) - \frac{1}{2}\log_2 3 + O\left(\frac{\log^* n}{n}\right), \qquad (n \to \infty),$$

where $D(x)$ is the periodic function of Theorem 1. We may mention that

$$\log_4 n - \tfrac{1}{2}\log_2 3 = \log_4 \tfrac{4}{3}n.$$

4. Conclusions. The path we have taken is general enough to enable us to treat the asymptotics of sums of the form

$$S_n = \sum_{k \geq 1} a_k \binom{2n}{n-k} \tag{4}$$

(where instead of binomial coefficients, differences of binomial coefficients may appear), when $\{a_k\}_{k \geq 1}$ is an *arithmetic sequence*, i.e. a sequence such that the Dirichlet generating function

$$\alpha(s) = \sum_{k \geq 1} a_k k^{-s}$$

is meromorphic and well enough behaved towards $i\infty$. Such sums appear in the analysis of algorithms in at least the following three cases:

(1) height of trees [2];
(2) register allocation [7], [12], [16];
(3) odd-even merge [8], [19].

The methods that have been employed to analyse sums of the form (4) are:

(A) With the Gaussian approximation of binomial coefficients, replace the study of S_n in (4) by the study of $S^*(1/\sqrt{n})$ where:

$$S^*(x) = \sum_{k \geq 1} a_k e^{-k^2 x^2} \tag{5}$$

and use Mellin transform techniques to evaluate (5) asymptotically. This is the way taken originally by de Bruijn, Knuth and Rice [2] (problem 1), Kemp [12] (problem 2) and Sedgewick [19] (problem 3).

(B) Use real analysis to obtain real expressions for

$$a_k \quad \text{or } A_k^{(1)} = \sum_{j<k} a_j \quad \text{or} \quad A_k^{(2)} = \sum_{j<k} A_j^{(1)} \cdots .$$

Developments based on techniques of Delange constitute the original treatment of register allocation in [7] (problem 2), and have been applied to rederive Sedgewick's solution to [8] (problem 3). In the context of problem 1, they lead to an elementary derivation of the main terms of the expected height of general trees (this fact has been pointed out to us by L. Guibas).

(C) Use singularity analysis of the generating function of the S_n,

$$S(z) = \sum_{n \geq 0} S_n z^n$$

as we have done in this paper. The method has the advantage of allowing rather simply derivation of asymptotic expansions to any order and also generalises easily, as we have seen, to cases where binomial coefficients are replaced by trinomial coefficients or even more generally to coefficients of powers of some fixed function. It could therefore have been applied to problems 1 and 3 as well; interestingly enough, this is the way Knuth started his partial attack to problem 3 [14, ex. 5.2.2.16, p. 135 and p. 607].

Acknowledgment. We would like to thank one of the referees for pointing out a reference to a further paper dealing with register problems [22].

REFERENCES

[1] G. ANDREWS, *Theory of Partitions*, Academic Press, New York-London, 1976.

[2] N. G. DE BRUIJN, D. E. KNUTH AND S. O. RICE, *The average height of planted plane trees*, in Graph Theory and Computing, R. C. Read, ed., Academic Press, New York-London, 1972, pp. 15-22.

[3] H. DELANGE, *Sur la fonction sommatoire de la fonction somme des chiffres*, l'Enseignement Mathématique, 21 (1975), pp. 31-47.

[4] G. DOETSCH, *Handbuch der Laplace Transformation*, Birkhäuser, Basel, 1950.

[5] A. P. ERSHOV, *On programming of arithmetic operations*, Comm. ACM, 1 (1958), pp. 3-6.

[6] P. FLAJOLET, *Analyse d'algorithmes de manipulation d'arbres et de fichiers*, Cahiers du BURO, 34-35 (1981), pp. 1-209.

[7] P. FLAJOLET, J. C. RAOULT AND J. VUILLEMIN, *The number of registers required for evaluating arithmetical expressions*, Theoret. Comput. Science, 9 (1979), pp. 99-125.

[8] P. FLAJOLET AND L. RAMSHAW, *A note on Gray code and odd-even merge*, this Journal, 9 (1980), pp. 142-158.

[9] P. FLAJOLET AND A. ODLYZKO, *The average height of binary trees and other simple trees*, J. Comput. System Sci., 25 (1982), pp. 171-213.

[10] P. FLAJOLET AND C. PUECH, *Partial match retrieval of multidimensional data*, INRIA Research Report 233, Aug. 1983, extended abstract in 24th IEEE Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 282-288.

[11] R. JUNGEN, *Sur les séries de Taylor n'ayant que des singularités algébrico-logarithmiques sur leur cercle de convergence*, Comm. Math. Helvetici, 3 (1931), pp. 266-306.

[12] R. KEMP, *The average number of registers to evaluate a binary tree optimally*, Acta Inf., 11 (1979), pp. 363-372.
[13] P. KIRSCHENHOFER, *On the height of leaves in binary trees*, J. Combin. Inform. Syst. Sci., 8 (1983), pp. 44-60.
[14] D. E. KNUTH, *The Art of Computer Programming*, Vol. 3, Addison-Wesley, Reading, MA, 1973.
[15] A. MEIR AND J. W. MOON, *On the altitude of nodes in random trees*, Canad. J. Math., 30 (1978), pp. 997-1015.
[16] A. MEIR, J. W. MOON AND J. R. POUNDER, *On the order of random channel networks*, SIAM J. Alg. Disc. Meth., 1 (1980), pp. 25-33.
[17] A. ODLYZKO, *Periodic oscillations of coefficients of power series that satisfy functional equations*, Adv. Math., 44 (1982), pp. 180-205.
[18] M. REGNIER, *Evaluation des performances du hachage dynamique*, Thèse, Paris-Sud-Orsay, 1983.
[19] R. SEDGEWICK, *Data movement in odd-even merging*, this Journal, 7 (1978), pp. 239-272.
[20] I. SNEDDON, *The Use of Integral Transforms*, McGraw-Hill, New York, 1952.
[21] E. T. WHITTAKER AND G. N. WATSON, *A Course of Modern Analysis*, Cambridge Univ. Press, Cambridge, 1927.
[22] R. KEMP, *The reduction of binary trees by means of an input-restricted deque*, RAIRO Inform. Theor., 17 (1983), pp. 249-284.
[23] R. SETHI AND J. D. ULLMAN, *The generation of optimal code for arithmetic expressions*, J. Assoc. Comput. Mach., 17 (1970), pp. 715-728.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986

© 1986 Society for Industrial and Applied Mathematics

CONSTRUCTING $O(n \log n)$ SIZE MONOTONE FORMULAE FOR THE kth THRESHOLD FUNCTION OF n BOOLEAN VARIABLES*

JOEL FRIEDMAN†

Abstract. In this paper we construct formulae for the kth elementary symmetric polynomial of n Boolean variables, using only conjunction and disjunction, which for fixed k are of size $O(n \log n)$, with the construction taking time polynomial in n. We also prove theorems involving $n \log n \cdot$ (polynomial in k) upper bounds on such formulae. Our methods involve solving the following combinatorial problem: for fixed k and any n construct a collection of $r = O(\log n)$ functions $f_1, \cdots, f_r$ from $\{1, \cdots, n\}$ to $\{1, \cdots, k\}$ such that any subset of $\{1, \cdots, n\}$ of order k is mapped $1-1$ to $\{1, \cdots, k\}$ by at least one f_i.

Key words. monotone formula, threshold function, error-correcting codes, partitions

AMS(MOS) subject classification. 68C25

Introduction. For Boolean variables $x_1, \cdots, x_n$ we define the "threshold k" function

$$\mathrm{Th}_k(x_1, \cdots, x_n) \equiv \begin{cases} 1 & \text{if at least } k \text{ of the } x_1, \cdots, x_n \text{ are } 1, \\ 0 & \text{otherwise.} \end{cases}$$

In [Kr] Krichevskii proved that any monotone Boolean formula, i.e. formula using only conjunction and disjunction, for $\mathrm{Th}_2(x_1, \cdots, x_n)$ must be of size (i.e. number of occurrences of variables) $\Omega(n \log n)$, and hence so does $\mathrm{Th}_k(x_1, \cdots, x_n)$, $k > 2$. In [Kh] Khasin proves the existence of formulae for $\mathrm{Th}_k(x_1, \ldots, x_n)$ of size $O(n \log n)$ for fixed k; unfortunately the proof is not constructive (an exhaustive search for such a formula would take time exponential in n). Kleiman and Pippenger, in [K, P], have constructed formulae for $\mathrm{Th}_k(x_1, \cdots, x_n)$ of size $O(n \log n (\binom{k}{2})^{\log^* n})$ for fixed k. In this paper we construct monotone formulae for $\mathrm{Th}_k(x_1, \cdots, x_n)$ of size $O(n \log n)$ in time polynomial in n for fixed k. In addition we prove the existence of monotone formulae of size $O(k^{12.6} n \log n)$, which improves on Khasin's bound of $O(k^{O(\log k)} n \log n)$. We also prove constructions for formulae of size $O(k^{8.74} n \log n)$ which also use negation and XOR.

As a byproduct of our construction we solve the following combinatorial problem. It is well known that for fixed k and any n there exists a collection of $r = O(\log n)$ functions, $f_1, \cdots, f_r$, from $\{1, \cdots, n\}$ to $\{1, \cdots, k\}$, such that any subset of $\{1, \cdots, n\}$ of order k is mapped $1-1$ to $\{1, \cdots, k\}$ by at least one f_i. Previous proofs are nonconstructive. We give a construction for such a collection of functions.

In § 1 we give a simple construction for $\mathrm{Th}_k(x_1, \cdots, x_n)$ which, for fixed k, is of size $O(n \log n)$ and requires polynomial time in n. The construction makes use of certain sets, which, to coding theorists, are certain types of error-correcting codes. In § 2 we explain this more thoroughly, and derive more general results using these codes, including the $O(k^{12.6} n \log n)$ existence theorem. The rest of the paper is devoted to quicker constructions of these codes, using well-known coding techniques. Section 3 gives a randomized algorithm for construction, simplified by using linear codes. Section 4 uses the idea of concatenating codes and Reed–Solomon codes, which greatly improves the construction at the cost of lengthening the code somewhat. Section 5 uses Justesen codes to give explicit codes; these codes are much longer than those of §§ 3 and 4 (giving threshold function formulae of size $c \cdot n \log n$ for fixed k with a much larger constant c).

* Received by the editors February 27, 1984, and in revised form March 18, 1985.

† Harvard University, Cambridge, Massachusetts 02138. Present address, IBM, San Jose, California 95193.

1. The basic construction. We construct formulae of the form

$$\mathrm{Th}_k(x_1, \cdots, x_n) = \bigvee_{i=1}^{r} F_i \tag{1.1}$$

where each F_j is of the form

$$\left(\bigvee_{i \in A_1^j} x_i\right) \wedge \left(\bigvee_{i \in A_2^j} x_i\right) \wedge \cdots \wedge \left(\bigvee_{i \in A_k^j} x_i\right)$$

where $A_1^j, \cdots, A_k^j$ are disjoint subsets of $\{1, \cdots, n\}$. We shall call each $A_1^j, \cdots, A_k^j$ a *partition* of $\{1, \cdots, n\}$, though we do *not* require $A_1^j \cup \cdots \cup A_k^j$ to be *all* of $\{1, \cdots, n\}$. The formula (1.1) is valid iff the r partitions have the property that each subset $\{i_1, \cdots, i_k\}$ of $\{1, \cdots, n\}$ is *separated* by at least one of the partitions, i.e. for some j the sets $A_1^j, \cdots, A_k^j$ each contain exactly one element of $\{i_1, \cdots, i_k\}$. Such a collection of partitions will be called an *n, k scheme of size r.* The formula (1.1) has length $\leqq rn$. In what follows we construct *n, k* schemes of size $O(\log n)$ for fixed *k.*

For $k = 2$ and any m we construct a $2^m, 2$ scheme of size m as follows. For positive integers i and j let $\mathrm{Bin}_j(i)$ be the jth digit (from the right) of the binary representation of i (i.e. $i = \mathrm{Bin}_1(i) + 2 \cdot \mathrm{Bin}_2(i) + 4 \cdot \mathrm{Bin}_3(i) + \cdots$). For $j = 1, \cdots, m$ we define the partition of $\{1, \cdots, 2^m\}$

$$A_0^j = \{i\colon \mathrm{Bin}_j(i) = 0, 1 \leqq i \leqq 2^m\},$$

$$A_1^j = \{i\colon \mathrm{Bin}_j(i) = 1, 1 \leqq i \leqq 2^m\}.$$

Since any two distinct integers $\{1, \cdots, 2^m\}$ differ at the jth digit of their binary representations for some j, $1 \leqq j \leqq m$, the above collection of partitions separates $\{1, \cdots, 2^m\}$. Thus they define a $2^m, 2$ scheme of size m.

This method does not directly generalize to $k > 2$. For example, one might try to use base 3 representations to construct $3^m, 3$ schemes of size m. Unfortunately there are triples such as (222, 122, 221) of distinct base 3 integers with no digit on which they *all* differ. Our idea is to consider, for some base b, a subset S of $\{1, \cdots, b^m\}$ such that any two distinct elements of S, expressed in base b, differ on more than $\frac{2}{3}$ of the (first) m digits. For such S, any triple (x, y, z) of distinct elements of S all differ on at least one of the first m digits, because each of the three pairs (x, y), (y, z), and (x, z) coincide on less than $\frac{1}{3}$ of these digits. Partitioning by digits as before we get a $|S|, 3$ scheme of size linear in m (where $|S|$ denotes the cardinality of S). If b is chosen large enough we can find an S of size exponential in m.

On $\{1, \cdots, b\}^m$, the set of m-tuples of integers from 1 to b, we define the "Hamming distance,"

$$\rho((x_1, \cdots, x_m), (y_1, \cdots, y_m)) \equiv |\{i\colon x_i \neq y_i\}|.$$

For $x \in \{1, \cdots, b\}^m$ we define the ball of radius r about x,

$$B_r(x) \equiv \{y\colon \rho(x, y) \leqq r\}.$$

For positive integer r clearly

$$|B_r(x)| \leqq \binom{m}{r} b^r \quad \text{where} \quad \binom{m}{r} = \frac{m!}{r!(m-r)!}.$$

For a subset $S \subset \{1, \cdots, b\}^m$ we define its *minimum distance* to be $\min\{\rho(x, y)\colon x, y \in S, x \neq y\}$.

LEMMA 1.1. *Let l be an integer >1. Then for $b=2^{2l}$, $c=2l$ we have that for any positive integer m there exists a subset S of $\{1,\cdots,b\}^{mc}$ with $|S|=b^m$ and with minimum distance $>(1-1/l)mc$.*

Proof. Let $\theta=1-1/l$. For any $x\in\{1,\cdots,b\}^{mc}$ we have

$$|B_{\theta mc}(x)|\leqq\binom{mc}{\theta mc}b^{\theta mc}<2^{mc}b^{\theta mc}=b^{(\theta+\log_b 2)mc}=b^{(1-1/l+1/2l)mc}$$

$$=b^{(1-1/2l)mc}=|\{1,\cdots,b\}^{mc}|\frac{1}{b^{mc/2l}}=|\{1,\cdots,b\}^{mc}|\frac{1}{b^m}$$

since $\log_b 2=1/2l$. To construct S we choose any $x_1\in\{1,\cdots,b\}^{mc}$ as our first element and subsequently choose x_i from

$$\{1,\cdots,b\}^{mc}-B_{\theta mc}(x_1)-B_{\theta mc}(x_2)-\cdots-B_{\theta mc}(x_{i-1}).$$

Since this set has at least $|\{1,\cdots,b\}^{mc}|(1-(i-1)/b^m)$ elements, we can keep choosing x_i's for $i\leqq b^m$.

The above construction of S takes polynomial time in b^{mc}, since we can go through the set $\{1,\cdots,b\}^{mc}$ in some order to find points for S; each time we add a point x to S we "mark off" the points in $B_{\theta mc}(x)$.

THEOREM 1.2. *For fixed k we can constrict n, k schemes of size $O(\log n)$ in time polynomial in n.*

Proof. In the above lemma let $l=\binom{k}{2}$, let m be fixed, and let $S=\{x_1,\cdots,x_{b^m}\}$. Let $y_1,\cdots,y_k$ be distinct points in S. Since each of the $\binom{k}{2}$ pairs of these points differ at a fraction $>1-\binom{k}{2}^{-1}$ of their components, there must be one component at which all pairs differ, i.e. for some u the uth component of $y_1,\cdots,y_k$ are all distinct. We therefore construct a b^m, k scheme as follows: for each integer j, $1\leqq j\leqq cm$, and integers $t_1,\cdots,t_k$ with $1\leqq t_1<t_2<\cdots<t_k\leqq b$ we define partition whose uth subset is

$$A_u^{j,t_1,\cdots,t_k}\equiv\{i\text{: the } j\text{th component of } x_i \text{ is } t_u\},$$

$u=1,\cdots,k$. Since any k x_i's are separated by some jth component, the above collection of partitions separates any k points of $\{1,\cdots,b^m\}$. This scheme is of size

$$mc\binom{b}{k}=mk(k-1)\binom{2^{k(k-1)}}{k}=O(m).$$

Hence for any n we can take $m=\lceil\log_b n\rceil$ ($\lceil a\rceil$ denoting the smallest integer $\geqq a$) to get an n, k scheme of size $O(\log n)$. Furthermore the construction of S is done in polynomial time of $b^{mc}=O(n^c)$.

We can significantly improve on Lemma 1.1 by using Stirling's formula

$$n!=\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{r_n}$$

where

$$\frac{1}{12n+1}<r_n<\frac{1}{12n}.$$

Then we have

$$\binom{mc}{\theta mc}<\frac{1}{\sqrt{2\pi\theta(1-\theta)mc}}[\theta^\theta(1-\theta)^{(1-\theta)}]^{-mc}e^{1/12mc}<[\theta^\theta(1-\theta)^{(1-\theta)}]^{-mc}$$

or

$$\binom{mc}{\theta mc} < h^{mc} \quad \text{where } h = [\theta^{\theta}(1-\theta)^{(1-\theta)}]^{-1}. \tag{1.2}$$

Using this bound, instead of $\binom{mc}{\theta mc} < 2^{mc}$ which we used before, we get

LEMMA 1.3. *Lemma* 1.1 *holds for any* c *and* $b = h^{\beta}$ *with*

$$\frac{1}{c} + \frac{1}{\beta} \leqq \frac{1}{l}.$$

Proof.

$$\frac{|B_{\theta mc}(x)|}{|\{1, \cdots, b\}^{mc}|} < \frac{\binom{mc}{\theta mc} b^{\theta mc}}{b^{mc}} < \frac{h^{mc} b^{\theta mc}}{b^{mc}} = (hb^{-1/l})^{mc}$$

$$= \frac{1}{b^m}(b^{1/c + \log_b h - 1/l})^{mc} = 1/b^m (b^{1/c + 1/\beta - 1/l})^{mc}$$

which is $\leqq 1/b^m$ if $1/c + 1/\beta - 1/l \leqq 0$.

COROLLARY 1.4. *Lemma* 1.1 *holds with* $c = 2l$ *and any* $b \geqq e^2 l^2 = (7.389 \cdots) l^2$ *or* $c = l^2$ *and* $b \geqq 8l$.

Proof. For $c = 2l$ we need $\beta \geqq 2l$ and thus

$$b \geqq h^{2l} = \left[l^{1/l} \left(\frac{l}{l-1} \right)^{(l-1)/l} \right]^{2l} = l^2 \left[\left(1 + \frac{1}{l-1} \right)^{l-1} \right]^2 < l^2 e^2$$

since $(1 + 1/n)^n < e$, for all integers n. Also, using $(1 + 1/n)^{n+1} < 4$ and $n^{1/(n-1)} \leqq 2$ for all n positive integer, we see that for $c = l^2$ it suffices to choose

$$b \geqq h^{l^2/(l-1)} = l^{l/(l-1)} \left(1 + \frac{1}{l-1} \right)^{l} = l l^{1/(l-1)} \left(1 + \frac{1}{l-1} \right)^{l} < l \cdot 2 \cdot 4 = 8l.$$

Thus in Theorem 1.2 we can take $c = k(k-1)$ and $b = 2k^2(k-1)^2$ or $c = \frac{1}{4}k^2(k-1)^2$ and $b = 4k(k-1)$.

Remark. R. Boppana has pointed out to me that this corollary holds with $c = O(l \log l)$, $b = O(l)$.

2. The general theory. In this section we generalize on the basic construction of Theorem 1.2 and describe how the theory of error-correcting codes ties into our construction.

By a b, m, c, θ *set* we shall mean a subset S of $\{1, \cdots, b\}^{mc}$ of minimal distance $> \theta mc$ and with $|S| = b^m$. In §§ 2 and 3 we will always take m to be an integer. Of course, b, cm, and b^m must always be integers.

Let us examine the role played by the b, m, c, θ set, $S = \{x_1, \cdots, x_{b^m}\}$, in the proof of Theorem 1.2. We constructed our b^m, k scheme by separating $x_1, \cdots, x_{b^m}$ by their components. For each j, $1 \leqq j \leqq cm$ we associated $\binom{b}{k}$ partitions with the jth component; we point out that the jth component of k points in S represent k points of $\{1, \cdots, b\}$, and so the $\binom{b}{k}$ partitions associated with the jth component simply represent a b, k scheme. We observe that a better b, k scheme, of size r, would lead to a construction of a b^m, k scheme of size $cm \cdot r$.

THEOREM 2.1. *Let* $\{\mathscr{A}_1^j, \cdots, \mathscr{A}_k^j\}_{j=1,\cdots,r}$ *be a* b, k *scheme of size* r. *Then from this scheme, and from a* b, m, c, θ *set* $S = \{x_1, \cdots, x_{b^m}\}$ *with* $\theta = 1 - \binom{k}{2}^{-1}$, *we can construct a* b^m, k *scheme of size* $cm \cdot r$.

Proof. For integers j_1, j_2 with $1 \leqq j_1 \leqq mc$ and $1 \leqq j_2 \leqq r$ we define the partition whose uth subset is

$$A_u^{j_1,j_2} \equiv \{i\text{: the } j_1\text{th component of } x_i \text{ is contained in } \mathscr{A}_u^{j_2}\}.$$

Clearly these $mc \cdot r$ partitions separate any k distinct points of $\{1, \cdots, b^m\}$ and thus form a b^m, k scheme of size $cm \cdot r$.

Theorem 2.1 tells us that the role of the b, m, c, θ set was to combine with a b, k scheme to form a b^m, k scheme. Next we show how to combine a b, m, c, θ set directly with a formula for $\mathrm{Th}_k(y_1, \cdots, y_b)$ to get a formula for $\mathrm{Th}_k(y_1, \cdots, y_{b^m})$:

THEOREM 2.2. *Let there exist a (monotone) formula for* $\mathrm{Th}_k(z_1, \cdots, z_b)$ *of size* τ *(i.e. total number of occurrences of variables). From this formula, and from a* b, m, c, θ *set* $S = \{x_1, \cdots, x_{b^m}\}$ *with* $\theta = 1 - \binom{k}{2}^{-1}$, *we can construct a (monotone) formula for* $\mathrm{Th}_k(y_1, \cdots, y_{b^m})$ *of size* τcmb^{m-1}.

Proof. For each j, $1 \leqq j \leqq cm$ we define the partition of $\{1, \cdots, b^m\}$ into b subsets, $A_1^j, \cdots, A_b^j$, where

$$A_u^j \equiv \{i\text{: the } j\text{th component of } x_i \text{ is } u\}.$$

Then

$$\mathrm{Th}_k(y_1, \cdots, y_{b^m}) = \bigvee_{j=1}^{cm} \mathrm{Th}_k\left(\bigvee_{i \in A_1^j} y_i, \bigvee_{i \in A_2^j} y_i, \cdots, \bigvee_{i \in A_b^j} y_i\right) \tag{2.1}$$

because a subset R of $\{1, \cdots, b^m\}$ satisfies $|R| \geqq k$ iff there is some j for which R has members in at least k of $A_1^j, \cdots, A_b^j$. On the right-hand side of (2.1) we have a disjunction of cm threshold-k functions of b arguments. Consider one of these functions

$$\mathrm{Th}_k\left(\bigvee_{i \in A_1^j} y_i, \bigvee_{i \in A_2^j} y_i, \cdots, \bigvee_{i \in A_b^j} y_i\right) \tag{2.2}$$

for some fixed j. The b arguments of this threshold-k function contain a total of b^m occurrences of variables (in fact, each variable $y_1, \cdots, y_{b^m}$ occurs once). If, in our formula for $\mathrm{Th}_k(z_1, \cdots, z_b)$, each variable z_i appears the same number of times, i.e. τ/b times, then the formula constructed for (2.2) will be of size $\tau/b \cdot b^m$. If some of the variables $z_1, \cdots, z_b$ occur less often than others, then by matching these ones up with the larger sets among $A_1^j, \cdots, A_b^j$ we will do no worse than $\tau/b \cdot b^m$. Thus using (2.1) we can construct a (monotone) formula for $\mathrm{Th}_k(y_1, \cdots, y_{b^m})$ of size $cm \cdot \tau/b \cdot b^m$.

The construction of b, m, c, θ sets is a central problem of the theory of error-correcting codes. In error-correcting codes one assumes that the characters of a transmitted message each have a certain probability of being received incorrectly; if we only transmit from a subset, S, of all possible messages, having minimum distance $\geqq 2t+1$, then we can correct the errors in the received message as long as no more than t characters are in error. Corollary 1.4 tells us that for $\theta = 1 - 1/l$ there are $8l, m, l^2$, sets for integers m, constructable in time polynomial in $(8l)^{ml^2}$. Actually, there are more sophisticated error-correcting codes which, for fixed θ, yield explicit b, m, c, θ sets for all m and some fixed b and c. We will devote §§ 3-5 to explaining some of these techniques. Before doing so we shall discuss the implications of Theorems 2.1 and 2.2 and discuss Lemma 1.3 in coding theory terms.

According to Corollary 1.4, for $\theta(k) = 1 - \binom{k}{2}^{-1}$ we can construct b, m, c, θ sets for any m and for $b(k) = 4k(k-1)$, $c(k) = \frac{1}{4}k^2(k-1)^2$. By Theorem 2.1, if we can construct a $b(k), k$ scheme of size $r(k)$ then we can constrict b^m, k schemes of size $cmr(k) = O(k^4 mr(k))$. Thus constructing $\mathrm{Th}_k(x_1, \cdots, x_n)$ formulae of size $n \log n$ times a

polynomial in k would follow from a construction of $4k^2$, k schemes of polynomial size in k. Unfortunately our methods do not improve on previously known bounds for $r(k)$, which are exponential in k. However, recently L. G. Valiant has demonstrated polynomial bounds for the size of monotone formulae for $\mathrm{Th}_k(x_1, \cdots, x_{4k^2})$.

LEMMA 2.3. *There exist monotone Boolean formulae for* $\mathrm{Th}_k(x_1, \cdots, x_n)$ *for any* k *of size* $O(n^{5.3})$.

Proof. See [V].

THEOREM 2.4. *There exist monotone Boolean formulae for* $\mathrm{Th}_k(x_1, \cdots, x_n)$ of size $O(k^{12.6} n \log n)$.

Proof. By Lemma 2.3 there exist monotone formulae for $\mathrm{Th}_k(x_1, \cdots, x_{4k^2})$ of size $O(k^{10.6})$. Combining this with Theorem 1.2 in which we take $b = 4k^2$, $c = k^4$ yields formulae for $\mathrm{Th}_k(x_1, \cdots, x_{b^m})$ of size

$$cm\frac{O(k^{10.6})}{b}b^m = O(k^{12.6} m b^m).$$

For any n set $m = \lceil \log n \rceil / \log b$ and construct $\mathrm{Th}_k(y_1, \cdots, y_n)$ out of our $\mathrm{Th}_k(x_1, \cdots, x_{b^m})$ formula by choosing the n variables of $x_1, \cdots, x_{b^m}$ which occur with the least frequency in our formula (and setting the others to 0). This yields a formula for $\mathrm{Th}_k(y_1, \cdots, y_n)$ no more than n/b^m times the size of our $\mathrm{Th}_k(x_1, \cdots, x_{b^m})$ formula, i.e. a formula for $\mathrm{Th}_k(y_1, \cdots, y_n)$ of size

$$\frac{n}{b^m}O(k^{12.6} m b^m) = O(k^{12.6} m n) = O(k^{12.6} n \log n).$$

Since Valiant's proof of Lemma 2.3 is nonconstructive, so is Theorem 2.4. Works of Pippenger [Pi], Paterson [Pa], and Peterson [Pe] have led to constructions of formulae for $\mathrm{Th}_k(x_1, \cdots, x_n)$ of size $O(n^{3.37})$ (for all k) in which negation and XOR are allowed as well. Also, a result of Ajtai, Komlos, and Szemeredi [A, K, S] on comparator sorting networks of depth $O(\log n)$ implies a construction for monotone formulae for $\mathrm{Th}_k(x_1, \cdots, x_n)$ (for all k) of size polynomial in n (but the degree of the polynomial is very large). Using Theorem 2.2, these results yield constructions for formulae for $\mathrm{Th}_k(x_1, \cdots, x_n)$ of size $O(k^{8.74} n \log n)$ using negation, XOR, conjunction, and disjunction, and monotone formulae of size $n \log n \cdot$ polynomial (of large degree) of k.

R. Boppana and a referee have pointed out that Lemma 2.3 can be improved to $O(k^{4.3} n \log n)$; along with the improvement of Corollary 1.4, this improves Theorem 2.4 to $O(k^{4.3} n \log n)$.

Let us now return to the construction of b, m, c, θ sets. Lemmas 1.1 and 1.3 are consequences of what coding theorists call the Gilbert (or Gilbert–Varshamov) bound, which simply says that b, m, c, θ sets exist if

$$\frac{|B_{\theta mc}(x)|}{|\{1, \cdots, b\}^{mc}|} \leqq \frac{1}{b^m} \qquad (x \in \{1, \cdots, b\}^{mc}). \tag{2.3}$$

In Lemma 1.3 we estimated $|B_{\theta mc}(x)|$ by $\binom{mc}{\theta mc} b^{\theta mc}$. For fixed b, c, and θ the following bound is better for large m:

$$|B_{\theta mc}(x)| \leqq \sum_{i=0}^{\theta mc} \binom{mc}{i} (b-1)^i.$$

This leads to the asymptotic Gilbert bound, which, in our terminology, says that for fixed b, c, and θ, bound (2.3) is achieved for sufficiently large m if b, c, and θ satisfy

$\theta < 1 - 1/b$ and

$$c > \frac{1}{1 - H_b(\theta)},$$

where $H_b(\theta)$ is given by

$$H_b(\theta) \equiv \theta \log_b (b-1) - \theta \log_b \theta - (1-\theta) \log_b (1-\theta).$$

See [vL] for details. This bound is slightly better than that of Lemma 1.3.

In §§ 3 and 4 we improve on the b, m, c, θ set construction of Lemma 1.1, which took time polynomial in b^{mc}. The constructions which follow take time polynomial in b^m where the degree of the polynomial is independent of c (and b and θ). In § 3 we give a simple random algorithm working in expected time polynomial in b^m. In § 4 we use more sophisticated techniques to give an algorithm without randomness. In § 5 we give an explicit construction of such set in terms of finite fields, but for $\theta = 1 + 1/l$ the c we choose is exponential in l.

3. A randomized linear construction. In Theorem 1.2 we constructed n, k schemes of size $O(\log n)$ in time polynomial in $n^{k(k-1)}$. In what follows we give a *random* algorithm which will construct a b, m, c, θ set, S, in space $O(m^2)$ bits and *expected* time $O(mb^m)$ with time measured as number of bit operations. We require that b be prime and that b, m, c, θ satisfy (2.3). The construction is known as a random linear code (see [vL]).

Let b, m, c, θ satisfy (2.3). Our first attempt at a random algorithm to construct S would be to pick $x_1, \cdots, x_{b^m}$ at random and to see if $\rho(x_i, x_j) > \theta mc$ for all i, j, $i \neq j$. We would have $\frac{1}{2}b^m(b^m - 1)$ pairs x_i, x_j to check, and each check would require $O(m)$ bit operations since the x_i's are of length $O(m)$ bits. Thus this check would require $\Omega(mb^{2m})$ bit operations, as well as $\Omega(mb^m)$ bits of storage for the x_i's. So even if our first guess is correct we need a fair amount of time and a lot of storage. To improve on this we use the idea of linear codes.

Let us further assume that b is a prime number, which we will henceforth denote p. Then $\mathbb{Z}/p\mathbb{Z}$, the integers modulus p, form a field and $(\mathbb{Z}/p\mathbb{Z})^{mc}$ forms a vector space over $\mathbb{Z}/p\mathbb{Z}$. We can identify $(\mathbb{Z}/p\mathbb{Z})^{mc}$ with $\{1, \cdots, p\}^{mc}$, and the notions of Hamming distance and minimal distance of a subset carry over to $(\mathbb{Z}/p\mathbb{Z})^{mc}$. Note that for vectors $u, v \in (\mathbb{Z}/p\mathbb{Z})^{mc}$ and scalar $\alpha \in \mathbb{Z}/p\mathbb{Z}$

$$\rho(u, v) = \rho(u - v, 0)$$

and

$$\rho(u, v) = \rho(\alpha u, \alpha v) \quad \text{if } \alpha \neq 0.$$

We define the *weight* of a vector, v, to be

$$w(v) \equiv \rho(v, 0).$$

Let T be any linear subspace of $(\mathbb{Z}/p\mathbb{Z})^{mc}$. Then the minimum distance of T,

$$\begin{aligned} \min \{\rho(u, v) \colon u, v \in T, u \neq v\} &= \min \{\rho(u - v, 0) \colon u, v \in T, u \neq v\} \\ &= \min \{w(x) \colon x \in T, x \neq 0\}. \end{aligned}$$

Hence to find the minimum distance of a linear subspace we need only find the minimum weight, which involves $|T|$ weight checks as opposed to $\frac{1}{2}|T|(|T| - 1)$ distance checks. It turns out that (2.3) is enough to guarantee the existence of a b, m, c, θ set which is also a linear subspace of $(\mathbb{Z}/p\mathbb{Z})^{mc}$.

PROPOSITION 3.1. *Let T be a subspace of $(\mathbb{Z}/p\mathbb{Z})^{mc}$ of minimum distance $\geq d$ and let v be a vector whose distance to T,*

$$\rho(v, T) \equiv \min_{t \in T} \rho(v, t) \geq d.$$

Then the subspace spanned by T and v also has minimum distance $\geq d$.

Proof. Any vector in the span of T and v can be written as $t + \alpha v$ with $t \in T$ and α scalar. If $\alpha = 0$ then $w(t+\alpha v) = w(t) \geq d$, while if $\alpha \neq 0$ then $w(t+\alpha v) = w(\alpha^{-1}(t+\alpha v)) = w(\alpha^{-1}t + v) \geq d$.

THEOREM 3.2. *We can construct an S with minimum distance $> \theta mc$ and $|S| = p^m$ via a random algorithm in space $O(m^2)$ bits and in expected time $O(mp^m)$ bit operations.*

Proof. Choose $v_1, \cdots, v_m$ at random from $(\mathbb{Z}/p\mathbb{Z})^{mc}$ and consider $S = \text{span}(v_1, \cdots, v_m)$. By Proposition 3.1, if $\rho(v_1, 0) \geq d$ and for each $i > 1$, $\rho(v_i, \text{span}(v_1, \cdots, v_{i-1})) \geq d$, then an easy induction argument shows span $(v_1, \cdots, v_m)$ has minimum distance $\geq d$. Since the probability that $\rho(v_i, \text{span}(v_1, \cdots, v_{i-1})) > \theta mc$ is

$$\geq 1 - \frac{p^{i-1} B_{\theta mc}(0)}{p^{mc}} = 1 - \frac{p^{i-1}}{p^m},$$

we have that S has minimum distance $> \theta mc$ with probability

$$\geq \left(1 - \frac{1}{p^m}\right)\left(1 - \frac{p}{p^m}\right) \cdots \left(1 - \frac{p^{m-1}}{p^m}\right).$$

Since $\prod_{i=1}^m (1 - a_i) \geq 1 - \sum_{i=1}^m a_i$ if $0 \leq a_i \leq 1$ (as shown by induction starting with $(1-a_1)(1-a_2) = 1 - a_1 - a_2 + a_1 a_2 \geq 1 - a_1 - a_2$), the above probability is

$$\geq 1 - \frac{1}{p^m} - \frac{p}{p^m} - \cdots - \frac{p^{m-1}}{p^m} = 1 - \frac{1}{p^m}(1 + p + \cdots + p^{m-1})$$

$$= 1 - \frac{1}{p^m}\left(\frac{p^m - 1}{p - 1}\right) > 1 - \frac{1}{p-1} = \frac{p-2}{p-1}.$$

Hence the expected number of tries to construct S with minimum distance $> \theta mc$ is $(p-1)/(p-2)$.

To find the minimum distance of S we need to check the weights of $\alpha_1 v_1 + \cdots + \alpha_m v_m$ for all α_i's between 0 and $p-1$; if we go through $(\alpha_1, \cdots, \alpha_m)$ in lexicographic order, i.e. in order increasing with $\alpha_1 + \rho\alpha_2 + p^2\alpha_3 + \cdots + p^{m-1}\alpha_m$ (i.e. count from 0 to $p^m - 1$ in base p), then to update $\alpha_1 v_1 + \cdots + \alpha_m v_m$ we only need to add v_1 to our running vector sum each time, to add v_2 to our sum $1/p$ of the time, add v_3 $1/p^2$ of the time, etc. Thus we can go through all combinations $\alpha_1 v_1 + \cdots + \alpha_m v_m$ in $p^m(1 + 1/p + \cdots + 1/p^m)$ vector additions $= O(p^m)$ vector additions $= O(mp^m)$ bit operations. Also we need store only $\alpha_1, \cdots, \alpha_m$, the vectors $v_1, \cdots, v_m$, and the running sum $\alpha_1 v_1 + \cdots + \alpha_m v_m$, requiring a total of $O(m^2)$ bits of storage.

Hence the above random algorithm runs in space $O(m^2)$ and in expected time $O(mp^m)$ bit operations.

COROLLARY 3.3. *Using the above random algorithm, for fixed k we can construct monotone formulae for $\text{Th}_k(x_1, \cdots, x_n)$ of size $O(n \log n)$, explicit up to a construction of a set, S, taking space $O((\log n)^2)$ bits and expected time $O(n \log n)$ bit operations.*

4. A concatenation construction. In § 2 we demonstrated ways of constructing new $\text{Th}_k(x_1, \cdots, x_n)$ formulae and n, k schemes from ones with smaller n by combining them with a b, m, c, θ set. In this section we show a way of combining two b, m, c, θ

sets to get a new one, and apply this technique. These are known as concatenated codes (see [vL]).

From now on we no longer require m to be an integer in a b, m, c, θ set (only that b, b^m, and cm be integers). Note that for any b, m, c, θ set S, S is also a b, m, c, θ' set for $\theta' < \theta$, and any subset of S of size $b^{m'}$ is a $b, m', c \cdot m/m', \theta$ set, for any $m' < m$ with $b^{m'}$ an integer.

Let S_1 be a b_1, m_1, c_1, θ_1 set and S_2 a b_2, c_2, m_2, θ_2 set with $b_2 = |S_1| = b_1^{m_1}$. Then we can identify $\{1, \cdots, b_2\}$ with S_1 and think of each element of S_2 as a $c_2 m_2$-tuple of elements of S_1. More precisely, let the points of S_1 be ordered $x_1, \cdots, x_{b_1^{m_1}}$, and let $\iota: \{1, \cdots, b_2\} \to S_1$ be defined by $\iota(j) \equiv x_j$. For each point $y = (y_1, \cdots, y_{m_2 c_2})$ in S_2 let $\bar{\iota}(y) \equiv (\iota(y_1), \cdots, \iota(y_{m_2 c_2}))$. Then $\bar{\iota}: s_2 \to (S_1)^{m_2 c_2}$ which we can identify with $\{1, \cdots, b_1\}^{m_1 c_1 m_2 c_2}$. We define $S_1 \circ S_2$, the concatenation of S_2 with S_1 (with respect to a given ordering of S_1), as the image $\bar{\iota}(S_2)$. Note that $|S_1 \circ S_2| = |S_2|$, and further that $S_1 \circ S_2$ is a $b_1, m_1 m_2, c_1 c_2, \theta_1 \theta_2$ set.

The above construction also works for $b_2 < |S_1|$, but then

$$|S_1 \circ S_2| = |S_2| = b_2^{m_2} = b_1^{\mu m_2} \quad \text{where } \mu = \frac{\log b_2}{\log b_1},$$

and so $S_1 \circ S_2$ is a $b_1, \mu m_2, m_1/\mu \cdot c_1 c_2, \theta_1 \theta_2$ set.

The following set is known to coding theorists as a Reed-Solomon code: for any prime p and integer $d < p$ we define

$$\mathcal{S}(p, d) \equiv \{(f(0), f(1), \cdots, f(p-1)) \in (\mathbb{Z}/p\mathbb{Z})^p : f \text{ is a polynomial of degree } < d\}.$$

Since every polynomial of degree $\leqq d-1$ (over a field) has at most $d-1$ roots, $\mathcal{S}(p, d)$ is a

$$p, \quad d, \quad \frac{p}{d}, \quad \left(1 - \frac{d}{p}\right) \quad \text{set.}$$

While $\mathcal{S}(p, d)$ is simple and explicit, it only gives b, m, c, θ sets for $m < b$. However, concatenating $\mathcal{S}(p, d)$ sets with other sets is a very powerful technique.

THEOREM 4.1. *For fixed $b, c, \theta = 1 - 1/2l$, let S_M, $M = 1, 2, \cdots$, be a sequence of b, M, c, θ sets. Then for each sufficiently large M we can construct a $b, m, c', 1 - 1/l$ set for some $c' < 4b^2 c(2l+1)$, by concatenating an $\mathcal{S}(p, d)$ set with an $S_{O(\log m)}$, for primes p between b^M and b^{M+1} for some M with $b^{M+1} < m$.*

Proof. For each M choose a prime p between b^M and b^{M+1} (it is well known that there is at least one prime between a and $2a-2$ for any $a > \frac{7}{2}$). Let $d = \lfloor p/2l \rfloor$ (where $\lfloor a \rfloor$ denotes the greatest integer $\leqq a$). Then $S_{M+1} \circ \mathcal{S}(p, d)$ is a

$$b, \quad \mu d, \quad \frac{1}{\mu d}(M+1)cp, \quad \theta\left(1 - \frac{d}{p}\right) \quad \text{set}$$

where $\mu = \log_b p$. For sufficiently large M, $d = \lfloor p/2l \rfloor > p/(2l+1)$; for such M we have $\mu d > Mb^M/(2l+1)$, and so any subset $\tilde{S}_M$ of $S_{M+1} \circ \mathcal{S}(p, d)$ of appropriate size is a

$$b, \quad \frac{M}{2l+1} b^M, \quad \left(\frac{M}{2l+1} b^M\right)^{-1} (M+1)cp, \quad \theta\left(1 - \frac{d}{p}\right) \quad \text{set.}$$

And since $\theta(1 - d/p) \geqq (1 - 1/2l)^2 > 1 - 1/l$,

$$\tilde{S}_M \text{ is a } b, \quad \frac{M}{2l+1} b^M, \quad c', \quad 1 - \frac{1}{l} \quad \text{set}$$

where

$$c' = (2l+1)c\frac{M+1}{M}\frac{p}{b^M} < (2l+1)c \cdot 2 \cdot b = 2bc(2l+1).$$

For any m let M be the integer such that $(M-1)/(2l+1)b^{M-1} < m \leqq M/(2l+1)b^M$; clearly $M = O(\log m)$. Then $m = \alpha(M/(2l+1)b^M)$ with $1 \geqq \alpha > (M-1)/Mb > 1/2b$, and thus any subset of $\tilde{S}_m$ of appropriate size is a $b, m, c'', 1-1/l$ set with

$$c'' < \frac{1}{\alpha} 2bc(2l+1) < 4b^2c(2l+1).$$

For integer $k \geqq 1$ let $\log^k x$ denote the kth iteration of $\log x$,

$$\log^k x \equiv \underbrace{\log \log \cdots \log}_{k \text{ times}} x,$$

and define $\log^0 x \equiv x$.

In § 1 we saw that for $\theta = 1-1/l$ there are constants b, c polynomial in l such that b, m, c, θ sets can be constructed, for all integers m, in time polynomial in b^{mc}. Repeated application of Theorem 4.1 yields

THEOREM 4.2. *Let k be a fixed positive integer. Then for $\theta = 1-1/l$ there exists constants b, c polynomial in l such that for each sufficiently large m (larger than some polynomial in l), a b, m, c', θ set with $c' < c$ can be constructed in terms of a set, S, constructable in time polynomial in $\log^{k-1} m$, and k appropriately chosen primes $< m$.*

Note that one can find all primes $< m$ in time polynomial in m. Thus Corollary 4.3 follows.

COROLLARY 4.3. *For fixed k we can construct monotone formulae for $\mathrm{Th}_k(x_1, \cdots, x_n)$ of size $O(n \log n)$ in polynomial time in n, where the degree of the polynomial does not depend on k.*

5. An explicit construction. In the previous sections our constructions for b, m, c, θ sets have always involved, at some point, the choosing of points in $\{1, \cdots, b\}^r$ which are a large distance apart. Such points were not explicitly specified. Using Justesen codes it is possible to specify sets explicitly (for $m = 1, 2, \cdots$, for fixed b, c, θ), in terms of finite fields. We remark that irreducible polynomials needed in the realization of finite fields can be found quickly; see [R].

Here we shall briefly describe these Justesen codes, assuming a familiarity with finite fields. For a more detailed discussion and background on finite fields, see [vL] or [B].

By $GF(q)$ we mean the finite field of q elements. $GF(q^m)$ is an m-dimensional vector space over $GF(q)$; it will often be useful to view it as $(GF(q))^m$, i.e. think of its elements as m-tuples of $GF(q)$ elements.

Let us generalize the notion of a concatenated set. Let T be a $b_2, m_2, c_2, \tilde{\theta}$ set and let S_i be b_1, m_1, c_1, θ_i sets for $i = 1, \cdots, c_2 m_2$, with $b_2 = b_1^{m_1}$. Then we define the concatenation of T with $\{S_i\}$, denoted $\{S_i\} \circ T$, as $\bar{\iota}(T)$ where $\bar{\iota}: T \to S_1 \times S_2 \times \cdots \times S_{c_2 m_2}$ is given by

$$\bar{\iota}((y_1, \cdots, y_{c_2 m_2})) \equiv (\iota_1(y_1), \cdots, \iota_{c_2 m_2}(y_{c_2 m_2}))$$

where ι_i are any bijections from $\{1, \cdots, b_2\}$ to S_i. In the construction of § 4 we constructed $\{S_i\} \circ \mathcal{S}(p, d)$ by taking a large minimum distance set, S, and setting $S_1 = S_2 = \cdots = S_{c_2 m_2} = S$. Unfortunately, we do not have an explicit description of S.

However, there are explicit descriptions of collections $\mathcal{A}$ of b_1, m_1, c_1, θ sets with fixed b_1, m_1, c_1 and with almost all of the θ's close to 1. The idea behind a Justesen code is to take such collections $\mathcal{A}$ and construct $\{S_i\} \circ T$ with the S_i distinct elements of $\mathcal{A}$.

PROPOSITION 5.1. *Let T be a $b_2, m_2, c_2, \tilde{\theta}$ set, S_i be b_1, m_1, c_1, θ_i sets for $i = 1, \cdots, c_2 m_2$ with $b_2 = b_1^{m_1}$. Of the numbers $\theta_1, \cdots, \theta_{c_2 m_2}$, let no more than $\varepsilon m_2 c_2$ of them be $< \theta$ for some $\varepsilon < 1$ and θ. Then $\{S_i\} \circ T$ is a*

$$b_1, m_1 m_2, c_1 c_2, \theta(\tilde{\theta} - \varepsilon) \quad \text{set.}$$

Proof. Two distinct elements of T differ at more than $\tilde{\theta}_{m_1 c_2}$ of their components, and more than $(\tilde{\theta} - \varepsilon) m_2 c_2$ of these components correspond to S_i's with $\theta_i \geqq \theta$.

Next we demonstrate collections $\mathcal{A}$ mentioned previously.

Let c, q, and m be fixed. For any $\alpha_1, \cdots, \alpha_{c-1} \in GF(q^m)$ consider the set

$$S_{\alpha_1, \cdots, \alpha_{c-1}} = \{(\beta, \beta\alpha_1, \cdots, \beta\alpha_{c-1}) : \beta \in GF(q^m)\} \subset (GF(q^m))^c.$$

Viewing $GF(q^m)$ as $(GF(q))^m$, we view each $S_{\alpha_1, \cdots, \alpha_{c-1}}$ as a q, m, c, θ set for some θ. Each element of $(GF(q))^{mc}$ of nonzero weight can belong to at most one $S_{\alpha_1, \cdots, \alpha_{c-1}}$. Thus, of the $q^{m(c-1)}$ sets $S_{\alpha_1, \cdots, \alpha_{c-1}}$, at most $B_{\theta mc}(0)$ have a nonzero element of weight $\leqq \theta mc$. The proof of Lemma 1.3 also proves

PROPOSITION 5.2. *Let $q = h^\beta$ where $h = [\theta^\theta (1-\theta)^{(1-\theta)}]^{-1}$, and let*

$$\frac{2}{c} + \frac{1}{\beta} \leqq \frac{1}{l}$$

where $1/l = 1 - \theta$. Then $|B_{\theta mc}(0)| < q^{m(c-2)}$.

COROLLARY 5.3. *Let q, c, and θ be as in Proposition 5.2. Then less than $1/q^m$ of the $S_{\alpha_1, \cdots, \alpha_{c-1}}$'s are not q, m, c, θ sets.*

Note that since Proposition 5.2 requires $2/c + 1/\beta \leqq 1/l$ instead of $1/c + 1/\beta \leqq 1/l$ as in Lemma 1.3, Corollary 1.4 shows that $c = 4l$ and $q \geqq e^2 l^2$ or $c = 2l^2$ and $q \geqq 8l$ will work in Proposition 5.2.

Next we describe the set T which we concatenate with $\{S_{\alpha_1, \cdots, \alpha_{c-1}}\}$. T will be taken to be a "*BCH* code". We shall give a brief account of the theory of *BCH* codes; for a more detailed account, see [vL, Chap. 6] and [B, Chap. 12], for example, as indicated below.

For a prime power q and positive integer n, consider the ring $\mathcal{R} \equiv GF(q)[x]/(x^n - 1)$, the polynomials over $GF(q)$ modulus $x^n - 1$. Each element in $\mathcal{R}$, $\beta_0 + \beta_1 x + \cdots + \beta_{n-1} x^{n-1}$ can be thought of as the n-tuple $(\beta_0, \cdots, \beta_{n-1}) \in (GF(q))^n$. Let $I \subset \mathcal{R}$ be an ideal. The monic polynomial of least degree in I, $g(x)$, generates I, i.e. $I = \mathcal{R}g$; $g(x)$ is called the generator of I. Furthermore, as a subset of $(GF(q))^n$, $(\beta_0, \cdots, \beta_{n-1}) \in I$ implies any cyclic shift of $(\beta_0, \cdots, \beta_{n-1})$, i.e. $(\beta_{n-1}\beta_0\beta_1 \cdots \beta_{n-2})$, $(\beta_{n-2}\beta_{n-1}\beta_0 \cdots \beta_{n-3})$, etc., is again $\in I$. I is called a cyclic code. Also, then, $h(x)g(x) = x^n - 1$ for some $h(x) \in \mathcal{R}$ called the check polynomial. Then

$$|I| = q^{\text{degree}(h)} = q^{n - \text{degree}(g)}.$$

Let us further assume q and n are relatively prime. Let c be the multiplicative order of q in n, i.e. the smallest positive number c with $q^c \equiv 1 \pmod{n}$. Then there exist primitive nth roots of units in $GF(q^c)$; let β be one of them. For any $d < n$ let

$$BCH(q, n, d) \equiv \{a(x) \in \mathcal{R} : a(\beta) = a(\beta^2) = \cdots = a(\beta^{d-1}) = 0\}.$$

$BCH(q, n, d)$ is an ideal.

LEMMA 5.4. *$BCH(q, n, d)$ has minimum distance $\geqq d$.*

Proof. $BCH(q, n, d)$ is a linear subspace of $(GF(q))^n$. If $a(x) \in BCH(n, q, d)$ and has weight $<d$, i.e. $a(x) = a_1 x^{i_1} + \cdots + a_{d-1} x^{i_{d-1}}$, then $(a_1, \cdots, a_{d-1})$ would be in the kernel of the matrix

$$\begin{bmatrix} \beta^{i_1} & \beta^{i_2} & \cdots & \beta^{i_{d-1}} \\ \beta^{2i_1} & & & \\ \vdots & & & \vdots \\ \beta^{(d-1)i_1} & & \cdots & \beta^{(d-1)i_{d-1}} \end{bmatrix}.$$

But the determinant of the above matrix is a Vandermonde determinant equal to

$$\beta^{i_1+\cdots+i_d} \prod_{r>s} (\beta^{i_r} - \beta^{i_s})$$

which is nonzero since β is a primitive nth root of unity.

For details of the above, see [vL, Chap. 6, § 6.3]. There, a quick method for constructing $BCH(q, n, d)$ is discussed, using calculations only in $GF(q)$.

Now consider the case $n = q^c - 1$ (note that the multiplicative order of q in $q^c - 1$ is c). BCH codes with such n are called primitive. For β, primitive nth root of unity, $\in GF(q^c)$, we have

$$x^n - 1 = \prod_{i=0}^{n-1} (x - \beta^i).$$

Let d be an integer $<n$, let $g(x)$ be a generator of $BCH(q, n, d)$, and let $h(x)$ be the check polynomial. $g(x)$ is the minimal monic polynomial with roots $\beta, \beta^2, \cdots, \beta^{d-1}$. Since $g(x)h(x) = x^n - 1$, each $\alpha \in GF(q^c)$ is a root of either $g(x)$ or $h(x)$. But $\alpha \in GF(q^c)$ is a root of a polynomial $a(x) \in \mathcal{R}$ iff all of its conjugates, $\alpha^q, \alpha^{q^2}, \cdots$, are roots of $a(x)$ as well. Hence degree (g) is the number of integers i, $0 \leqq i < n$, with $i \equiv jq^k \pmod{n}$ for some k and j with $0 \leqq j \leqq d-1$.

For details of the above see [B, Chap. 12].

LEMMA 5.5. *Let $d = (q-a)a^{c-1}$ for some integer a. Then $|BCH(q, n, d)| = q^{a^c}$. Also this BCH has minimum distance $>d$.*

Proof. For any integer i define $[i]$ by $i \equiv [i] \pmod{n}$, $0 \leqq [i] < n$. If $[i]$ has base q representation $i_1 i_2 \cdots i_c$ $(q = i_1 + qi_2 + \cdots + q^{c-1} i_c)$ then $[q \cdot i]$ has representation $i_c i_1 i_2 \cdots i_{c-1}$, and in general $[q^k i]$ has representation equal to a cyclic shift of $i_1 \cdots i_c$. For any integer j, $0 \leqq j < n$, we have $0 \leqq j \leqq d-1$ iff j has representation $j_1 \cdots j_c$ with $0 \leqq j_c < q - a$. It follows that β^i, $0 \leqq i < n$, is a root of $h(x)$, not $g(x)$, iff i has representation $i_1 \cdots i_c$ with each $i_k \geqq q - a$. There are a^c such i. Thus $|BCH(n, q, d)| = q^{a^c}$. In addition, the smallest such i has representation $(q-a) \cdots (q-a)$, and thus β^j is a root of $g(x)$ for any $0 \leqq j < (q-a)(q^{c-1} + q^{c-2} + \cdots + 1)$; the proof of Lemma 5.4 then shows the minimum distance $\geqq (q-a)(q^{c-1} + \cdots + 1) > d$.

If in Lemma 5.5 we replace q by q^m and a by aq^{m-1}, then we see that the BCH set has size $(q^m)^{a^c q^{mc-c}}$ and minimum distance $>(q-a)q^{mc}$. Hence we have

COROLLARY 5.6. *$BCH(q^m, q^{mc} - 1, (q-a)q^{mc-1})$ is a*

$$q^m, \quad q^{mc}\left(\frac{a}{q}\right)^c, \quad \left(\frac{q}{a}\right)^c \frac{q^{mc}-1}{q^{mc}}, \quad 1 - \frac{a}{q} \quad \textit{set.}$$

Returning to our construction, let $\theta = 1 - 1/l$, $c = 4l$, and q be any prime power $\geqq e^2 l^2$ and, say, $\leqq 2e^2 l^2 < 15l^2$. Let $a = \lfloor (1/l)q \rfloor$. Then $a > (1/l)q - 1 = q(1/l - 1/q) > q/(l+1)$. For any positive integer m, let T be the BCH set in Corollary 5.6 with c

replaced by $c-1$. Since $\{S_{\alpha_1,\cdots,\alpha_{c-1}}\}$ has at least $1-1/q^m$ of its elements with minimum distance $>\theta mc$, so does $\mathscr{A}\equiv\{S_{\alpha_1,\cdots,\alpha_{c-1}}\}-S_{0,0,\cdots,0}$ ($S_{0,0,\cdots,0}$ has minimum distance 1). Note $\mathscr{A}$ consists of $q^{m(c-1)}-1$ sets. Consider $\mathscr{A}\circ T$. By Proposition 5.1 it is a

$$q,\quad mq^{m(c-1)}\left(\frac{a}{q}\right)^{c-1},\quad c\left(\frac{q}{a}\right)^{c-1}\frac{q^{m(c-1)}-1}{q^{m(c-1)}},\quad \theta\left(1-\frac{a}{q}-\frac{1}{q^m}\right)\quad \text{set.}$$

To simplify these terms, note $1/(l+1)<a/q<1/l$, and so $\theta(1-a/q-1/q^m)>(1-1/l)(1-2/l)>1-3/l$. Next since $(a/q)^{c-1}>(1/(l+1))^{c-1}$, any subset of $\mathscr{A}\circ T$ of appropriate size is a

$$q,\quad mq^{m(c-1)}\left(\frac{1}{l+1}\right)^{c-1},\quad c(l+1)^{c-1}\frac{q^{m(c-1)}-1}{q^{m(c-1)}},\quad 1-\frac{3}{l}\quad \text{set,}$$

that is, a

$$q,\quad mq^{m(4l-1)}\left(\frac{1}{l+1}\right)^{4l-1},\quad 4l(l+1)^{4l-1}\left(1-\frac{1}{q^{m(4l-1)}}\right),\quad 1-\frac{3}{l}\quad \text{set} \tag{5.1}$$

or a $q, f(m), g(m), 1-3/l$ set with $f(m)$ and $g(m)$ according to (5.1). Note $g(m)<4l(l+1)^{4l-1}$. Thus to find a $q, n, c', 1-3/l$ set for any n, we choose m with $f(m)\leqq n<f(m+1)$, and since

$$\frac{f(m+1)}{f(m)}<\frac{m+1}{m}q^{4l-1}<2(15l^2)^{4l-1},$$

the above yields a $q, n, c', 1-3/l$ set with

$$c'<g(m)\cdot 2(15l^2)^{4l-1}<r(l)$$

with $r(l)\equiv 4l(l+1)^{4l-1}\cdot 2(15l^2)^{4l-1}$. Thus we have

THEOREM 5.7. *The above construction gives for each integer n a $q, n, c', 1-1/l$ set for fixed $q\leqq 15l^2$ and some $c'\leqq r(l)$ defined as above.*

We finish by remarking that one can modify the construction to avoid the construction of irreducible polynomials needed for realizing $GF(p^m)$, $m>1$. We do this as follows: First note that Proposition 5.1 holds with $b_2\leqq b_1^{m_1}$ and $\{S_i\}\circ T$ then being a

$$b_1,\quad \mu m_2,\quad \frac{m_1}{\mu}c_1c_2,\quad \theta(\theta_2-\varepsilon)\quad \text{set},\qquad \mu=\frac{\log b_2}{\log b_1}.$$

As before, set $\theta=1-1/l$, $c=4l$, but now take q to be any integer $\geqq e^2l^2$. For each m find the largest prime, p, $\leqq q^m$. Let $T=BCH(p, p^{c-1}-1, d)$ with $d=(p-\lfloor p/l\rfloor)p^{c-2}$. For $\alpha_1,\cdots,\alpha_{c-1}$ in $\mathbb{Z}/p\mathbb{Z}$ let

$$S_{\alpha_1,\cdots,\alpha_{v-1}}\equiv\{(a,\alpha_1 a,\cdots,\alpha_{c-1}a)\in(\mathbb{Z}/p\mathbb{Z})^c\colon a\in\mathbb{Z}/p\mathbb{Z}\}$$

and let $\mathscr{A}=\{S_{\alpha_1,\cdots,\alpha_{c-1}}\}-S_{0,0,\cdots,0}$. Let $u=\{1,\cdots,q\}^m$. Since $<q^{m(c-2)}$ points in U^c are of weight $\leqq\theta mc$, less than a fraction $q^{m(c-2)}/p^{c-1}=(a^m/p)^{c-2}\cdot 1/p$ of the sets $U\circ S_{\alpha_1,\cdots,\alpha_{c-1}}$ have minimum distance $\leqq\theta mc$. Then, proceeding as in Theorems 5.7 and 4.1, the sets $U\circ\mathscr{A}\circ T$ will work for the construction (one $U\circ\mathscr{A}\circ T$ for each positive integer m). As remarked before, according to [vL, § 6.3], $BCH(p, n, d)$ can be quickly constructed using calculations only in $GF(p)$; one need not construct $GF(p^m)$, $m>1$.

Acknowledgments. The $\mathrm{Th}_k(x_1,\cdots,x_n)$ construction problem was posed to the author by L. G. Valiant; the author would like to thank him, as well as P. Elias, R. Boppana, and M. O. Rabin, for several helpful conversations and suggestions.

REFERENCES

[A, K, S] M. AJTAI, J. KOMLOS AND E. SZEMEREDI, *An $O(n \log n)$ sorting network*, Proc. 15th ACM Symposium on Theory of Computing, 1983.

[B] E. R. BERLEKAMP, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[Kh] L. S. KHASIN, *Otsenka slozhnusti realizatsii monotonnykh simmetrich eskikh funktsii formulami* $\vee$ *bazise* $\vee$, $\&$, $\neg$, Dokl. Akad. Nauk SSSR 189 (1969), pp. 752–755; transl. as *Complexity bounds for the realization of monotonic symmetrical functions by means of formulas in the basis* $\vee$, $\&$, $\neg$, Soviet Physics Dokl., 14 (1970), pp. 1149–1151.

[K, P] M. KLEIMAN AND N. PIPPENGER, *An explicit construction of short monotone formulae for the monotone symmetric functions*, Theoret. Comput. Sci., 7 (1978), pp. 325–332.

[Kr] R. E. KRICHEVSKII, *Slozhnost kontaktnykh skhem, realizuyushchikh odnu funktsiyu algebry logiki*, Dokl. Akad. Nauk. SSSR, 151 (1963), pp. 803–806; transl. as *Complexity of contact circuits realizing the function of logical algebra*, Soviet Physics Dokl., 8 (1964), pp. 770–772.

[vL] J. H. VAN LINT, *Introduction to Coding Theory*, Springer-Verlag, New York, 1982.

[Pa] M. S. PATERSON, *New bounds on formula size*, Lecture Notes in Computer Science 48, Springer-Verlag, Berlin, 1977, pp. 17–26.

[Pe] G. L. PETERSON, *An upper bound on the size of formulae for symmetric boolean functions*, Dept. Computer Science Tech. Report 78-03-01, Univ. Washington, Seattle, 1978.

[Pi] N. PIPPENGER, *Short formulae for symmetric functions*, IBM Research Report RC-5143, Yorktown Heights, NY, 1974.

[R] M. O. RABIN, *Probabilistic algorithms in finite fields*, this Journal, 9, (1980), pp. 273–280.

[V] L. G. VALIANT, *Short monotone formulae for the majority function*, J. Algorithms, to appear.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
003

THE COMPLEXITY OF COUNTING STABLE MARRIAGES*

ROBERT W. IRVING† AND PAUL LEATHER‡

Abstract. In an instance of size n of the stable marriage problem, each of n men and n women ranks the members of the opposite sex in order of preference. A stable matching is a complete matching of men and women such that no man and woman who are not partners both prefer each other to their actual partners under the matching.

It is well known that at least one stable matching exists for every stable marriage instance, so that the decision version of the problem always has a "yes" answer. Furthermore, efficient algorithms are known for the determination of such a stable matching, so that the search version of the problem is polynomially solvable.

However, by exploring the structure of the set of stable matchings for any particular instance of the problem, and exploiting its relationship with the set of antichains of an associated partially ordered set, we prove that the enumeration version of the problem—determining the number of stable matchings—is #P-complete, and therefore cannot be solved in polynomial time if $P \neq NP$.

Key words. stable marriage, computational complexity, #P-complete, partially ordered set, antichain

1. Introduction and background.

The stable marriage problem, first described and studied by Gale and Shapley [1], involves two disjoint sets of equal cardinality n, the men and the women. Each person ranks all members of the opposite sex in order of preference. A stable matching is defined as a complete matching between men and women with the property that no man and woman who are not partners both prefer each other to their actual partners under the matching. A readable introductory treatment of the problem is given by Pólya, Tarjan and Woods [7].

Gale and Shapley proved that at least one stable matching exists for all instances of the problem, and gave an efficient algorithm for finding such a matching. Later, McVitie and Wilson [5] gave an alternative formulation of the algorithm, together with an extension to find all stable matchings for a given problem instance. A quite different algorithm for all stable matchings, based on backtrack search, was given by Wirth [10], but although easier in concept, it is much less efficient than that of [5].

As a concluding chapter of a wide-ranging treatise on stable marriage, Knuth [4] presented a set of twelve unsolved problems. The sixth of these asked for a description of the structure of the set of stable matchings that would allow these matchings to be characterised without having to enumerate them. He also raised various issues concerning the number of stable matchings that may arise for particular instances of the problem. In particular, he showed that the maximum number of solutions for instances of size n (i.e. involving n men and n women) grows exponentially with n, a fact that we shall re-establish in a different way in § 2. It follows that any algorithm for finding all stable matchings is necessarily of exponential worst-case time-complexity. However, one might ask whether at least the number of stable matchings for a given problem instance can be determined in polynomial time, and this is one of the central issues that we shall investigate.

The concept of #P-completeness was introduced by Valiant [8], [9] (see also Garey and Johnson [2, pp. 167–170]) in order to describe those enumeration problems that are "at least as hard" as NP-complete problems. Enumeration problems associated

* Received by the editors August 16, 1984, and in revised form April 1, 1985.
† Department of Computing Science, University of Glasgow, Glasgow, Scotland G12 8QQ.
‡ Department of Computing, Salford College of Technology, Salford, England.

with most NP-complete problems (such as "how many Hamiltonian cycles does a given graph contain?") are #P-complete, but so also are many enumeration problems arising from polynomially solvable recognition or optimisation problems. We shall establish the stable marriage problem as one of these.

However, in pursuit of this objective, we shall obtain a good deal of insight into the fascinating structure of the set of stable matchings for a general stable marriage instance, and in so doing, we produce at least a partial answer to Knuth's sixth problem. We would claim that Theorem 4.1 provides a description of the kind sought by Knuth, and in view of the #P-completeness result (Theorem 5.2), it seems unlikely that any more effective characterisation of the stable matchings is possible.

2. The maximum number of stable matchings. One of the unsolved problems raised by Knuth [4] concerns the maximum number $f(n)$ of stable matchings for any problem instance of size n, and the nature of the instance(s) that yield(s) this maximum. In this section, we provide a construction which we might conjecture provides at least a partial answer to Knuth's question, and which certainly establishes the exponential growth of the function $f(n)$.

An instance of the stable marriage problem is specified by its male and female *preference matrices.* If both men and women are arbitrarily numbered $1, 2, \cdots, n$, then these matrices are defined by

$$mp(i,j) = k \Leftrightarrow \text{woman } k \text{ occupies position } j \text{ in the list of preferences of man } i;$$

$$wp(i,j) = k \Leftrightarrow \text{man } k \text{ occupies position } j \text{ in the list of preferences of woman } i.$$

Alternatively, a specification may be given in terms of the male and female *ranking matrices*, which we shall use in later sections, defined by

$$mr(i,k) = j \Leftrightarrow \text{woman } k \text{ occupies position } j \text{ in the preference list of man } i;$$

$$wr(i,k) = j \Leftrightarrow \text{man } k \text{ occupies position } j \text{ in the preference list of woman } i.$$

Clearly, $mr(i, mp(i,j)) = j$, $wr(i, wp(i,j)) = j$, etc.

Our result is based on the following lemma, for which we give a rather informal proof, the formal proof being a little messy and tedious.

LEMMA 2.1. *Given an instance I_n of size n of the stable marriage problem with $g(n)$ stable matchings, there exists an instance I_{2n} of size $2n$ with at least $2\{g(n)\}^2$ stable matchings.*

Proof. Let I_n be based on men and women labelled $1, \cdots, n$, and let I'_n be an isomorphic instance based on men and women labelled $n+1, \cdots, 2n$. Let mp_n, wp_n and mp'_n, wp'_n be the preference matrices for I_n and I'_n respectively.

The men and women in I_{2n} are labelled $1, \cdots, 2n$. The preference lists for the men in I'_n (respectively I_n) are appended to the preference lists for the men in I_n (respectively I'_n). The preference lists for the women in I_n (respectively I'_n) are prefixed by the preference lists for the women in I'_n (respectively I_n). This results in preference matrices as shown for I_{2n}:

$$mp_{2n} = \begin{bmatrix} mp_n & mp'_n \\ mp'_n & mp_n \end{bmatrix}, \qquad wp_{2n} = \begin{bmatrix} wp'_n & wp_n \\ wp_n & wp'_n \end{bmatrix}.$$

Now, let S be a stable matching in I_n, and S' a stable matching in I'_n. In I_{2n}, if each man from I_n is given his S-partner and each man from I'_n is given his S'-partner, then we have a stable matching in I_{2n}. For:

(i) no man/woman pair from I_n (respectively I'_n) can cause instability because of the stability of S (respectively S');

(ii) no man from I_n and woman from I'_n, or man from I'_n and woman from I_n, can cause instability because such a man prefers his partner to any such woman.

Also, if each woman from I_n is given the S'-partner of the corresponding woman in I'_n, and each woman from I'_n the S-partner of the corresponding woman from I_n, then we again have a stable matching in I_{2n}, by an essentially identical argument.

Hence, as S and S' run independently through all $g(n)$ stable matchings of I_n, we obtain $2\{g(n)\}^2$ stable matchings in I_{2n}. □

We are now able to establish the exponential growth of the function $f(n)$.

COROLLARY 2.1. *For each* i $(\geqq 0)$ *there exists a stable marriage instance of size* $n = 2^i$ *with at least* 2^{n-1} *stable matchings.*

Proof. Such a sequence of stable marriage instances is obtained by starting with the only stable marriage instance of size 1, which has of course a single stable matching, and repeatedly applying the construction of Lemma 2.1. If $g(n)$ denotes the number of solutions of the instance of size n so generated, then we have

$$g(n) \geqq 2\{g(n/2)\}^2, \qquad g(1) = 1,$$

from which the result of the corollary follows. □

In fact, it can be shown that the number of stable matchings in the family of instances described in Corollary 2.1 satisfies the recurrence relation

$$g(n) = 3\{g(n/2)\}^2 - 2\{g(n/4)\}^4 \qquad (n \geqq 4)$$

giving the table of values of $g(n)$ for $n = 2^k$, $k \leqq 5$ shown in Table 1. We might conjecture that this sequence of problem instances provides the answer, for the case when n is a power of 2, to Knuth's question concerning the maximum possible number of stable matchings. However, we have not even been able to solve the above recurrence relation to give a closed form for $g(n)$, so no proof of this conjecture is in sight.

TABLE 1

Size of instance	No. of stable matchings
1	1
2	2
4	10
8	268
16	195472
32	104310534400

3. The fundamental algorithm. In this section, we outline the "deferred acceptance" algorithm of McVitie and Wilson [5] and summarise the properties of the reduced preference lists that arise from its application.

The algorithm is based on a sequence of "proposals" from the men to the women.

When a woman receives a proposal, she:

(i) holds it for consideration if it is the best proposal she has so far received, simultaneously rejecting any poorer proposal already held;

(ii) rejects it outright if she already holds a better proposal.

(A "better" proposal means a proposal from some man higher in the woman's preference list.)

Each man proposes in turn to the women on his preference list, in order of course, pausing when a woman agrees to hold his proposal, and resuming his sequence of proposals upon any subsequent rejection.

It is not difficult to show, as in [5], that:

(i) the sequence of proposals so specified ends with every woman holding a single unique proposal, and that the proposals held constitute a stable matching;

(ii) the outcome of the sequence of proposals is independent of the order in which the men propose.

A similar outcome results if the roles of males and females are interchanged, in which case the resulting stable matching may or may not be the same as that obtained from the male proposal sequence. Throughout, we adopt a male-oriented approach, while recognising that a female-oriented approach would be equivalent in all respects.

Two fundamental implications of this initial proposal sequence, implicit in [5], are:

(i) if m proposes to w then there is no stable matching in which m has a better partner than w;

(ii) if w receives a proposal from m then there is no stable matching in which w has a worse partner than m.

These observations suggest that we should explicitly remove m from w's list, and w from m's, if w receives a proposal from someone she likes better than m. We refer to the resulting lists as the (male-oriented) *shortlists*. The following properties are either immediate, or are explicit or implicit in [5].

Property 1. If w does not appear on m's shortlist, then there is no stable matching in which m and w are partners.

Property 2. m appears on w's shortlist if and only if w appears on m's.

Property 3. w appears first on m's shortlist if and only if m appears last on w's.

Property 4. If every man is paired with the first woman on his shortlist then the resulting matching is stable; it is called the male optimal solution, for no man can have a better partner than he does in this matching, and indeed no woman can have a worse one; this is the solution given by the male proposal sequence. (Correspondingly, a female-oriented approach would yield the female optimal solution.)

It follows that, if all the shortlists have just a single entry, then they constitute the unique solution for that problem instance, but as we shall see later, if any shortlist has more than one entry, then there may well be other stable matchings.

Finally, in this section, we establish a theorem that we shall need later. In the proof of this theorem, as throughout, we regard a stable matching as a set of ordered male/female pairs.

THEOREM 3.1. *If man m and woman w are partners in some stable matching S then*

(i) *there is no stable matching S' in which both m and w have worse partners*;

(ii) *there is no stable matching S' in which both m and w have better partners.*

Proof. (i) This is immediate, for m and w would cause instability in such an S'.

(ii) Suppose that such an S' exists. Let M and W (respectively M' and W') denote the sets of men and women who have better partners in S than in S' (respectively in S' than in S). Then $m \in M'$, $w \in W'$. For the stability of S, every man in M' has an S'-partner in W, so $|M'| \leqq |W|$. For the stability of S', every woman in W has an S-partner in M', so $|W| \leqq |M'|$. Hence $|M'| = |W|$, and $|M| = |W'|$. So no man in M' can have an S-partner in W', giving a contradiction.

4. Rotations. In this section we introduce the concept of a rotation, which is crucial in establishing the relationship between the structure of a stable marriage

instance and that of an associated partially ordered set. This concept was first introduced and studied by Irving [3] under the name "all-or-nothing cycle" in the context of the stable roommates problem, which is a version of the stable marriage problem involving just a single set.

We recall that, in the male optimal solution, every man is partnered by the first woman on his shortlist. If we wish to generate a different stable matching, then it is clear that some of the men (two at the very least) must sacrifice their optimal partners.

Suppose that man m sacrifices his optimal partner, woman w. Then the best partner that man m can have is the woman that appears second on his shortlist, say woman v. But man m would represent an improvement, from woman v's point of view, on her partner, say man k, in the male optimal solution, so that, for the sake of stability, man k is forced to sacrifice woman v. This argument may be repeated for the second woman in man k's shortlist, and so on, generating a chain of forced sacrifices. Because of the finiteness of the problem, this process can end in one of only two ways, namely:

(i) the chain eventually cycles, yielding a sequence $m_0, m_1, \cdots, m_{r-1}$ $(r \geqq 2)$ of men such that the second woman on m_i's shortlist is first on that of m_{i+1}, where the subscripts are taken modulo r; or

(ii) the chain reaches a man whose shortlist contains just one woman.

In case (i), we refer to such a cycle of implications as a *rotation* relative to the shortlists. The terminology arises because, as we shall see, the partners can be rotated one place without destroying stability. A more precise definition is given below.

For a given instance of the stable marriage problem, a set of *reduced* preference lists is a set obtainable from the originals by zero or more deletions, such that:

(i) no list is empty;

(ii) woman w is absent from man m's list if and only if man m is absent from that of woman w.

Relative to such a set $\mathcal{S}$ of reduced preference lists, we denote by $\text{first}_{\mathcal{S}}(x)$, $\text{second}_{\mathcal{S}}(x)$ and $\text{last}_{\mathcal{S}}(x)$ the first, second and last person respectively on x's list. (We shall often omit the subscript $\mathcal{S}$ when it is obvious which set of lists applies.) Of course, $\text{second}_{\mathcal{S}}(x)$ may be undefined in the event that x's reduced list in $\mathcal{S}$ has just one entry.

A set $\mathcal{S}$ of reduced preference lists will be called *stable* if, for each man m and woman w,

(i) $w = \text{first}_{\mathcal{S}}(m)$ if and only if $m = \text{last}_{\mathcal{S}}(w)$;

(ii) w is absent from m's list if and only if $wr(w, m) > wr(w, \text{last}_{\mathcal{S}}(w))$.

(Note: mr and wr are the original ranking matrices for the instance of the problem.)

For brevity, we shall refer throughout to a stable set of reduced preference lists as a *stable set.*

LEMMA 4.1. *The shortlists form a stable set.*

Proof. Property (i) was noted in § 3 above.

For property (ii), we consider two cases:

Case (a): $mr(m, w) < mr(m, \text{first}(m))$. In this case, w is absent from m's shortlist if and only if m proposed to w and was rejected. This happens if and only if w already held, or subsequently received, a better proposal, so that the last proposal held by w must also be better—i.e. $wr(w, m) > wr(w, \text{last}(w))$.

Case (b): $mr(m, \text{first}(m)) < mr(m, w)$. In this case w is absent from m's shortlist if and only if m was removed from that of w by virtue of w receiving a proposal from someone better than m. As in case (a), this happens if and only if $wr(w, m) > wr(w, \text{last}(w))$. □

LEMMA 4.2. *If, relative to a stable set, each man m is partnered by* $\text{first}(m)$, *then there results a stable matching.*

Proof. By property (i), if $w = \text{first}(m_1) = \text{first}(m_2)$ then $m_1 = \text{last}(w) = m_2$, so that a matching is certainly specified.

If m prefers w to first (m), then w is absent from m's list. Hence, by property (ii), w prefers her partner, last (w), to m, and there can be no instability. □

The stable matching obtained from a stable set $\mathcal{S}$ as described in Lemma 4.2 will be referred to as the *stable matching corresponding to* $\mathcal{S}$.

LEMMA 4.3. *If, for some stable set,*

(i) $mr(m, \text{first}(m)) < mr(m, w)$ *and*

(ii) w *is absent from* m*'s list,*

then there is no stable matching in which m *and* w *are partners.*

Proof. By Lemma 4.2, $(m, \text{first}(m)) \in S$ for some stable matching S. Suppose that $(m, w) \in S'$ for some stable matching S'. Then m is better off in S than in S'.

Now, since w is absent from m's list, we have $wr(w, m) > wr(w, \text{last}(w))$. So w is better off in S (with last (w)) than in S' (with m), and Theorem 3.1(ii) is contradicted. □

An ordered sequence $(m_0, w_0), \cdots, (m_{r-1}, w_{r-1})$, $r \geqq 2$, of man/woman pairs forms a *rotation* in a stable marriage instance if, relative to some stable set $\mathcal{S}$,

$$w_{i+1} = \text{first}(m_{i+1}) = \text{second}(m_i)$$

for each i $(0 \leqq i \leqq r-1$, $i+1$ taken modulo $r)$.

The rotation is said to be *exposed* in $\mathcal{S}$.

LEMMA 4.4. *Let* $\mathcal{S}$ *be a stable set, and let* S *be the corresponding stable matching. If* S' *is a stable matching in which man* m *has a worse partner than* $\text{first}_{\mathcal{S}}(m)$ *(his partner in* S*), then there is a rotation exposed in* $\mathcal{S}$ *all of whose male members have worse partners in* S' *than in* S.

Proof. By Lemma 4.3, if first (m) is the only entry in m's list then there is no such stable matching S' and we have nothing to prove. Otherwise, we form the sequence $\{(m_i, w_i)\}$, where

(i) $m_0 = m$,

(ii) $w_i = \text{first}(m_i)$, $i = 0, 1, \cdots$,

(iii) $m_{i+1} = \text{last}(\text{second}(m_i))$, $i = 0, 1, \cdots$.

Since $w_0 = \text{first}(m_0)$, it follows that $(m_0, w_0) \in S$. By Lemma 4.3, m_0 has an S'-partner no better than second $(m_0) = \text{first}(m_1) = w_1$. Hence, for stability of S', w_1 has an S'-partner no worse than m_0, and therefore better than $m_1 = \text{last}(w_1)$. So, by Theorem 3.1(ii), m_1 must have an S'-partner worse than w_1, and so, in particular, by Lemma 4.3, second (m_1) is defined. This argument can be repeated to show that second (m_i) is defined for all i, and that all the m_i have worse partners in S' than in S.

Now, the sequence $\{(m_i, w_i)\}$ must cycle eventually; suppose $m_0, \cdots, m_{s-1}$ are distinct but $m_s = m_t$ for some t $(0 \leqq t \leqq s-2)$. Then it is immediate that $(m_t, w_t), \cdots, (m_{s-1}, w_{s-1})$ forms a rotation that is exposed in $\mathcal{S}$ and has the required property. □

For a given man m and a given stable set $\mathcal{S}$, we call the rotation obtained from m in the way described in Lemma 4.4 the rotation *generated* by m. Clearly, if m is himself in a rotation exposed in $\mathcal{S}$, then this is the rotation generated by m.

The following are immediate corollaries of Lemma 4.4.

COROLLARY 4.1. *If* $\mathcal{S}$ *is a stable set with corresponding stable matching* S*, then either:*

(i) *at least one rotation is exposed in* $\mathcal{S}$*; or*

(ii) *no man can have a worse partner in any stable matching than he has in* S *(and* S *is the female optimal solution).*

COROLLARY 4.2. *If* $(m_0, w_0), \cdots, (m_{r-1}, w_{r-1})$ *is a rotation, and if, in some stable matching* S, *some fixed* m_j *has a partner worse than* w_j, *then each of the* m_i *has an* S*-partner worse than* w_i.

We now introduce the notion of rotation elimination. Suppose that $\rho = (m_0, w_0), \cdots, (m_{r-1}, w_{r-1})$ is a rotation that is exposed in some stable set $\mathcal{S}$. If, for each i $(0 \leqq i \leqq r-1)$ all successors of m_i are deleted from w_{i+1}'s list (in $\mathcal{S}$), and w_{i+1} is removed from the corresponding men's lists, we shall say that the rotation has been *eliminated.* (Here, as usual, $i+1$ is taken modulo r.)

The significance of rotation elimination is contained in the next lemma.

LEMMA 4.5. *If a rotation is eliminated from a stable set* $\mathcal{S}$ *in which it is exposed, then the resulting set* $\mathcal{T}$ *of lists is also stable.*

Proof. We have to establish properties (i) and (ii) for the new lists.

(i) If m is not in the rotation, then

$$w = \text{first}_{\mathcal{T}}(m) \Leftrightarrow w = \text{first}_{\mathcal{S}}(m), \text{ since } \text{first}_{\mathcal{S}}(m) \text{ is not removed,}$$
$$\Leftrightarrow m = \text{last}_{\mathcal{S}}(w), \text{ since } \mathcal{S} \text{ is a stable set,}$$
$$\Leftrightarrow m = \text{last}_{\mathcal{T}}(w), \text{ since } \text{last}_{\mathcal{S}}(w) \text{ is not removed.}$$

If m is in the rotation, then

$$w = \text{first}_{\mathcal{T}}(m) \Leftrightarrow w = \text{second}_{\mathcal{S}}(m), \text{ since } \text{first}_{\mathcal{S}}(m) \text{ is removed but } \text{second}_{\mathcal{S}}(m) \text{ is not,}$$
$$\Leftrightarrow m = \text{last}_{\mathcal{T}}(w), \text{ since all successors of } m \text{ are removed from } w\text{'s list.}$$

(ii) w is absent from m's new list

$\Leftrightarrow$ (w was already absent from m's old list) or

(m was removed from w's list during the rotation elimination)

$\Leftrightarrow (wr(w, m) > wr(w, \text{last}_{\mathcal{S}}(w))$ or

$(wr(w, \text{last}_{\mathcal{S}}(w)) > wr(w, m) > wr(w, \text{last}_{\mathcal{T}}(w)))$

$\Leftrightarrow wr(w, m) > wr(w, \text{last}_{\mathcal{T}}(w))$. □

We are now in a position to establish the one-to-one relationship between stable sets and stable matchings for any given instance of the stable marriage problem.

LEMMA 4.6. *For a given instance of the stable marriage problem, there is a one-to-one correspondence between the stable matchings and the stable sets. Furthermore, each stable set can be obtained from the set of shortlists by a sequence of zero or more rotation eliminations.*

Proof. Given a stable set, a unique stable matching can be constructed as in Lemma 4.2.

On the other hand, given a stable matching $S = \{(m_0, w_0), \cdots, (m_{n-1}, w_{n-1})\}$, we construct a stable set as follows.

We start from the set $\mathcal{M}$ of shortlists which, of course, has corresponding stable matching M, the male optimal solution. If $S \neq M$, then for some i, $w_i \neq \text{first}_{\mathcal{M}}(m_i)$. Now, by Lemma 4.3, w_i is in m_i's shortlist, and by Lemma 4.4, the exposed rotation ρ generated by m_i is such that all of its male members have worse partners in S than in M. If ρ is eliminated to obtain, by Lemma 4.5, a new stable set, again by Lemma 4.3,

w_i remains in m_i's list, and vice-versa, for all i. This process may now be repeated relative to the new set of lists, and then again, as often as necessary, until $w_i = \text{first}(m_i)$ for all i. We shall then have a stable set to which the stable matching S corresponds, and it will have been obtained by a sequence of zero or more rotation eliminations starting from the set of shortlists. □

LEMMA 4.7. *In a given stable marriage instance, no pair* (m, w) *can belong to two different rotations.*

Proof. Suppose that the pair (m, w) belongs to rotations ρ_1 and ρ_2, and that (m', w') belongs to ρ_1 but not to ρ_2. We shall show that these assumptions lead to a contradiction.

If $\mathcal{S}$ is a stable set in which ρ_2 is exposed, and if v is the partner of m' in the corresponding stable matching S, then $mr(m', v) \leqq mr(m', w')$. For otherwise, Corollary 4.2, applied to ρ_1, would force m to have an S-partner worse than w.

Let $\mathcal{T}$ be the stable set, and T the corresponding stable matching, obtained by eliminating ρ_2 from $\mathcal{S}$. We consider two cases.

Case (a). $(m', v) \in \rho_2$: since $(m', w') \notin \rho_2$ it follows that $v \neq w'$, and $mr(m', v) < mr(m', w')$. Since $(m, w) \in \rho_2$ and $\mathcal{T}$ arises from the elimination of ρ_2, we deduce that, in T, m has a partner worse than w. However, Lemma 4.3 applied to the stable set $\mathcal{S}$ reveals that, in $\mathcal{S}$, w' is present in the list of m'. Further, this presence is not affected by the elimination of ρ_2, so that, in T, m' has a partner at least as good as w'. Therefore pairs (m, w), (m', w') of ρ_1 and the stable set $\mathcal{T}$ provide a contradiction of Corollary 4.2.

Case (b). $(m', v) \notin \rho_2$: then the elimination of ρ_2 from $\mathcal{S}$ does not affect the presence of v in the list of m', so that (m', v) is in T. Exactly as in Case (a), we have a contradiction of Corollary 4.2. □

LEMMA 4.8. *Suppose* (m, w) *belongs to a rotation. Then*:

(i) (m, w) *belongs to some stable matching*;

(ii) *in a stable set obtained from the shortlists by a sequence of rotation eliminations,* w *is absent from* m*'s list if and only if the rotation containing* (m, w) *has been eliminated.*

Proof. (i) There must be a stable set in which the rotation containing (m, w) is exposed, and m and w are partners in the corresponding stable matching.

(ii) When a rotation ρ is eliminated, entries may disappear from m's list in two ways:

(a) the first entry will disappear if $(m, \text{first}(m)) \in \rho$;

(b) one or more entries may disappear as a result of the deletion of m from one or more of the women's lists.

If w were to disappear from m's list by method (b), then immediately thereafter, the conditions of Lemma 4.3 would apply and m and w could not, after all, be partners in a stable matching. □

Given a rotation ρ of a stable marriage instance, there may be several stable sets in which ρ is exposed. As observed in Lemma 4.6, any such stable set can be obtained from the shortlists by a sequence of rotation eliminations. If it should happen that no stable set in which ρ is exposed can be obtained without eliminating rotation π, then we say that π is a *predecessor* of ρ, and write $\pi < \rho$.

The relation $\leqq$ is clearly antisymmetric, transitive and reflexive, and therefore defines a partial order on the set of rotations. We call this the *rotation poset* for that stable marriage instance.

As further terminology, we say that π is an *immediate predecessor* of ρ if $\pi < \rho$ and there is no σ such that $\pi < \sigma < \rho$. We also refer to a subset of a poset that is closed under predecessors simply as a *closed subset.*

An alternative representation for the poset is in the form of an acyclic directed graph with a node representing each rotation, and an arc from the node representing rotation π to the node representing rotation ρ if and only if π is an immediate predecessor of ρ.

An *antichain* in a poset $P, \leqq$ is a subset A of P containing no elements π, ρ such that $\pi < \rho$. Given such an antichain A, we define the *closure* A^* of A by

$$A^* = \{\pi \in P: \pi < \rho \text{ for some } \rho \in A\}.$$

It is clear that, for any closed subset C of P, there is a unique antichain A such that $A^* = C$. We call this the *spanning antichain* of C.

We are now in a position to establish a theorem that is central to our main result.

THEOREM 4.1. *For any stable marriage instance, there is a one-to-one correspondence between the stable matchings for that instance and the antichains of its rotation poset.*

Proof. Given an antichain A, it is immediate that A^* is closed, so that all rotations in the set represented by A^* can be eliminated, one by one, starting from the shortlists, to produce a stable set with its corresponding stable matching.

If two different sets of rotations are eliminated, then it follows from Lemma 4.8(ii) that the resulting stable sets are different, and hence so also are the stable matchings. Since $A \neq B \Rightarrow A^* \neq B^*$, it follows that different antichains yield different stable matchings.

On the other hand, by Lemma 4.6, any stable matching arises from the shortlists via a sequence of rotation eliminations. The set $\mathscr{R}$ of rotations concerned must be closed, for a rotation cannot be eliminated until it is exposed, and it cannot become exposed until all of its predecessors have been eliminated. So this set $\mathscr{R}$ has a unique spanning antichain. □

As well as giving a characterisation of the set of stable matchings for a given problem instance, Theorem 4.1 provides the basis of an efficient algorithm for the generation of these stable matchings. Such an algorithm would involve the derivation of the rotation poset, suitably represented, the determination of all of its antichains, and the interpretation of each one as a stable matching.

5. #P-completeness of stable marriage enumeration. The problem of determining the number of stable matchings for a given stable marriage instance is clearly in #P, for any particular matching can be checked for stability in polynomial time. In order to prove the problem #P-complete, it suffices to describe a polynomial transformation (a "parsimonious" transformation, in the terminology of Garey and Johnson [2]) from a known #P-complete problem such that the instance of the original enumeration problem and the corresponding instance of the stable marriage problem have the same number of solutions. We shall now demonstrate such a transformation, using the following theorem of Provan and Ball [6].

THEOREM 5.1. *Determining the number of antichains in a poset is* #P*-complete.*

We are now in a position to state and prove our main theorem.

THEOREM 5.2. *Given a poset $P, \leqq$ with n elements, there exists an instance I of the stable marriage problem, constructible from $P, \leqq$ in time polynomial in n, such that the stable matchings of I are in one-to-one correspondence with the antichains of $P, \leqq$.*

In order to prove Theorem 5.2, we first describe a transformation, and then we prove that the transformation has the required property.

From the poset, we first form an acyclic directed graph with one node for each element of P, and an arc from node u to node v if u is an immediate predecessor of v. Two extra nodes are included in this directed graph; one, the source, has an arc to

every node representing a minimal element of the poset, and the other, the sink, has an arc from every node representing a maximal element. The arcs of this acyclic directed graph are numbered arbitrarily $1, \cdots, t$. Each node, except the source and the sink, is labelled with an ordered subset, of size $\geqq 2$, of $\{1, \cdots, t\}$, namely the subset consisting of the numbers on the arcs incident to or from that node, with those numbers in an arbitrary but fixed order, say increasing order.

In the course of the construction, from this digraph, of the stable marriage instance, each node of the digraph, with the exception of the source and the sink, is "processed" according to rules that we prescribe below. In order to ensure that no node is processed before one of its predecessors, the nodes are processed according to a *topological order.* This is a one-to-one mapping f from the nodes of the digraph onto the set $\{0, \cdots, n-1\}$ such that if node u is a predecessor of node v then $f(u) < f(v)$. (It is well-known that such a topological ordering of the nodes of an acyclic digraph can be found in time polynomial in the number of nodes.) During this "node processing" phase, partial preference lists for the men are built up in natural (first to last) order, while those for the women are built up in reverse order; after processing all the nodes, each list is completed by appending absentees, in arbitrary order, after those already present.

The stable marriage instance involves t men and t women; the first woman on man i's list is woman i, and man i is also initially placed on woman i's list ($i = 1, 2, \cdots, t$). So the male optimal solution pairs man i with woman i for each i ($1 \leqq i \leqq t$).

Suppose that, after the first k ($\geqq 0$) nodes of the digraph have been processed, in the chosen topological order, the woman most recently appended to man i's list is $w(k, i)$ ($1 \leqq i \leqq t$). Then node $k+1$, labelled $\{a_0, \cdots, a_{r-1}\}$ say, is processed as follows: for each i ($0 \leqq i \leqq r-1$) woman $w(k, a_{i+1})$ is appended to man a_i's list, and man a_i is placed on woman $w(k, a_{i+1})$'s list ahead of any men already there, the most recent of whom is clearly man a_{i+1}. Hence

$$mr(a_i, w(k, a_{i+1})) = 1 + mr(a_i, w(k, a_i))$$

and

$$wr(w(k, a_{i+1}), a_{i+1}) = 1 + wr(w(k, a_{i+1}), a_i)$$

for $i = 0, \cdots, r-1$, where $i+1$, throughout, is taken modulo r.

As mentioned previously, once all the nodes have been processed, each list is completed by appending, in arbitrary order all the members of the opposite sex not already present in that list. It is clear that the entire construction can be carried out in time polynomial in n. We first have to justify the claim that this construction produces a genuine stable marriage instance of size t.

LEMMA 5.1. *The construction described above does yield a stable marriage instance of size t.*

Proof. It suffices to show that, during the processing of the nodes, no woman can be appended twice to the same man's list (and therefore that no man can be included twice in the same woman's list).

For a given arc i in the digraph, denote by init (i) and term (i) the initial and terminal nodes respectively of that arc. Consider to which men's lists woman i is appended during the processing of the nodes. Woman i is first involved when start (i) is processed, where start (i) is defined to be init (i) unless init (i) is the source, in which case start (i) is defined to be term (i). At this point she is appended to the list of man i_1 ($\neq i$), where i_1 immediately precedes i in the label of node start (i). If start (i) = term (i_1) then the list of man i takes no further part in the processing of the nodes, so woman i appears on no other lists. Otherwise, when term (i_1) is processed,

woman i is appended to the list of man i_2, where i_2 labels term (i_1). Proceeding thus, we obtain a sequence $i_1, i_2, \cdots, i_s$ $(s \geqq 2)$ of men such that

(i) start (i) = init (i_1);

(ii) init (i_j) = term (i_{j-1}) $(j = 2, \cdots, s-1)$;

(iii) stop (i_s) = term (i_{s-1}), where stop (i_s) is defined to be term (i_s) unless term (i_s) is the sink, in which case stop (i_s) is defined to be init (i_s);

(iv) during the processing of the nodes, woman i is appended to the lists of men $i, i_1, \cdots, i_s$, and to no others.

Clearly, by (ii), arcs $i_1, \cdots, i_{s-1}$ form a directed path in the acyclic digraph, and so are all distinct. Also, by (i) and (iii), i and i_s are distinct from $i_1, \cdots, i_{s-1}$. Finally $i \neq i_s$, for otherwise $i_1, \cdots, i_{s-1}$ is a path from start (i) to stop (i_s) and i and i_s could be the same arc only if start (i) = init (i_s) = init (i) and stop (i_s) = term (i) = term (i_s). But then the path $i_1, \cdots, i_{s-1}$ would contradict the fact that init (i) is an immediate predecessor of term (i). □

LEMMA 5.2. *For the stable marriage instance constructed above, the set of shortlists contains precisely the men and women appended before and during the processing of the nodes.*

Proof. This is obvious and is left as an exercise for the reader. □

LEMMA 5.3. (i) *If* $\{a_0, \cdots, a_{r-1}\}$ *is the label on the digraph node numbered* k *in the chosen topological ordering, then the stable marriage instance constructed above contains a rotation*

$$\rho_k = (a_0, b_0), \cdots, (a_{r-1}, b_{r-1})$$

where

$$b_i = w(k-1, a_i).$$

This rotation has as its predecessors precisely those rotations ρ_j *for which node* j *is a predecessor of node* k. *Furthermore, when* ρ_k *is eliminated from any stable set in which it is exposed, only* b_i *is removed from* a_i*'s list, for each* i $(0 \leqq i \leqq r-1)$, *and no woman is removed from any other man's list.*

(ii) *There are no rotations other than those described in* (i).

Proof. (i) We first observe that the statement of (i) is true for node number 1. For, when this node was processed during the construction of the stable marriage instance, we set

$$\text{second}_{\mathcal{M}}(a_i) = mp(a_i, 2) = w(0, a_{i+1}) = mp(a_{i+1}, 1) = \text{first}_{\mathcal{M}}(a_{i+1})$$

for each i, and this shows that a rotation ρ_1 of the stated kind is exposed in $\mathcal{M}$. Clearly, the rotation, like the node, has no predecessors. Further, when ρ_1 is eliminated, it is immediate that $b_i = w(0, a_i) = mp(a_i, 1)$ is removed from a_i's list. But since $wr(b_i, a_{i-1}) = wr(b_i, a_i) - 1$, for each i, no other entries are removed from any of the men's lists.

We now assume that (i) is true for all nodes numbered up to k, and we prove that, as a consequence, it must also be true for node $k+1$. Starting from $\mathcal{M}$, for each node j that is a predecessor of node $k+1$, there is, by the induction hypothesis, a rotation ρ_j which may be eliminated, provided this is done in topological order. Let $\mathcal{S}$ be the stable set obtained after this sequence of rotation eliminations, and consider the preference list of a_i in $\mathcal{S}$.

Case (a). Node $k+1$ is init (a_i), or init (a_i) is the source. Then a_i is not a label on any predecessor of node $k+1$, so no woman has been removed from a_i's list. Hence $\text{first}_{\mathcal{S}}(a_i) = \text{first}_{\mathcal{M}}(a_i) = w(k, a_i)$.

Case (b). Node $k+1$ is term (a_i), and init (a_i) is not the source. Then init (a_i) is a predecessor of term (a_i), and just woman $\text{first}_{\mathcal{S}}(a_i)$ has been removed from a_i's list, namely when $\rho_{\text{init}(a_i)}$ was eliminated. Hence, again

$$\text{first}_{\mathcal{S}}(a_i) = \text{second}_{\mathcal{M}}(a_i) = w(k, a_i).$$

In both cases, because

$$mr(a_i, w(k, a_{i+1})) = 1 + mr(a_i, w(k, a_i))$$

we see that

$$\text{second}_{\mathcal{S}}(a_i) = w(k, a_{i+1}) = \text{first}_{\mathcal{S}}(a_{i+1}).$$

Therefore ρ_{k+1} is a rotation as specified, and is exposed in $\mathcal{S}$.

If some rotation ρ_j is not eliminated, where node j is a predecessor of node $k+1$, then neither is some rotation ρ_m for a node m that is an immediate predecessor of node $k+1$. Hence there is an a_i for which node $k+1$ is term (a_i) and $\rho_{\text{init}(a_i)}$ has not been eliminated. But then

$$\text{first}_{\mathcal{S}}(a_i) = \text{first}_{\mathcal{M}}(a_i) \neq w(k, a_i)$$

and the rotation ρ_{k+1} is not exposed.

Finally, when ρ_{k+1} is eliminated, it is an immediate consequence of the fact that $wr(b_i, a_{i-1}) = wr(b_i, a_i) - 1$, for each i, that the only entries removed from the men's lists are the b_i from the list of a_i $(0 \leqq i \leqq r-1)$.

(ii) Clearly (m, w) cannot be in a rotation if w is not in m's shortlist, nor if $w = \text{last}_{\mathcal{M}}(m)$. All other pairs (m, w) are in one of the rotations ρ_k described in (i), so that (ii) follows from Lemma 4.7. □

Proof of Theorem 5.2. This is now an immediate consequence of the given construction, Lemmas 5.1 and 5.3, together with Theorem 4.1. □

COROLLARY 5.1. *Determining the number of stable matchings for an instance of the stable marriage problem is* #P-*complete.*

6. Conclusions. In this paper, a characterisation of the stable matchings for any stable marriage instance as the antichains of the associated rotation poset has been presented. It has been shown further that every finite poset is the rotation poset of some stable marriage instance, obtainable from the poset in polynomial time. Therefore, the fact that enumerating the antichains of a poset is #P-complete leads to the conclusion that counting stable marriages is also #P-complete.

It has also been pointed out that this characterisation of the stable matchings could be exploited to yield an algorithm for the generation of all stable matchings.

A new constructive proof has been given to show that the maximum number $f(n)$ of stable matchings in a stable marriage instance of size n grows exponentially with n, leading to a conjecture concerning the precise nature of $f(n)$. It would be of interest to investigate whether the new characterisation of stable matchings can be used to throw light on the function $f(n)$, or on the function representing the *average* number of stable matchings per problem instance of size n, a function about which next to nothing appears to be known.

Acknowledgments. The authors are grateful to the referees and to Dan Gusfield for their helpful comments. Work on this paper was carried out while the first author was in the Department of Mathematics and Computer Science, University of Salford, Salford, England.

REFERENCES

[1] D. GALE AND L. S. SHAPLEY, *College admissions and the stability of marriage*, Amer. Math. Monthly, 69 (1962), pp. 9-15.
[2] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, Freeman, San Francisco, CA, 1979.
[3] R. W. IRVING, *An efficient algorithm for the stable room-mates problem*, J. Algorithms, to appear.
[4] D. E. KNUTH, *Mariages stables*, Les Presses de l'Université de Montréal, Montreal, 1976.
[5] D. MCVITIE AND L. B. WILSON, *The stable marriage problem*, Comm. ACM, 14 (1971), pp. 486-492.
[6] J. S. PROVAN AND M. O. BALL, *The complexity of counting cuts and of computing the probability that a graph is connected*, this Journal, 12 (1983), pp. 777-788.
[7] G. PÓLYA, R. E. TARJAN AND D. R. WOODS, *Notes on Introductory Combinatorics*, Birkhauser Verlag, Boston, MA, 1983.
[8] L. G. VALIANT, *The complexity of computing the permanent*, Theoret. Comp. Sci., 8 (1979), pp. 189-201.
[9] ———, *The complexity of enumeration and reliability problems*, this Journal, 8 (1979), pp. 410-421.
[10] N. WIRTH, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
004

ON DETERMINISTIC MULTI-PASS ANALYSIS*

CLAUDIO CITRINI†, STEFANO CRESPI-REGHIZZI‡ AND DINO MANDRIOLI‡§

Abstract. Chains (or cascade composition) of push-down transducers are introduced as a model of multi-pass compilers. We focus on deterministic chains, since nondeterministic transducer chains of length two define the recursively enumerable sets. Deterministic chains recognize in linear time a superset of context-free deterministic languages. This family $\mathscr{CH}$ is closed under Boolean operations, disjoint shuffle, and reverse deterministic pushdown translation, but not under homomorphism. Equivalent definitions of the family in terms of composition of syntax-directed translation schemes and control languages are considered. The family is a strict hierarchy ordered by the length of the chain. The complexity of $\mathscr{CH}$ is obviously linear, but not all linear-time parsable languages are in $\mathscr{CH}$. On the other hand it strictly includes the Boolean closure of deterministic languages. Finally $\mathscr{CH}$ is not comparable with another classical Boolean algebra of formal languages, namely real-time languages.

Key words. multi-pass translation, cascade composition of push-down automata, deterministic push-down transducers, syntax-directed translation, deterministic languages, *LR* grammars, control languages, Boolean algebra of languages

AMS (MOS) subject classifications. 68Q45, 68Q05, 68Q50, 68N20

1. Introduction.

The theory of syntax directed translation is in widespread use for compiler design. Translation processes of different complexity have been modeled by abstract transducers (finite-state, push-down) or by equivalent syntax directed translation schemes. In this formal picture a rather important part is surprisingly missing: the study of multi-pass translation, a technique frequently adopted for designing complex compilers, from the earliest Algol 60 projects. This omission is the more startling if one looks back to the early studies of multi-pass sequential machines (e.g., Hartmanis and Stearns (1966)). In contrast very little research on composition of deterministic push-down transducers is known to us (studies of composition of tree transducers do not address the same problem, as the parsing of a string is not tackled by a tree automaton). Scattered studies concentrating on special cases have been published by Aho and Ullman (1972-73) in connection with bottom-up polish translations, by Khabbaz (1974) for operator precedence grammars, and by Tadashi Ae (1977) for nondeterministic push-down transducers. The latter case is really out of our main interest, since composition of two such machines can model any Turing machine, whereas our major motivation is from compilation, where slower than linear algorithms are seldom used. For completeness we mention that the family of deterministic cancellation push-down automata of Vitànyi and Savitch (1978) is a restricted case of our 2-chains. The main objectives of this study are:

—to provide a formal model of multi-pass translation;
—to investigate the power of translation and language recognition which can be performed by a chain of deterministic push-down devices;
—to study formal properties of languages recognizable by such a chain.

It is worth contrasting our goals with those of earlier studies on sequential machine (de)composition. Since in that case cascade composition does not affect the family of languages recognized (the regular languages), motivation came from a concern for minimization: how to build a chain which performs as a single automaton, but such

* Received by the editors October 6, 1983, and in final revised form April 8, 1985. This work was supported by a grant of Ministero Pubblica Istruzione.

† Dipartimento di Matematica, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy.
‡ Dipartimento di Elettronica, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy.
§ Previously at Istituto di Matematica, Informatica e Sistemistica, Università di Udine.

that each stage is smaller than the original automaton. On the other hand, we are interested in studying gains in computing power obtained from cascade composition. Minimality could also be relevant for chains of push-down transducers, but is not pursued in this paper. Our research is connected with other areas of the theory of computation and formal languages, namely: it extends the notion of deterministic push-down transducer in the direction undertaken by Wotschke (1978) and by Kintala (1979), which already considered unions and other nonclosed operations on deterministic languages; it relates to linear-time multi-tape Turing machine complexity.

The paper is organized as follows.

In § 2 we define chains of deterministic push-down translators and related languages, and illustrate them by examples. Some classical counterexamples of the inadequacies of the (deterministic) context-free model are shown to be in the new family of languages.

In § 3 we contrast the parsing and translation power of deterministic push-down transducers with two other formalisms (which have a strong relevance in formal language theory), namely syntax directed translation schemes and control languages. It is shown that the different power of one-pass analyzers and translators of various models vanishes when they are serially composed.

In § 4 some closure properties of the new family are investigated. In particular it is shown to be a Boolean algebra but not an AFL.

In § 5 we show that our family of languages is a strict hierarchy with respect to the length of the chain. The proof of this intuitive result is rather complex, and is based on a generalisation of a lemma by Ginsburg and Greibach (1966). Details of the proof are in the Appendix.

In § 6 the relations with other classes of languages are investigated: it is shown that our family strictly includes the Boolean closure of deterministic languages and is strictly included in the class of linear time recognizable languages, but it is not comparable with real-time languages. This result is proved by means of a corollary of the hierarchy theorem.

Some open problems are listed both along the paper and in the conclusion. For the sake of clarity, we tried to use convincing but somewhat informal arguments in the simpler proofs. The exception is the proof of the hierarchy theorem, which needs a careful analysis. In this case the details are postponed into the Appendix.

Part of the contents of this paper has been presented in a preliminary version in Citrini, Crespi-Reghizzi, Mandrioli (1983).

2. Fundamental definitions and examples.

2.1. Definitions. For definitions not included here we refer to the texts by Aho and Ullman (1972–73), Hopcroft and Ullman (1969), and Salomaa (1973). The transducers to be considered will be deterministic, except for some side remark.

Let us recall some basic definitions. Let $T=(Q, \Sigma, \Gamma, \Omega, \delta, q_0, Z_0, F)$ be a deterministic push-down transducer (dpdt). If the output alphabet Ω is dropped and δ is accordingly projected, we obtain the *subjacent* deterministic push-down automaton (dpda).

The term *configuration* of a dpdt will either denote the quadruple $C=\langle q, x, \alpha; u\rangle$, $q\in Q$, $x\in\Sigma^*$, $\alpha\in\Gamma^*$, $u\in\Omega^*$, or the triple $C=\langle q, x, \alpha\rangle$, when the output is irrelevant. The choice will be made clear by the context.

It is convenient to consider so-called d (for deterministic) computations and loop-free transducers, as in Ginsburg and Greibach (1966) and Harrison (1978). A *d-computation* is defined via the relation $\vdash^{d*}$ as follows. Let $C_1=\langle q_1, x_1, \alpha_1\rangle$ and

$C_2 = \langle q_2, x_2, \alpha_2 \rangle$ be two configurations. Then

$$C_1 \vdash^{d*} C_2$$

if and only if

i) $\langle q_1, x_1, \alpha_1 \rangle \vdash^* \langle q_2, x_2, \alpha_2 \rangle$ with $\alpha_2 = \beta Z$, $Z \in \Gamma$, and

ii) $\delta(q_2, \varepsilon, Z)$ is undefined.

Intuitively a d-computation cannot be extended by ε-moves, that is the automaton has gone as far as possible in computing on $\langle q_1, x_1, \alpha_1 \rangle$ without reading the first symbol of x_2.

A dpda is *loop-free* if for all $x \in \Sigma^*$ there exists $q \in Q$ and $\alpha \in \Gamma^*$ such that $\langle q_0, x, Z_0 \rangle \vdash^{d*} \langle q, \varepsilon, \alpha \rangle$. A dpdt is loop-free if its subjacent dpda is loop-free. Intuitively a loop-free machine always exhausts its input string and stops.

Throughout the paper we shall suppose (without loss of generality), that any dpdt or dpda is loop-free.

DEFINITION 2.1. Let T be a dpdt.

1. The *translation generated by* T during the computation $C_1 \vdash^* C_2$, where $C_1 = \langle q_1, zx_2, \alpha_1; u_1 \rangle$ and $C = \langle q_2, x_2, \alpha_2; u_1 u \rangle$ is

$$\tau(C_1 \vdash^* C_2) \triangleq u.$$

2. If C is $\langle q, x, \alpha \rangle$, $\tau(C)$ is defined by

$$\tau(C) \triangleq \tau(\langle q, x, \alpha; \varepsilon \rangle \vdash^{d*} \langle q', \varepsilon, \alpha'; u \rangle) = u.$$

3. $\tau(x)$, $x \in \Sigma^*$, is defined by

$$\tau(x) \triangleq \tau(\langle q_0, x, Z_0 \rangle).$$

Notice that the above definitions are well posed because T is loop-free.

DEFINITION 2.2. A *chain* S (or *cascade composition*) of $k \geqq 1$ dpdt's $T_1, T_2, \cdots, T_k$, is a sequence, written $\langle T_1, T_2, \cdots, T_k \rangle$, such that

$$\Sigma_1 = \Sigma, \quad \Sigma_2 = \Omega_1, \quad \cdots, \quad \Sigma_k = \Omega_{k-1}.$$

S is termed a *k-chain* and k is its *length*. Similarly, we denote by $\langle S_1, S_2, \cdots, S_m \rangle$ the cascade composition of the chains $S_1, S_2, \cdots, S_m$.

DEFINITION 2.3. *The language* $L(S)$ *accepted by a k-chain* S *is*

$$L(S) = \{x \in \Sigma^* \mid \tau_{k-1}(\cdots(\tau_1(x))\cdots) \in L(A_k)\}$$

where A_k is the dpda subjacent to T_k, and τ_j is the translation computed by T_j, $1 \leqq j < k$.

The translation σ *computed by a k-chain* $S = \langle T_1, \cdots, T_k \rangle$ is the mapping $\Sigma^* \to \Omega_k^*$ defined as

$$\sigma(x) = \tau_k(\tau_{k-1}(\cdots \tau_1(x) \cdots)).$$

The translation computed by the subchain $\langle T_1, \cdots, T_j \rangle$ will be denoted by σ_j; in particular $\sigma_1 = \tau_1$ and $\sigma_k = \sigma$.

Remark. Alternatively and equivalently we could assume that $\sigma_{j+1}(x)$ is defined only if $\tau_j(\cdots \tau_1(x) \cdots) \in L(A_{j+1})$. The alternative definition is only used in § 3.

DEFINITION 2.4. The family $\mathscr{CH}_k$ is the set of languages recognized by some k-chain.

$\mathscr{CH}$ is $\bigcup_{k=1}^{\infty} \mathscr{CH}_k$. Obviously $\mathscr{CH}_k \subseteq \mathscr{CH}_{k+1}$. $\mathscr{CH}_1$ equals the family of deterministic context-free languages.

A k-chain S is *nondeterministic* if some T_j, $i \leqq j \leqq k$, is nondeterministic. The definitions of $L(S)$ and $\sigma(x)$ could be obtained in the natural way recalling that $\tau(x)$

is a set of strings; they are omitted because the nondeterministic case is only of marginal interest for this article.

DEFINITION 2.5. The family $\mathcal{NCH}_k$ is the set of languages recognized by some nondeterministic k-chain.

Unless otherwise stated, all chains considered in this paper are deterministic.

2.2. Examples. In order to give an idea of the extension of the family of languages which are accepted by a chain of dpdt (even with only few transducers), we present some examples, covering both nondeterministic and inherently ambiguous context-free languages and noncontext-free ones. Most tedious details are omitted for brevity.

Example 2.1. $\{a^n ba^{2n} ba^{3n} b \cdots ba^{kn} b \mid n \geqq 0,\ k \text{ fixed}\} \in \mathcal{CH}_2$.

Hint. T_1 produces $\tau_1(x) = x1$ if $x \in \{a^n ba^{2n} ba^{3m} ba^{4m} \cdots\} = x0$ (or $x10$) otherwise; T_2 verifies whether $\tau_1(x) \in \{a^n ba^{2m} ba^{3m} ba^{4p} ba^{5p} \cdots 1\}$.

Many generalizations are possible.

Example 2.2. $\{a^n ba^{2n} ba^{3n} b \cdots ba^{kn} \mid n \geqq 0,\ k \geqq 1\} \in \mathcal{CH}_3$.

Hint. T_1 "accepts" (i.e. produces an acceptable output) only if the input is of the form $x = a^i ba^j ba^h \cdots ba^q ba^r$ and outputs easily

$$\sigma_1(x) = a^i c^j a^j c^h a^h \cdots c^q a^q c^r;$$

T_2 verifies whether $i \leqq j \leqq h \leqq \cdots \leqq q \leqq r$, and gives

$$\sigma_2(x) = a^i c^{j-i} a^{j-i} c^{h-j} a^{h-j} \cdots c^{r-q}.$$

Lastly T_3, by comparing consecutive pairs of exponents, accepts x if and only if $i = j - i$, $j - i = h - j, \cdots$.

Example 2.3. The languages $\{uu^R c \mid u \in \Sigma^*,\ c \notin \Sigma\}$, where, as usual, u^R denotes the mirror image of u, and $\{uuc \mid u \in \Sigma^*,\ c \notin \Sigma\}$ both belong to $\mathcal{CH}_3$.

Notice that $\{uu^R c\}$ is a context-free nondeterministic language because of Corollary 2 of Theorem 3.4 and Corollary 1 of Theorem 3.5 of Ginsburg and Greibach (1966). $\{uuc\}$ is noncontext-free.

Hint. The accepting chains are similar. T_1 "accepts" a string $xc \in \Sigma^* c$ iff its length is odd, say $|xc| = 2m+1$, producing $\sigma_1(xc) = c^m x^R$. Now T_2 marks the center of x^R by a c sign, producing $\sigma_2(xc) = z^R c y^R$ [$z^R c y$ in the second case], with $|y| = |z|$, $x = yz$. Lastly T_3 must only check if $\sigma_2(xc) \in \{wcw^R \mid w \in \Sigma^*\}$, a context-free deterministic language.

Example 2.4. The language $L_j =$

$$\{a_1^{n_1} a_2^{n_2} \cdots a_j^{n_j} a_1^{n_1} a_2^{n_2} \cdots a_j^{n_j} \mid n_i \geqq 0,\ 1 \leqq i \leqq j\}$$

belongs to $\mathcal{CH}_2$ for every fixed j. In general, $\{ucu \mid u \in \Sigma^*,\ c \notin \Sigma\} \in \mathcal{CH}_2$.

This example is interesting because we need only a (fixed) 2-chain independent of j, despite the fact that we deal with an infinite hierarchy of j-intersection languages (Liu and Weiner (1973)). The chain is just a simplified version (without T_1) of Example 2.3.

Example 2.5. The language $\{a^{n_1} ba^{n_2} b \cdots a^{n_k} \mid k \geqq 2,\ n_i \geqq 0,\ 1 \leqq i \leqq k,$ and, for some $i,\ 1 < i \leqq k,\ n_1 = n_i\} \in \mathcal{CH}_2$. Notice that this is an inherently ambiguous context-free language, with unbounded degree of ambiguity (Harrison 1978)).

Hint. $\tau_1(x) = a^{n_1} c^{n_2} a^{n_2} c^{n_3} a^{n_3} \cdots$. Then T_2 pushes a^{n_1} on the stack, and compares n_2 with n_1 by reading c's. If $n_1 = n_2$, T_2 accepts; otherwise, if $n_1 < n_2$ (the case $n_1 > n_2$ is easier), it pushes the remaining $(n_2 - n_1)$ c's on the stack. By reading the next n_2 a's, the stack is first emptied, and then grows again to n_1 symbols, to be compared with c^{n_3}, etc.

3. Chains of transducers, syntax directed translations and control languages. In this section we compare $\mathscr{CH}$ with cascaded syntax directed translations and with control languages, and show their essential equivalence. However, more generality will be claimed, in some sense, for $\mathscr{CH}$.

DEFINITION 3.1. A *simple syntax directed translation* (ssdt) *scheme* is a system $Y(V_N, \Sigma, \Omega, P, S)$, where V_N (the nonterminal alphabet), Σ (the terminal alphabet) and S (the axiom) are as in a grammar; Ω is the output alphabet; $\Sigma \cap \Omega = \varnothing$ without loss of generality.

P is the set of productions of the form $B \to \alpha$, where $B \in V_N$, $\alpha \in (V_N \cup \Sigma \cup \Omega)^*$.

Let us introduce the two homomorphisms (projections)

$$h_I(a) = a \quad \text{if } a \in \Sigma \cup V_N, \qquad h_I(a) = \varepsilon \quad \text{if } a \in \Omega,$$

$$h_O(a) = \varepsilon \quad \text{if } a \in \Sigma, \qquad h_O(a) = a \quad \text{if } a \in \Omega \cup V_N$$

and extend them in the natural way to the domain $(\Sigma \cup \Omega \cup V_N)^*$.

The two grammars

$$G_I = (V_N, \Sigma, P', S), \qquad G_O = (V_N, \Sigma, P'', S)$$

where $P' = h_I(P)$, $P'' = h_O(P)$, are the *input* and the *output* grammars of Y. For a production $p \in P$, let $p' = h_I(p)$ and $p'' = h_O(p)$.

The translation τ_Y is the mapping $\Sigma^* \to \Omega^*$, defined by $\tau_Y(x) \triangleq y$, where

$$S \overset{p'_1}{\Rightarrow} \alpha_1 \overset{p'_2}{\Rightarrow} \alpha_2 \cdots \overset{p'_n}{\Rightarrow} \alpha_n = x,$$

$$S \overset{p''_1}{\Rightarrow} \beta_1 \overset{p''_2}{\Rightarrow} \beta_2 \cdots \overset{p''_n}{\Rightarrow} \beta_n = y,$$

that is the two derivations result from the same derivation with regard to P, under application of homomorphisms h_I and h_0. Notice that in this case τ_Y is only defined in $L(G_I)$, whereas for dpdt's τ is defined in Σ^*. This does not affect the generality of the following Definitions 3.2 and 3.5.

A ssdt Y is *postfix* (respectively *prefix*) iff for any production $B \to \alpha$ in P it is $\alpha \in (V_N \cup \Sigma)^*.\ \Omega^*$ (resp. $\alpha \in \Omega^* \cdot (V_N \cup \Sigma)^*$).

DEFINITION 3.2 (cascade composition of ssdt schemes).

1. A language L belongs to the family $\mathscr{TR}_k$ if there exist k ssdt schemes Y_j such that G_{I_j} is LR (i.e. is $LR(n)$ for some $n \geqq 0$), and $x \in L$ if and only if $\tau_{Y_{j-1}}(\cdots(\tau_{Y_1}(x))\cdots)$ is defined and belongs to $L(G_{I_j})$ for all $j \leqq k$.

2. The family $\mathscr{TL}_k$ is defined similarly to $\mathscr{TR}_k$, with the difference of G_{I_j} being $LL(n)$, $n \geqq 1$.

3. The family $\mathscr{TR}p_k$ is defined as $\mathscr{TR}_k$ by constraining Y_j to be postfix, for all $j \leqq k$.

4. $\mathscr{TR}$, $\mathscr{TL}$, $\mathscr{TR}p$ are resp. defined as $\bigcup_{k=1}^{\infty} \mathscr{TR}_k$, $\mathscr{TL}_k$, $\mathscr{TR}p_k$.

Let us now briefly comment on the relations between the above families and $\mathscr{CH}$.

First, since the theory of deterministic parsing and translation is developed assuming the presence of *endmarkers* (Aho and Ullman (1972-73)), we need the following technical definition.

DEFINITION 3.3. Let $\perp \notin \Sigma$. A language L is in $\mathscr{CH}\perp$ (resp. $\mathscr{CH}\perp_k$) iff $L \cdot \perp$ is in $\mathscr{CH}$ (resp. $\mathscr{CH}_k$).

It is well known that ssdt's are equivalent to pdt's as any translation defined by means of a ssdt can be obtained by a pdt and conversely (Aho and Ullman (1972-73)). The same fact in general does not hold in the case of deterministic parsing and translation. In fact if, for a given scheme Y, G_I is LR, it is not guaranteed that τ_Y can

be computed by a dpdt. On the other hand, if G_I is LL, then τ_Y can be computed by a dpdt, but $L(G_I)$ cannot be any deterministic language, since deterministic top-down languages are strictly included in the deterministic context-free ones.

However, dpdt's are equivalent to postfix ssdt schemes, what implies that $\mathscr{CH}\perp_k \equiv \mathscr{TRP}_k \; \forall \; k$.

On the other hand it is known (Aho and Ullman (1972-73, p. 736)) that for any ssdt scheme Y such that G_I is LR, there exists a chain of 4 dpdt's such that $\tau_4(\cdots \tau_1(x)\cdots) = \tau_Y(x)$ for all $x \in L(G_I)$. This leads to

STATEMENT 3.1. $\mathscr{CH}\perp \equiv \mathscr{TRP} \equiv \mathscr{TR}$.

This statement recasts, for the chains of dpdt's, the full equivalence with ssdt's existing in the nondeterministic case.

Instead, it is not known whether between $\mathscr{TL}$ and $\mathscr{CH}\perp$ equality or strict inclusion holds: i.e. can multi-pass top-down deterministic parsing reach the same power as in the bottom-up case? The answer is obviously negative for single pass parsing.

Let us now relate $\mathscr{CH}$ with the languages obtainable by means of control sets (Ginsburg and Spanier (1968)).

DEFINITION 3.4.

1. A *labeled grammar GL* is a triple $\langle G, \text{Lab}, \Lambda\rangle$, where
   1. G is a context-free grammar $G = \langle V_N, \Sigma, P, S\rangle$;
   2. Lab is a finite set of labels;
   3. Λ is a mapping Λ: $P \to \text{Lab} \cup \{\varepsilon\}$;

$\Lambda(p)$ is the label of production p. If $\Lambda(p) = \varepsilon$, the production is unlabeled. Notice that two productions may have the same label. The mapping Λ is naturally extended to P^*.

2. As in Aho and Ullman (1972-73), we denote by $\overset{\pi}{\Rightarrow}$ (resp. $\Rightarrow^{\pi}$) a leftmost (resp. rightmost) derivation, where π denotes the string of the elements of P composing the derivation.

3. $\pi l(x)$, $\pi r(x)$, $\pi r^R(x)$, $\pi l^R(x)$ are defined as:

$$\pi l(x) \triangleq \{y \in \text{Lab}^* \mid \text{there exists } \pi,\ S \overset{\pi}{\Rightarrow} x,\ y = \Lambda(\pi)\};$$

$$\pi r(x) \triangleq \{y \in \text{Lab}^* \mid \text{there exists } \pi,\ S \Rightarrow^{\pi} x,\ y = \Lambda(\pi)\};$$

$$\pi r^R(x) \triangleq \{y \in \text{Lab}^* \mid \text{there exists } \pi,\ S \Rightarrow^{\pi} x,\ y = \Lambda(\pi^R)\};$$

$$\pi l^R(x) \triangleq \{y \in \text{Lab}^* \mid \text{there exists } \pi,\ S \overset{\pi}{\Rightarrow} x,\ y = \Lambda(\pi^R)\}.$$

Let $\hat{\pi}(x)$ stand for any one of the four cases above. If G is unambiguous (and, a fortiori, if it is LR), then $\forall x \in \Sigma^*$, $\hat{\pi}(x)$ is either empty or a singleton.

Considering labeled grammars which are LR, the following families of *deterministically controlled languages* can be defined.

DEFINITION 3.5 (composition of controlled LR grammars). Let $\{GL_j, 1 \leqq j \leqq k\}$, be a set of labeled LR grammars with $\text{Lab}_1 = \Sigma_2$, $\text{Lab}_2 = \Sigma_3, \cdots, \text{Lab}_{k-1} = \Sigma_k$. The following scheme defines the language recognized by the deterministically controlled chain: $L \triangleq \{x \mid x \in L(G_1)$ and, $\forall j,\ 1 \leqq j \leqq k-1,\ \hat{\pi}_j(\cdots \hat{\pi}_1(x)\cdots) \in L(G_{j+1})\}$.

Four families of languages are obtained according to the choice of $\hat{\pi}$:

1. $\hat{\pi} = \pi l$; then $\mathscr{CL}_k \triangleq \{L\}$;
2. $\hat{\pi} = \pi r$; then $\mathscr{CR}_k \triangleq \{L\}$;
3. $\hat{\pi} = \pi r^R$; then $\mathscr{CR}R_k \triangleq \{L\}$;
4. $\hat{\pi} = \pi l^R$; then $\mathscr{CL}R_k \triangleq \{L\}$.

The families of controlled languages have been widely studied in the literature by using several slightly different definitions (Ginsburg and Spanier (1968), Salomaa (1973)). In general, the attention has been focused on the nondeterministic case—i.e.

relaxing the hypothesis that all G_j are *LR*. A notable exception is provided by Khabbaz (1974), where linear simple precedence grammars are considered.

For the *nondeterministic case* we denote by a prefix $\mathcal{N}$ the nondeterministic counterpart of the families of Definition 3.5 (thus in $\mathcal{NCL}_k$ the labeled grammars may be nondeterministic, etc.).

The main result in the nondeterministic case is that $\mathcal{NCL}_2 \equiv \mathcal{RE}$, the family of all recursively enumerable languages (Ginsburg and Spanier (1968)). Since it is immediate, for any labeled grammar, to build a pdt such that $\forall x$: $\tau(x) = \pi l(x)$, and conversely for any pdt to build a labeled grammar such that $\pi r(x) = \tau(x)$, we have $\mathcal{NCL}_2 \equiv \mathcal{NCH}_2 \equiv \mathcal{NCR}_2 \equiv \mathcal{RE}$.

Furthermore, since context-free languages are closed with respect to reversal operation, we have

STATEMENT 3.2. $\mathcal{NCH}_2 \equiv \mathcal{NCL}_2 \equiv \mathcal{NCR}_2 \equiv \mathcal{NCR}R_2 \equiv \mathcal{NCL}R_2 \equiv \mathcal{RE}$.

Notice that the statement $\mathcal{NCH}_2 \equiv \mathcal{RE}$ was rediscovered by Tadashi Ae (1977), who instead left open the question whether "the cascade product of two dpda's is equivalent to a Turing machine". The question will next receive a trivial answer.

For any labeled grammar $GL = \langle G, \text{Lab}, \Lambda \rangle$ such that G is deterministic it is immediate to build a postfix ssdt scheme Y such that $G_I = G$ and $\tau_Y(x) = \pi r^R(x) \forall x \in L(G)$. Hence $\mathcal{CH}\perp_k \equiv \mathcal{CR}R_k \equiv \mathcal{TR}_k \forall k$.

Furthermore, since the reversal $\tau(x\perp) = x^R\perp$ can be easily obtained by means of anyone of dpdt's, (postfix) ssdt's, or (regular) labeled grammars, we have $\mathcal{CH}\perp \equiv \mathcal{CR}R \equiv \mathcal{TR} \equiv \mathcal{CR}$.

Also, for any labeled *LR* grammar, a (prefix) ssdt scheme Y can be immediately obtained such that $G_I = G$ and $\tau_Y(x) = \pi l(x) \forall x \in L(G)$.

Conversely, the following statement holds.

STATEMENT 3.3. *Let Y be a postfix* ssdt *scheme such that G_I is LR. Then a prefix* ssdt *scheme Y' exists such that G'_I is equivalent to G_I and LR, and $\tau_{Y'}(x) = \tau_Y(x) \forall x \in L(G'_I)$.*

Proof. Let P and P' denote the productions of Y and Y' respectively. To construct P' replace any production i: $A \to \alpha\omega$, $\alpha \in (V_N \cup \Sigma)^*$, $\omega \in \Omega^*$ in P by the productions $A \to \alpha Q_i$, $Q_i \to \omega$, where Q_i are new nonterminals. Notice that

$$h_I(Q_i \to \omega) = Q_i \to \varepsilon, \qquad h_O(Q_i \to \omega) = Q_i \to \omega.$$

The equivalence of Y and the ssdt scheme Y' thus obtained is obvious. But G'_I is *LR* if G_I is so, since it is immediate to construct a canonical *LR* parser associated to G'_I, by a straightforward modification of the parser associated to G_I. Q.E.D.

On the other hand, it is immediate to obtain from a given prefix ssdt scheme Y, whose G_I is *LR*, a labeled *LR* grammar GL such that $\forall x \in L(G_I)$: $\tau_Y(x) = \pi l(x)$.

Therefore, since any language in $\mathcal{TR}$ is also in $\mathcal{TR}p$, it is in $\mathcal{CL}$ as well. In summary, we have proved the following statement.

STATEMENT 3.4. $\mathcal{CH}\perp \equiv \mathcal{TR} \equiv \mathcal{CL} \equiv \mathcal{CR} \equiv \mathcal{CR}R \equiv \mathcal{CL}R$.

Remark. With a little more insight it can also be realized that the same equalities continue to hold when in a labeled grammar the mapping Λ is bijective (isomorphism).

Notice the symmetry of the above statement, in spite of the fact that deterministic languages are not closed with respect to reversal, even when provided with an endmarker.

Notice also that the equivalence $\mathcal{CH}\perp \equiv \mathcal{CL}$ implies that a leftmost derivation of any deterministic language can be deterministically obtained, but not necessarily in a single pass. This is not to be confused with the question whether $\mathcal{TL} \equiv \mathcal{CH}\perp$.

4. Closure properties of $\mathscr{CH}$. In this section we investigate some closure properties of $\mathscr{CH}$ (and $\mathscr{CH}\perp$) with respect to usual language operations. The main result shows that both of them are Boolean algebras.

THEOREM 4.1. *$\mathscr{CH}$ and $\mathscr{CH}\perp$ are closed with respect to union.*

Proof. We sketch the rather simple construction for $\mathscr{CH}$. A simplified reasoning applies to $\mathscr{CH}\perp$.

Let L', $L''\subseteq\Sigma^*$ be resp. recognized by the chains $S'=\langle T'_1,\cdots,T'_k\rangle$, $S''=\langle T''_1,\cdots,T''_j\rangle$. First we derive $\bar{S}'$ from S' by modifying each dpdt T'_r of S' as follows.

Whenever T'_1 reads a character a_i, $\bar{T}_1$ emits $\bar{a}_i$ ($\notin\Sigma$), then it performs the same d-computation as T'_1. The following dpdt's $\bar{T}'_r$, $1<r\leqq k$, when reading $\bar{a}_i$, simply copy it on the output tape and, otherwise, compute as T'_r. The last dpda T'_k of S' is changed into a dpdt $\bar{T}'_k$ as follows. Whenever T'_k reads a_i, $\bar{T}'_k$ performs the same d-computation as T'_k, and finally it outputs 1 (resp. 0), if the state of T'_k is final (resp. nonfinal). Thus the output $\bar{x}$ of $\bar{S}'$ is in $(\bar{\Sigma}\cup\{0,1\})^*\cdot\{0,1\}$, and $h(\bar{x})=x$, where h is the homomorphism defined by $h(\bar{a}_i)=a_i$, $h(0)=h(1)=\varepsilon$.

Now build a modification $\bar{S}''$ of S'' such that each dpdt performs the same computation in $\bar{S}''$ as in S'', apart from copying 0's and 1's. The last dpda $\bar{T}''_j$ accepts if T''_j accepts, or if the input string ends by 1. Q.E.D.

Remark. In order to show the closure with respect to union and to intersection of $\mathscr{CH}\perp$, a simpler proof can be given as briefly sketched below.

1. First translate $x\perp$ to $x\perp x^R\perp$;
2. then translate this to $x\perp x\perp$ by reversing x^R;
3. finally check whether the first copy of $x\in L'$ or the second copy $\in L''$.

The strength of this approach resides in the fact that by means of $k+1$ dpdt's one can build 2^k copies of a string $x\perp$. Thus by letting $LG(k)$ be the integer j such that $2^{j-1}<k\leqq 2^j$, we have the following corollary.

COROLLARY 4.2. *Let $L=\bigcup_{j=1}^{k}L_j$, $L_j\in\mathscr{CH}\perp_{h_j}$. Then $L\in\mathscr{CH}\perp_r$, where $r=\max\{h_j\}+LG(k)+1$. (An analogous statement holds for $\cap$ too).*

THEOREM 4.3. *$\mathscr{CH}$ and $\mathscr{CH}\perp$ are closed with respect to the complement (to Σ^*).*

Proof. The proof is quite similar to the well-known proof of the closure of deterministic context-free languages with respect to complement, a proof that also relies on the loop-free form of dpda.

It just suffices to observe that for any k-chain, $\sigma_{k-1}(x)$ is defined $\forall x\in\Sigma^*$. For the last automaton just set $\bar{F}_k=Q_k-F_k$ to obtain the complement of the language accepted by $\langle T_1,\cdots,T_k\rangle$. Q.E.D.

Consequently both $\mathscr{CH}$ and $\mathscr{CH}\perp$ are Boolean algebras.

It is perhaps worth emphasizing that obtaining the same results directly on the equivalent classes $\mathscr{TR}$, $\mathscr{CL}$, $\mathscr{CR}$ etc. would have been quite less immediate.

The following trivial properties somewhat complete the framework of closure properties of $\mathscr{CH}$ and $\mathscr{CH}\perp$,

PROPOSITION 4.4. *The families $\mathscr{CH}$ and $\mathscr{CH}\perp$ are closed under τ^{-1}, where τ is a deterministic push-down translation.*

Vitànyi and Savitch (1978) have mainly investigated the inverse dpdt translation of Dyck sets, the latter obviously deterministic.

Their results thus concern a special case of $\mathscr{CH}_2$.

COROLLARY 4.5. *$\mathscr{CH}\perp$ is closed under reversal.*

PROPOSITION 4.6. *The families $\mathscr{CH}$ and $\mathscr{CH}\perp$ are closed with respect to the shuffle of languages with disjoint alphabets.*

Hint. Let $L'=L(S')$, $L''=L(S'')$ with disjoint alphabets Σ' and Σ''. Then a chain $S=\langle\hat{S}',S''\rangle$ accepting the shuffle of L' and L'' is obtained by modifying S' into $\hat{S}'$, so

that each $\hat{T}'_i \in \hat{S}'$, when encountering a symbol $a'' \in \Sigma''$, simply copies it on the output tape.

PROPOSITION 4.7. *The family $\mathscr{CH}$ is not closed under homomorphism.*

Proof. By a theorem of Ginsburg and Spanier (1968), every recursively enumerable language L is the homomorphic image of the intersection of two deterministic context-free languages: $L = h(L' \cap L'')$.

But $L' \cap L'' \in \mathscr{CH}$, this latter being a Boolean algebra. Hence we would obtain $\mathscr{RE} \subseteq \mathscr{CH}$, which is absurd (see also § 6). Q.E.D.

Thus $\mathscr{CH}$ and $\mathscr{CH}\perp$ are not AFL's, which is also implied by the fact that they are Boolean algebras and contain the language $\{a^n b^n \mid n \geqq 1\}$ (Ginsburg (1975)). Closure with respect to concatenation and $*$ is still open, but unlikely in our opinion, as it is often the case for Boolean algebras of languages, the only important exception being regular languages.

5. The $\mathscr{CH}$ hierarchy. The main result of this section is that both $\{\mathscr{CH}_i\}$ and $\{\mathscr{CH}\perp_i\}$ are a strict hierarchy with respect to i. This is an immediate consequence of the following theorem.

THEOREM 5.1. *The language $L = \{a^n b^{kn} \perp \mid n \geqq 0,\ k \in J\} \notin \mathscr{CH}_i$ if $|J| \geqq 2^{(2^{i+1}-2-i)}$.*

The proof is based on a series of technical lemmas which carefully study the behaviour of a dpdt, extending the results in the literature (Ginsburg and Greibach (1966), C. Kintala (1979)). Each lemma is preceded by an informal description. The formal statement follows, with a brief sketch of the proof. Technical details of the proofs are postponed to the Appendix.

Let us first recall that Lemma 4.1 by Ginsburg and Greibach (1966) studies the behaviour of a loop-free dpda reading a single character repeated arbitrarily many times. It states that either the length of the stack is bounded independently of the length of the input—and therefore the same contents periodically returns to the stack—or a loop is entered by which a fixed string is pushed onto the stack. Notice that the former case can be considered as a special case of the latter, by assuming that the string pushed onto the stack during the loop is null.

As a first generalization of this lemma, consider a dpdt in a configuration with a generic state and stack contents, and an input consisting of an arbitrary number of repetitions of a string x in Σ^+. Then the dpdt, after reading a bounded number of occurrences of x, ultimately enters a loop. At each loop iteration a fixed number of repetitions of x are read and consumed, a fixed string is pushed onto the stack, and a fixed output string is produced (both strings may be null).

This is formally stated in the following lemma.

LEMMA 5.2. *For any* dpdt, *and for any $q \in Q$, $x \in \Sigma^+$, $\alpha \in \Gamma^+$, there exist:*

—two integers $\bar{k}$, $\hat{k}$ (called the threshold and the period respectively);

—two strings $\bar{\alpha}$, β in Γ^*;

—two strings w, y in Ω^*, such that for any L, $0 \leqq L < \hat{k}$ there exist

—a state $q_L \in Q$,

—a string $\zeta_L \in \Gamma^*$,

—a string w_L (prefix of y), such that for every integer k of the type $k = \bar{k} + h\hat{k} + L$, the following derivation holds:

$$\langle q, x^k, \alpha; \varepsilon \rangle \vdash^{d*} \langle q_L, \varepsilon, \bar{\alpha}\beta^h \zeta_L; wy^h w_L \rangle.$$

The proof in the Appendix is a careful exploitation and extension of the proof of Lemma 4.1 by Ginsburg and Greibach (1966); as such it constitutes also a more accurate (and perhaps simpler) demonstration of their result. The core of the proof

splits the computation of the dpdt into three distinct phases, called opening transient, loop iteration, closing transient.

The opening transient consists of the derivation

$$\langle q, x^{\bar{k}}, \alpha; \varepsilon\rangle \vdash^{d*} \langle \bar{q}, \varepsilon, \bar{\alpha}\beta\bar{\zeta}; wy\rangle,$$

where $\bar{k}$, $\bar{q}$, $\bar{\alpha}$, β, $\bar{\zeta}$, w, y are effectively computed.

The loop consists of the derivation

$$\langle \bar{q}, x^{\hat{k}}, \bar{\alpha}\beta^i\bar{\zeta}; \varepsilon\rangle \vdash^{d*} \langle \bar{q}, \varepsilon, \bar{\alpha}\beta^{i+1}\bar{\zeta}; y\rangle, \qquad i \geqq 1$$

where $\hat{k}$ is the fixed number of repetitions of x consumed by each iteration of the loop.

The closing transient consists of

$$\langle \bar{q}, x^{L}, \bar{\alpha}\beta^i\bar{\zeta}; \varepsilon\rangle \vdash^{d*} \langle q_L, \varepsilon, \bar{\alpha}\beta^i\zeta_L; w_L\rangle.$$

Remember that these computations always exist because the automaton is loop-free.

In the following we will use the notations:

$$\begin{aligned}\bar{K} &= \{k \mid k \geqq \bar{k},\ k - \bar{k} \equiv L \pmod{\hat{k}}\} \\ &= \{k \mid \text{there exists an integer } h \geqq 0 \text{ such that } k = \bar{k} + h\hat{k} + L\}.\end{aligned}$$

The notation will also be extended to bidimensional and n-dimensional sets.

The function $h(k) \triangleq (k - \bar{k} - L)/\hat{k}$ for $k \in \bar{K}$ is *linear*. In general in the sequel the notation $h(k_1, k_2, \cdots, k_n)$ indicates that h is a linear function of arguments $k_1, \cdots, k_n$.

To further analyze the behaviour of dpdt's, consider the scanning of x^k, starting with a stack containing an arbitrary number of string repetitions. Precisely the starting configuration is of the type: $\langle q, x^k, \alpha_0\beta^i\alpha_1; \varepsilon\rangle$, shortly denoted as $C(k; i)$.

Several cases may occur: in the most complete one, the dpdt starts with an erasing phase where α_1 and a certain number of β's are popped while reading some (linearly related) number of x's, producing proportionally many occurrences of an output string. For k sufficiently large, all β's are consumed and the situation of Lemma 5.2 is reentered. Otherwise the dpdt halts in a configuration of the type $\langle q', \varepsilon, \alpha_0\beta^j\zeta; w_0 y^{h(k)} w_1\rangle$, with $j \leqq i$. Intuitively we shall say that a *comparison* has been performed between the exponents k and i of the periodically repeated input and stack strings x and β. In fact a linear function states how many x's are to be read in order to pop all β's from the stack. In the other cases some phase may be ineffective or missing.

This analysis is formalized by the next lemma.

LEMMA 5.3. *For any* dpdt, *consider a configuration of the type*

$$C(k; i) \triangleq \langle q, x^k, \alpha_0\beta^i\alpha_1; \varepsilon\rangle, \text{ with } |\beta| \geqq 1.$$

There exist then $\bar{k}$, $\hat{k}$, $\bar{\imath}$, $\hat{\imath}$ *such that* $\forall L$ *and* l, $0 \leqq L < \hat{k}$, $0 \leqq l < \hat{\imath}$, *one can effectively compute*:

1. *a linear function* $f(k; i)$ *and a constant* $\bar{f} \geqq 0$ *which partition the set*

$$\bar{X} = \{(k, i) \mid k \geqq \bar{k},\ k - \bar{k} \equiv L \pmod{\hat{k}},\ i \geqq \bar{\imath},\ i - \bar{\imath} \equiv l \pmod{\hat{\imath}}\}$$

into three disjoint subsets:

$$\begin{aligned}X_{(-1)} &= \{(k, i) \in \bar{X} \mid f(k; i) < 0\}, \\ X_{(0)} &= \{(k, i) \in \bar{X} \mid 0 \leqq f(k; i) \leqq \bar{f}\}, \\ X_{(+1)} &= \{(k, i) \in \bar{X} \mid f(k; i) > \bar{f}\};\end{aligned}$$

2. *a set of strings* $w_{0,p}, y_{1,p}, w_{1,p}, y_{2,p}, w_{2,p} \in \Omega^*$, *for* $p = -1, 0, 1$, *and of linear functions* $h_{1,p}(k, i)$, $h_{2,p}(k, i)$, *such that* $(k, i) \in X_{(p)}$ *and* $C(k; i) \vdash^{d*} \langle q_p, \varepsilon, y_p; u_p \rangle$ *imply*

$$u_p = w_{0,p} y_{1,p}^{h_{1,p}(k,i)} w_{1,p} y_{2,p}^{h_{2,p}(k,i)} w_{2,p}.$$

In particular

$$w_{1,-1} = y_{2,-1} = w_{2,-1} = \varepsilon, \qquad y_{2,0} = w_{2,0} = \varepsilon.$$

To clarify, Fig. 1 shows the partition of the lattice $\bar{X}$.

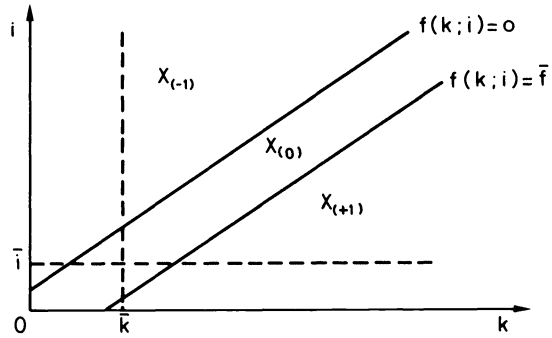


FIG. 1. *The partition of* $\bar{X}$.

We call $X_{(0)}$ a *strip*, $X_{(+1)}$, $X_{(-1)}$ *angular domains*. In the region $X_{(-1)}$, f is <0, i.e. there are not enough x's to completely pop β's from the stack. In $X_{(0)}$ β's are completely popped, but a new loop is not entered. In $X_{(+1)}$, k is large enough, with respect to i, to enter a new loop, as in Lemma 5.2.

Notice that, as a special case, the line $f(k; i) = 0$ may be parallel to the i-axis. This means that either all β's are popped while scanning only a bounded number of x's (through ε-moves), or that β's are not popped (but in a bounded number). In this case $X_{(-1)}$ and $X_{(0)}$ are empty, and no comparison of exponents has been performed. The figure relates in fact to the most complex case.

As a further generalization, we consider now a starting configuration where the stack contains a number s of periodically repeated strings β_i. To illustrate with a simple case, where the β_i's and the input period are single characters, consider a dpda in the configuration

$$\langle q, a^k, Z_0 B^{i_1} C^{i_2} \rangle,$$

and suppose that δ is defined by:

$$\delta(q, a, C) = \delta(q, a, B) = \langle q, \varepsilon \rangle,$$

$$\delta(q, a, Z_0) = \langle q_1, Z_0 \rangle,$$

$$\delta(q_1, a, Z_0) = \langle q, Z_0 A \rangle,$$

$$\delta(q, a, A) = \langle q, AA \rangle.$$

It is immediate to realize that the following configurations are respectively reached:
—if $k < i_2$: $\langle q, \varepsilon, Z_0 B^{i_1} C^{i_2 - k} \rangle$;
—if $k \geqq i_2$, $k - i_2 < i_1$: $\langle q, \varepsilon, Z_0 B^{i_1 - (k - i_2)} \rangle$;
—if $k = i_2 + i_1$: $\langle q_1, \varepsilon, Z_0 \rangle$;
—if $k > i_2 + i_1$: $\langle q, \varepsilon, Z_0 A^{k - (i_2 + i_1) - 1} \rangle$.

Notice that the comparisons between exponents are expressed by linear inequalities involving k and the i_j.

This is formally and generally stated in the following lemma.

LEMMA 5.4. *Consider a configuration of the type*

$$C(k; i_1, \cdots, i_s) \triangleq \langle q, x^k, \alpha_0 \beta_1^{i_1} \alpha_1 \cdots \beta_s^{i_s} \alpha_s; \varepsilon \rangle, \qquad |\beta_r| \geqq 1, s \geqq 1.$$

There exist then $\bar{k}$, $\hat{k}$, $\bar{\imath}$, $\hat{\imath}$ *such that* $\forall L$, l_r: $0 \leqq L < \hat{k}$, $0 \leqq l_r < \hat{\imath}$ $(1 \leqq r \leqq s)$; *one can effectively compute an integer* c, *with* $0 \leqq c \leqq s$, *and*:

1. $c \leqq s$ *constants* $\{\bar{f}_t, s-c+1 \leqq t \leqq s\}$ *and* c *linear functions* $\{f_t(k; i_s, i_{s-1}, \cdots, i_t) \,|\, s-c+1 \leqq t \leqq s\}$, *which partition the set*

$$\bar{X} = \{(k, i_1, \cdots, i_s) \,|\, k \geqq \bar{k},\ k - \bar{k} \equiv L \pmod{\hat{k}};\ i_r \geqq \bar{\imath},\ i_r \equiv l_r \pmod{\hat{\imath}},\ 1 \leqq r \leqq s\}$$

into $2c+1$ *disjoint subsets* $X_{(p)}$, $p = +1, 0, -1, \cdots, -2c+1$:

$$X_{(-2j+1)} = \{(k, i_1, \cdots, i_s) \in \bar{X} \,|\, f_{s-c+j} < 0, f_{s-c+j+1} > \bar{f}_{s-c+j+1}\}$$

with $1 \leqq j \leqq c$ (*if* $j = c$ *the last inequality is missing*),

$$X_{(-2j+2)} = \{(k, i_1, \cdots, i_s) \in \bar{X} \,|\, 0 \leqq f_{s-c+j} \leqq \bar{f}_{s-c+j}\},$$

$$X_{(+1)} = \{(k, i_1, \cdots, i_s) \in \bar{X} \,|\, f_{s-c+1} > \bar{f}_{s-c+1}\}.$$

2. $(2c+3)(2c+1)$ *strings*

$$w_{0,p}, y_{1,p}, w_{1,p}, \cdots, y_{c+1,p}, w_{c+1,p} \in \Omega^*,$$

$(c+1)(2c+1)$ *linear functions*

$$h_{r,p}(k, i_s, i_{s-1}, \cdots, i_{s-r+1}), \quad -2c+1 \leqq p \leqq 1, \quad 1 \leqq r \leqq c+1,$$

(*actually* $h_{c+1,p}$ *does not depend on* i_{s-c}), *such that*

$$(k, i_1, \cdots, i_s) \in X_{(p)} \quad \textit{and} \quad C(k; i_1, \cdots, i_s) \vdash^{d*} \langle q_p, \varepsilon, \gamma_p; u_p \rangle$$

imply

$$u_p = w_{0,p} y_{1,p}^{h_{1,p}(k, i_s)} w_{1,p} y_{2,p}^{h_{2,p}(k, i_s, i_{s-1})} \cdots y_{c+1,p}^{h_{c+1,p}(k, i_s, \cdots, i_{s-c+1})} w_{c+1,p}.$$

Similar to Lemma 5.3, some $w_{i,p}$, $y_{i,p}$ may vanish.

When p is even, $X_{(p)}$ is called a *strip*; when p is odd $X_{(p)}$ is called an *angular domain.* Intuitively the integer c is the maximum number of comparisons between the exponents k and i_j, that are actually performed when k is "large".

The proof consists of a repetitive application of Lemma 5.3 to each group $\beta_r^{i_r}$, starting with $r = s$. At some step, if k is large enough, the situation of Lemma 5.2 is entered. In order to make the notation not too cumbersome, a single congruence with modulus $\hat{\imath}$ is obtained by considering the l.c.m. of the moduli of the congruences determined by each subcase.

The next lemma generalizes Lemma 5.2 by allowing a number n of periodically repeated strings in the input. Although the initial stack contains only Z_0, as a result of iterated pushes, the stack can reach the configuration that was considered as the starting one in Lemma 5.4. Therefore in the most complex case, this lemma has to analyze the behaviour of a dpdt from a configuration which has a number of periodically repeated strings in the input *and* on the stack. The result is that the output produced contains a number of periodically repeated strings: this number does not exceed the double of n, and the number of repetitions of each string is a linear function of the numbers of repetition of the strings in the input.

LEMMA 5.5. *Consider an initial configuration*

$$C_0(k_1, \cdots, k_n) \triangleq \langle q_0, u_0 x_1^{k_1} u_1 \cdots x_n^{k_n} u_n, Z_0; \varepsilon \rangle.$$

One can effectively compute $\bar{k}$, $\hat{k}$ *such that,* $\forall L_j$ $(0 \leqq L_j < \hat{k})$ *the set* $\bar{X} = \{k_j \geqq \bar{k}, k_j - \bar{k} \equiv L_j \pmod{\hat{k}}, 1 \leqq j \leqq n\}$ *is partitioned into finitely many parts* $X_{(p)}$. *Each* $X_{(p)}$ *is defined by at most* $n-1$ *relations of the same type as in Lemma* 5.4. *Furthermore* $(k_1, \cdots, k_n) \in X_{(p)}$ *and* $C_0(k_1, \cdots, k_n) \vdash^{d*} \langle q_p, \varepsilon, \gamma_p; \tau_p \rangle$ *imply*:

1. $$\tau_p = w_{0,p} y_{1,p}^{h_{1,p}} \cdots y_{q,p}^{h_{q,p}} w_{q,p}, \qquad q \leqq 2n,$$

where $w_{i,p}$, $y_{i,p}$, $h_{i,p}$ *are as in Lemma* 5.4.

2. $$\gamma_p = \alpha_{0,p} \beta_{1,p}^{g_{1,p}(k_1, \cdots, k_n)} \alpha_{1,p} \cdots \beta_{r,p}^{g_{r,p}(k_1, \cdots, k_n)} \alpha_{r,p},$$

where $g_{i,p}$ *are linear functions and* $r \leqq \min\{n, 2n-q\}$.

The proof is an inductive application of the previous lemma.

The previous lemmas are next applied to analyze the behaviour of an i-chain of dpdt's, with input of the type $a^n b^m \perp$. The first output can thus contain at most four periodically repeated strings, and the numbers of repetitions are linear functions of n and m. It results that after i translations the output contains a number of periodically repeated strings that is bounded by an exponential function of i, and that each number of repetition is a linear function of n and m. The actual linear functions involved depend on the equivalence classes to which (n, m) belongs, with respect to a set of congruences. The number of different sets of functions can be bounded by an hyper-exponential function of i.

LEMMA 5.6. *Let* $S = \langle T_1, \cdots, T_i \rangle$ *be a chain of* dpdt's *with input of the type* $x = a^n b^m \perp$. *One can effectively compute* $\bar{n}$, $\hat{n}$, $\bar{m}$, $\hat{m}$ *such that* $\forall l_n < \hat{n}$, $l_m < \hat{m}$ *the set*

$$\bar{X} = \{(n, m) \mid n \geqq \bar{n},\ n - \bar{n} \equiv l_n \pmod{\hat{n}},\ m \geqq \bar{m},\ m - \bar{m} \equiv l_m \pmod{\hat{m}}\}$$

is partitioned into $s \leqq a_i = 2^{(2^{i+1}-2-i)}$ *angular domains and* $s-1$ *strips, both denoted as* $X_{(p)}$, *such that* $(n, m) \in X_{(p)}$ *implies*

$$\sigma(x) = w_{0,p} y_{1,p}^{h_{1,p}} \cdots y_{q,p}^{h_{q,p}} w_{q,p}, \qquad q \leqq 2^{i+1},$$

where $h_{r,p}$ *are linear functions of* n, m.

We are finally ready to give

Proof of Theorem 5.1. For any k in J, let

$$L_k = \{a^n b^{kn} \perp \mid n \geqq 0\} \quad \text{and} \quad P_k = \{n, kn) \mid a^n b^{kn} \perp \in L_k\},$$

i.e. the Parikh mapping of L_k. Assume by contradiction that L is recognized by an i-chain. Compute $\bar{n}$, $\bar{m}$, $\hat{n}$, $\hat{m}$ as in Lemma 5.6. Let $h = \text{l.c.m.}\{\hat{m}, \hat{n}\}$, and choose any $l_n < h$, $l_m < h$. By Lemma 5.6 the partition of the corresponding $\bar{X}$ has at most $s-1$ strips.

Clearly, for any strip $X_{(p)}$ there exists at most one $k \in J$ (i.e. a straight line) such that $P_k \cap X_{(p)}$ is unbounded. Thus, since $|J| \geqq a_i$, there exists at least one angular domain $X_{(p)}$ such that $P_k \cap X_{(p)}$ is unbounded (that is at least one straight line is ultimately contained in $X_{(p)}$).

Let $X_{(p)}$ be defined by a system of congruences and of inequalities of the types:

$$n \equiv l_n \pmod{h}, \qquad m \equiv l_m \pmod{h},$$

and

$$n \geqq \bar{n}, \quad m \geqq \bar{m}, \quad a'n + b' < m < a''n + b'', \quad 0 < a' < a'', \quad a' \leqq k \leqq a''.$$

By a proper choice of $\bar{m}$ and $\bar{n}$, it can be assumed (without loss of generality) that

$(\bar{n}, \bar{m}) \in P_k \cap X_{(p)}$, therefore:

$$\{(n, m) \mid n = \bar{n} + rh,\ m = \bar{m} + rhk = kn,\ r \geqq 0\} \subseteq P_k \cap X_{(p)}.$$

Assume now $k < a''$ (if $k = a''$, then $k > a'$ and proceed in a similar way). Then the point

$$\tilde{P} = (n, \tilde{m}), \text{ with } n = \bar{n} + rh,\ \tilde{m} = \bar{m} + rhk + h = kn + h$$

belongs to $X_{(p)}$ provided that $kn + h = \tilde{m} < a''n + b''$, i.e.

$$h < (a'' - k)n + b'' = (a'' - k)\bar{n} + (a'' - k)rh + b''.$$

Since $(a'' - k)\bar{n} + b'' > 0$, it suffices to choose $r > 1/(a'' - k)$ in order that $\tilde{P} \in X_{(p)}$ (see Fig. 2).

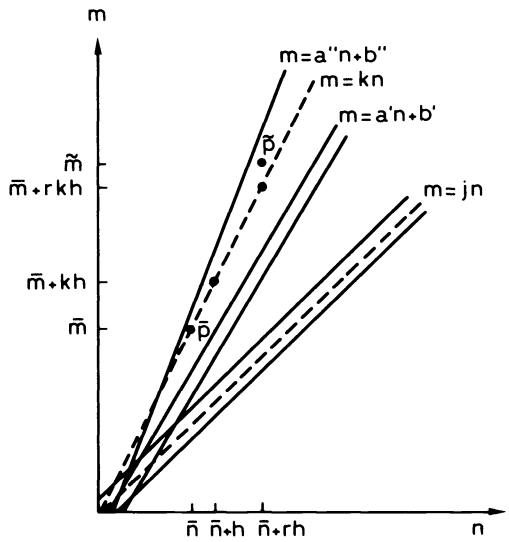


FIG. 2. *Proof of Theorem* 5.1.

But $a^n b^{\tilde{m}} \perp \notin L$ if $r \geqq 1$, because $\tilde{m}/n = k + h/n < k + 1$. On the other hand $\forall (n, m) \in X_{(p)}$, $a^n b^m \perp$ is accepted by the chain and this proves the contradiction. Q.E.D.

Finally we observe that Lemma 5.6 proves that the *translations* which can be computed by an i-chain are an infinite strict hierarchy.

6. Relations to other families of languages. In this section we investigate the relations between $\mathscr{CH}$, $\mathscr{CH}\perp$ and other relevant families of languages. Since the former are Boolean algebras, we focus attention on families sharing the same property. Precisely we consider:

—the family $\mathscr{R}$ of regular languages;

—the family $\mathscr{RT}$ of real-time languages (Rosenberg (1967)) i.e. those languages recognized by some deterministic multitape Turing machine in a number of steps which equals the length of the input string;

—the family $\mathscr{BC}$, consisting of the Boolean closure of deterministic context-free languages;

—the family $\mathscr{LT}$ of linear-time languages, i.e. those languages recognized by some deterministic multitape Turing machine in a number of steps which is bounded by $K|x|$, where K is a fixed integer and x is the input string;

—the family $\mathscr{DCS}$ of deterministic context-sensitive languages, i.e. those languages recognized by a deterministic single-tape Turing machine by using a number of cells of the tape bounded by $K|x|$.

Figure 3 displays the relations between the above families of languages.

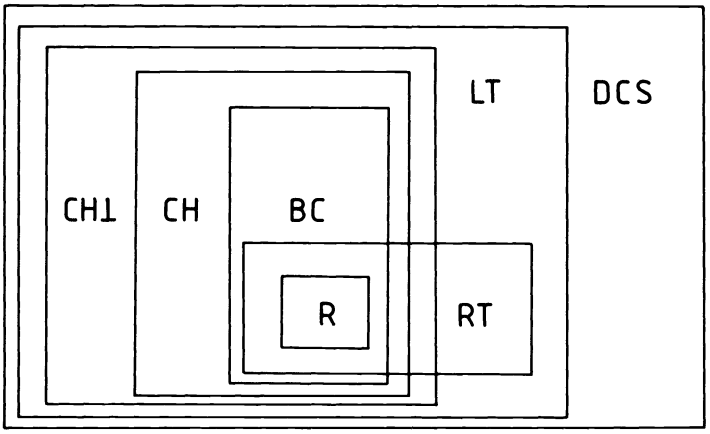


FIG. 3. *Relations between families of languages.*

We now briefly sketch the proofs of the nontrivial relations displayed in Fig. 3.

1. $\mathscr{BC} \subset \mathscr{RT}$. The language

$$L = \{0^{n_1}10^{n_2}1 \cdots 0^{n_k}12^r3^{n_{k-r+1}} \mid n_i \geqq 1, 1 \leqq r \leqq k\}$$

is deterministic context-free but not real-time (Rosenberg 1967)).

2. $\mathscr{BC} \subset \mathscr{CH}$. The language $L = \{ucu \mid u \in \{a, b\}^*\}$ is in $\mathscr{CH}_2$ (see Example 2.4) but not in $\mathscr{BC}$ (Wotschke (1978)).

3. $\mathscr{CH} \subseteq \mathscr{CH}\perp$ is obvious. We conjecture the inclusion to be strict.

4. $\mathscr{RT} \not\subseteq \mathscr{CH}\perp$. The language $L = \{a^m b^{km} \perp \mid m \geqq 0, k \geqq 0\}$ is in $\mathscr{RT}$ but not in $\mathscr{CH}$.

L is recognized in real-time by a Turing machine with read-only input and two other tapes. The machine operates as follows. After reading a^m, tape 1 is $\bar{a}a^{m-1}$ and tape 2 is empty. As the machine reads b's, it moves leftwards on tape 1 erasing a's, and moves rightwards on tape 2 writing $\bar{b}\bar{b}\cdots$, until it hits either $\bar{a}$ or $\perp$ (on input). If head 1 is on $\bar{a}$, the motion on the two tapes is reversed; the machine moves rightwards writing a's on tape 1 and moves leftwards on tape 2 erasing b's, until it hits $\bar{b}$ or $\perp$. When the input is $\perp$, the machine accepts, provided head 1 is on $\bar{a}$ or head 2 is on $\bar{b}$.

$L \notin \mathscr{CH}$ is a corollary of Theorem 5.1. In fact, suppose by contradiction L is recognized by some i-chain. Since for any J defined as in Theorem 5.1 it is

$$L' = \{a^n b^{kn} \perp \mid n \geqq 0, k \in J\} \subseteq L,$$

we can find a pair $(n, \tilde{m})$ as in the proof of Theorem 5.1, which is accepted by the chain but is not in L.

5. $\mathscr{CH}\perp \subset \mathscr{LT}$. The inclusion is obvious as an essential property of our class. The inclusion is strict because of relation 4 above.

Notice that every i-chain $\langle T_1, \cdots, T_i \rangle$ can be simulated in linear time by a deterministic Turing machine provided with 3 tapes in the following way. First the input of T_1 is on tape 1, while tape 2 is used to simulate the stack of T_1 and tape 3 stores the output of T_1. When T_1 has been completely simulated by the Turing machine, $\tau_1(x)$ is on tape 3. Since $|\tau_1(x)| \leqq K_1|x|$, the contents of tape 3 can be copied to tape 1 by linearly many moves. Then the 3 heads can be reset by a linear number of steps, and simulation of T_2 begins, etc.

Thus $\mathscr{CH}\perp$ is contained in the family $\mathscr{LT}_3$ of languages recognized by deterministic 3-tape (plus one read-only tape) Turing machines.

6. $\mathscr{LT} \subseteq \mathscr{DCS}$ is well known (Hopcroft and Ullman (1969)).

Besides the above relations it would be interesting to investigate whether $\mathscr{CH}$ (or $\mathscr{CH}\perp$) contains all context-free languages. We conjecture a negative answer for the following reasons:

1. It is likely that there exist nonlinear time recognizable context-free languages.
2. Although $L=\{ww^R\perp\}$ is in $\mathscr{CH}$, and therefore $L'=\{ww^R\}$ is in $\mathscr{CH}\perp$, we suspect that L' is not in $\mathscr{CH}$. This would only imply {context-free languages} $\not\subseteq \mathscr{CH}$.
3. We are unable to find a chain recognizing $L'L'$.

7. Conclusion. We have proposed a formal model of multi-pass translation, a frequently used technique for compiler writing, and investigated its basic properties. The model extends the well-known syntax-directed deterministic translation, towards more powerful, yet linear-time, translation schemes.

In this paper we have attempted to investigate the basic properties of chains (or cascades) of deterministic push-down transducers. From a generative viewpoint, languages accepted by these devices are between the deterministic context-free and the languages recognized in linear time by deterministic multi-tape Turing machines.

The family $\mathscr{CH}$ of multi-pass deterministic languages has several interesting closure properties, notably with respect to Boolean operators and reverse homomorphism.

An important result proved in this paper is that the family $\mathscr{CH}_k$ is a strict hierarchy ordered by k. The rather complex proof extends the classical analysis by Ginsburg and Greibach (1966) of the behaviour of a dpda with input a^n to the case of a chain with input a^nb^m.

Several interesting corollaries derive from this result. Our hierarchy theorem[1] generalizes previous results (by Liu and Weiner (1973), Wotschke and Kintala (1979)) related to the class $\mathscr{BC}$ of Boolean closure of deterministic context-free languages.

Hierarchy results have also appeared for tree-transducers (Engelfriet (1982)). However such formalism is suitable for language generation, not for recognition.

Several problems are still open. Perhaps the most interesting one is whether $\mathscr{TL} \subset \mathscr{CH}\perp$, i.e. whether multi-pass deterministic top-down parsing is less powerful than bottom-up, as it happens for single pass parsing.

The same problem can also be considered for different restricted models of dpda's, such as those recognizing by empty stack, by single elements in the stack, and so on (see Harrison (1978) for a systematic description of such models).

Another problem which is perhaps worth mentioning is to improve the present bounds for the power of i-chains: on the basis of Corollary 4.2 an i-chain can recognize[2]

$$L=\{a^nb^{kn}\perp \mid n\geqq 1, 1\leqq k\leqq 2^i\},$$

but cannot recognize

$$L'=\{a^nb^{kn}\perp \mid n\geqq 1, 1\leqq k\leqq 2^j\}, \qquad j=2^{i+1}-2-i.$$

What can an i-chain do at most in this respect?[3]

---

[1] After preparation of this paper, we learned of an independent claim of the hierarchy theorem by Platek (1983), but we are unaware of a complete proof of such a result in the open literature.

[2] With a little more insight the endmarker can be removed.

[3] We recently received a communication by Yehudai (1984), where it is shown that

$$L_j=\{a^nb^{kn}\perp \mid k\in J, |J|\leqq 2^{(2^{i-1}-1)}\}$$

can be recognized by an i-chain. His proof is based on the idea of performing a binary search.

8. Appendix.

Proof of Lemma 5.2. For any $r>0$ consider the derivation

(1) $\langle q, x^r, \alpha\rangle \vdash^{d*} \langle \hat{q}_1, x^{r-1}, \hat{\gamma}_1\rangle \vdash^{d*} \cdots \vdash^{d*} \langle \hat{q}_r, \varepsilon, \hat{\gamma}_r\rangle$.

Define γ_r as the string in Γ^* such that $\hat{\gamma}_r = \gamma_r\zeta_r$, and

(i) $\forall s>0$, $\forall x'$ suffix of x:

$$\langle \hat{q}_r, x^s, \hat{\gamma}_r\rangle \vdash^* \langle q', x', \gamma_r\zeta_r'\rangle, \quad \text{with } \zeta_r' \in \Gamma^+$$

(i.e. γ_r is the stack portion never affected when reading any number of x's, after the scanning of x^r),

(ii) for no $\tilde{\gamma}_r$ properly containing γ_r, such that $\hat{\gamma}_r = \tilde{\gamma}_r\tilde{\zeta}_r$, (i) still holds.

A decomposition $\hat{\gamma}_r = \gamma_r\zeta_r$ meeting (i) is called *admissible*, while a decomposition meeting (i) and (ii) is called *maximal*. Obviously $\forall r$ the maximal decomposition exists and is unique. Notice also that $|\zeta_r| \geqq 1$.

For any $r>0$, because of the maximality of γ_r, there exists $s_r>0$ such that computation (1) can be continued as

(2) $$\langle q, x^{r+s_r}, \alpha\rangle \vdash^{d*} \langle \hat{q}_{r+s_r-1}, x, \gamma_r\tilde{\zeta}\rangle \vdash^* \langle q', x', \gamma_r Z'\rangle \vdash \langle q'', x'', \gamma_r\zeta''\rangle \vdash^{d*} \langle \hat{q}_{r+s_r}, \varepsilon, \gamma_r\hat{\zeta}_r\rangle,$$

for some x' suffix of x, x'' suffix of x' and $Z' \in \Gamma$, ζ'', $\hat{\zeta} \in \Gamma^+$.

Next we show that the length of $\hat{\zeta}_r$ is bounded. Since the dpdt is loop-free, there exists $N>0$ such that, for any suffix x' of x, $\forall q' \in Q$, $\forall Z' \in \Gamma$, if

$$\langle q', x', Z'\rangle \vdash^{d*} \langle q'', \varepsilon, \zeta\rangle$$

then $|\zeta| \leqq N$. Thus in derivation (2) $|\hat{\zeta}_r| \leqq N$.

Next we show that with reference to derivation (2) a sequence $\{k_t, q_t, \eta_t, \theta_t\}_{t>0}$ can be sampled where, $\forall t$, k_t is an integer, with $k_t < k_{t+1}$, $q_t \in Q$, η_t, $\theta_t \in \Gamma^*$, $|\theta_t| \leqq N$, such that:

(iii) $$\langle q, x^{k_t}, \alpha\rangle \vdash^{d*} \langle q_t, \varepsilon, \eta_t\theta_t\rangle,$$

and, for any $s>0$, x' suffix of x:

(iv) $$\langle q_t, x^s, \eta_t\theta_t\rangle \vdash^* \langle q', x', \eta_t\theta'\rangle \quad \text{with } |\theta'| \geqq 1.$$

Such a sequence can be inductively built as follows. For $t=1$ let $\gamma_1\zeta_1$ be the maximal decomposition of $\hat{\gamma}_1$ (of derivation 1)), and s_1 be such that (2) holds for $r=1$. Then assume

$$k_1 = 1+s_1, \quad q_1 = \hat{q}_{1+s_1}, \quad \eta_1 = \gamma_1, \quad \theta_1 = \hat{\zeta}_1.$$

For $t=i+1$, $i>0$, assume $r=k_i$, $\gamma_r\zeta_r$ the maximal decomposition of $\hat{\gamma}_r$, $s_r>0$ such that (2) holds. Then set

$$k_{i+1} = r+s_r, \quad q_{i+1} = \hat{q}_{r+s_r}, \quad \eta_{i+1} = \gamma_r, \quad \theta_{i+1} = \hat{\zeta}_r.$$

It is immediate to realize that the sequence built in this way has the required properties (iii), (iv). For the finiteness of Q and Γ and from the fact that $|\theta_t| \leqq N \forall t$, there exist r and s, with $r<s$, such that $q_r = q_s$, $\theta_r = \theta_s$, $\eta_s = \eta_r\beta$ (with $\beta \in \Gamma^*$), because the decomposition $\eta_t\theta_t$ is admissible for any k_t.

Thus we have

$$\langle q, x^{k_s}, \alpha\rangle \vdash^{d*} \langle q_r, x^{k_s-k_r}, \eta_r\theta_r\rangle \vdash^{d*} \langle q_r, \varepsilon, \eta_r\beta\theta_r\rangle.$$

Notice that, once the existence of such q_r, k_r, k_s, η_r, β and θ_r is stated, they can be effectively computed by running the dpdt and testing for the fulfillment of the conditions.

Finally the lemma follows by letting $\bar{q} = q_r$, $\bar{\alpha} = \eta_r$, $\bar{\zeta} = \theta_r$, $\bar{k} = k_r$, $\hat{k} = k_s - k_r$ and by defining w, y, q_L, ζ_L, w_L via:

$$\langle q, x^{\bar{k}}, \alpha; \varepsilon\rangle \vdash^{d*} \langle \bar{q}, \varepsilon, \bar{\alpha}\bar{\zeta}; w\rangle,$$

$$\langle \bar{q}, x^{\hat{k}}, \bar{\zeta}; \varepsilon\rangle \vdash^{d*} \langle \bar{q}, \varepsilon, \beta\bar{\zeta}; y\rangle,$$

$$\langle \bar{q}, x^{L}, \bar{\zeta}; \varepsilon\rangle \vdash^{d*} \langle q_L, \varepsilon, \zeta_L; w_L\rangle. \qquad \text{Q.E.D.}$$

Remark. The previous proof overcomes a technical inaccuracy of Ginsburg and Greibach (1966, Lemma 4.1), where it is claimed that either

(i) there exists $n \geqq 1$ such that for every $m \geqq 1$ if

$$\langle q_0, a^m, Z_0\rangle \vdash^* \langle q, \varepsilon, \gamma\rangle \quad \text{then } |\gamma| \leqq n, \text{ or}$$

(ii) there exist integers m, f, words w, $y \in \Gamma^*$, $Z \in \Gamma$, $q \in Q$ such that, $\forall h \geqq 0$:

(a) $\langle q_0, a^{m+hf}, Z_0\rangle \vdash^{d*} \langle q, \varepsilon, wy^hZ\rangle$,

(b) $\langle q, a^k, wy^hZ\rangle \vdash^* \langle q', \varepsilon, \gamma\rangle$ implies $\gamma = wy^h\gamma'$, $\gamma' \neq \varepsilon$. In fact the following loop-free dpdt does not meet the above claim.

$$\delta(q_0, a, Z_0) = \langle q_1, Z_0AA\rangle,$$

$$\delta(q_1, a, A) = \langle q_2, AAZZ\rangle,$$

$$\delta(q_2, a, Z) = \langle q_3, \varepsilon\rangle,$$

$$\delta(q_3, \varepsilon, Z) = \langle q_4, \varepsilon\rangle,$$

$$\delta(q_4, \varepsilon, A) = \langle q_2, AAZZ\rangle.$$

Proof of Lemma 5.3. Let us analyze the behaviour of the stack γ of a configuration $C(k; i)$ while reading x^k. First we prove that there exist two alternatives:

(A) there exists i' such that $\forall i \geqq i'$, $k \geqq 1$ and $\forall$ configuration $\bar{C} = \langle \bar{q}, \bar{x}, \bar{\gamma}\rangle$ reachable from $C(k; i)$ (i.e. such that $C \vdash^* \bar{C}$) it is $\bar{\gamma} = \alpha_0\beta^{i-i'}\zeta$, $\zeta \in \Gamma^*$.

(B) $\forall i$ there exist $k = k(i)$ and a configuration $\bar{C} = \langle \bar{q}, \bar{x}, \bar{\gamma}\rangle$ reachable from $C(k; i)$ such that, for no ζ, $\gamma = \alpha_0\beta\zeta$.

Intuitively either (A) a bounded number of β's are popped, independently of k, or (B) for k sufficiently large all β's are erased.

Now we show that proposition (B) is implied by the negation of (A), namely: $\forall i'$ there exist: $i \geqq i'$, $k \geqq 1$ and a configuration $\bar{C} = \langle \bar{q}, \bar{x}, \bar{\gamma}\rangle$ reachable from $C(k; i)$ such that, for no ζ: $\bar{\gamma} = \alpha_0\beta^{i-i'}\zeta$. Consider the computation

$$C(k; i) \vdash^* C' = \langle q', x', \gamma'\rangle \vdash C'' = \langle q'', x'', \gamma''\rangle \vdash^* \bar{C},$$

where C'' is the first configuration such that, for no ζ, $\gamma'' = \alpha_0\beta^{i-i'}\zeta$. By necessity:

$$\gamma'' = \alpha_0\beta^{i-i'-1}\eta \quad \text{and} \quad \gamma' = \alpha_0\beta^{i-i'}.$$

Therefore we have

$$\langle q, x^k, \alpha_0\beta^i\alpha_1\rangle \vdash^* \langle q', x', \alpha_0\beta^{i-i'}\rangle \vdash \langle q'', x'', \alpha_0\beta^{i-i'-1}\eta\rangle,$$

and consequently by letting $i = i' + 1$:

$$C(k; i) = \langle q, x^k, \alpha_0\beta^{i'+1}\alpha_1\rangle \vdash^* \langle q', x', \alpha_0\beta\rangle \vdash \langle q'', x'', \alpha_0\eta\rangle,$$

which is equivalent to (B), since for no ζ, $\eta = \beta\zeta$.

Hence tertium non datur and we can separately consider cases (A) and (B).

CASE (A). Here we can apply Lemma 5.2, with $\beta^{i'}\alpha_1$ instead of α, since, by hypothesis (A), the portion $\alpha_0\beta^{i-i'}$ of the stack remains unchanged. The thesis is thus satisfied with the following values:

$\bar{\imath} = i'$, $\hat{\imath} = 1$;
$l = 0$ (obviously condition $i \equiv 0 \pmod 1$ is not a constraint);
$\bar{k}$, $\hat{k}$ are derived in the same way as in Lemma 5.2;
$\bar{f} = 0$, $f(k; i) = k+1$.

In particular:

$X_{(-1)} = X_{(0)} = \varnothing$, and $X_{(+1)} = \bar{X}$, i.e. $p = 1$;
w_0, y_1, w_1 are computed as in Lemma 5.2;
$y_2 = w_2 = \varepsilon$;
$h_1 = h(k) = (k - \bar{k} - L)/\hat{k}$ as in Lemma 5.2.

CASE (B). Denote by $K(i)$ the minimal k for which (B) holds. First we show that $K(i)$ is nondecreasing. By contradiction suppose that $k_2 = K(i_2) < k_1 = K(i_1)$, $i_1 < i_2$. From (B) and by the definition of K we have:

$$C(k_2; i_2) = \langle q, x^{k_2}, \alpha_0\beta^{i_2}\alpha_1\rangle \vdash^* C'' = \langle q'', x'', \gamma''\rangle,$$

with $x = x'x''$, $\gamma'' \in \Gamma^*$ and, for no ζ, $\gamma'' = \alpha_0\beta\zeta$.

Let now $\hat{C} = \langle \hat{q}, \hat{x}, \alpha_0\beta^{i_2-i_1}\hat{\zeta}\rangle$, be the first configuration such that $C(k_2; i_2) \vdash^* \hat{C} \vdash^* C''$, and for no ζ: $\hat{\zeta} = \beta\zeta$. Because of the determinism, we also have:

$$C(k_2; i_1) = \langle q, x^{k_2}, \alpha_0\beta^{i_1}\alpha_1\rangle \vdash^* \langle \hat{q}, \hat{x}, \alpha_0\hat{\zeta}\rangle,$$

and $k_2 < K(i_1)$ would satisfy (B), against the definition of $K(i)$. Hence $K(i)$ is nondecreasing. Therefore $\lim_{i->\infty} K(i)$ is either (B1) finite, $= \bar{k}$, or (B2) infinite.

The two cases, to be next discussed, correspond to (B1) clearing the stack through an ε-cycle, or (B2) clearing it by a cycle which reads some x's to remove some β's.

CASE (B1). Since $K(i)$ is an integer valued function, there exists $\bar{\imath}$ (that we may assume to be the minimal) such that $i \geqq \bar{\imath}$ implies $K(i) = \bar{k}$. Thus $\forall i \geqq \bar{\imath}$:

$$C(\bar{k}; i) = \langle q, x^{\bar{k}}, \alpha_0\beta^i\alpha_1\rangle \vdash^* \langle \bar{q}, \bar{x}_i'', \alpha_0\hat{\alpha}_i\rangle = C_i,$$

where $\bar{x}_i''$ is a suffix of x and for no $\zeta \in \Gamma^*$, $\hat{\alpha}_i = \beta\zeta$. Therefore we also have

$$C(\bar{k}; i) \vdash^* \langle \hat{q}_i, \hat{x}_i, \alpha_0\beta\rangle \vdash^* C_i$$

where $\hat{x}_i$ is a suffix of $x^{\bar{k}}$.

From the finiteness of Q and Σ, there exist i_1, i_2, with $\bar{\imath} \leqq i_1 < i_2$ such that: $\hat{q}_{i_1} = \hat{q}_{i_2} = \hat{q}$, $\hat{x}_{i_1} = \hat{x}_{i_2} = \hat{x}$, i.e.:

$$C(\bar{k}; i_1) \vdash^* \langle \hat{q}, \hat{x}, \alpha_0\beta\rangle = \hat{C},$$

$$C(\bar{k}; i_2) \vdash^* \hat{C}.$$

But the second computation can be rewritten as:

$$C(\bar{k}; i_2) \vdash^* C' = \langle \hat{q}, \hat{x}, \alpha_0\beta^{i_2-i_1+1}\rangle \vdash^* \hat{C} = \langle \hat{q}, \hat{x}, \alpha_0\beta\rangle,$$

which shows a cycle of ε-moves which pops $\hat{\imath} = i_2 - i_1$ β's from the stack.

By introducing also the output we have:

$$\langle C(\bar{k}; i_1); \varepsilon\rangle \vdash^* \langle C'; w_0\rangle,$$

$$\langle C'; \varepsilon\rangle \vdash^* \langle \hat{C}; y_1\rangle.$$

After configuration $\hat{C}$ has been reached, Lemma 5.2 can be applied, thus producing as output w_1 (through a derivation which scans $\hat{x}$ followed by some number of x's), y_2 (periodically repeated), and w_2. Note that the forms of these strings depend, in general, on the value $l < \hat{t}$ such that $i - \bar{\imath} \equiv l \pmod{\hat{t}}$, which is fixed a priori.

In conclusion the lemma is proved for case (B1) by letting:

$X_{(-1)} = X_{(0)} = \varnothing$, as in case (A);

$\bar{f} = \bar{k}$, $f(k; i) = k+1$;

$w_{0,1} = w_0$, $y_{1,1} = y_1$, $w_{1,1} = w_1$, $y_{2,1} = y_2$, $w_{2,1} = w_2$;

$h_{1,1}(k, i) = (i - \bar{\imath} - l)/\hat{t}$;

$h_{2,1} = h_2(k)$, where $h_2(k)$ is computed by the application of Lemma 5.2.

CASE (B2). In this case consider the computations of the type

$$\langle q, x^{K(i)}, \alpha_0\beta^i\alpha_1\rangle \vdash^{d*} \langle q_1, x^{K(i)-1}, \alpha_0\beta^{i-i_1}\zeta_1\rangle \vdash^{d*} \cdots$$
$$\vdash^{d*} \langle q_r, x^{K(i)-r}, \alpha_0\beta^{i-i_r}\zeta_r\rangle,$$

where $\forall j$, $1 \leqq j \leqq r < K(i)$, $q_j \in Q$, for no $\zeta' \in \Gamma^*$, $\zeta_j = \beta\zeta'$, and the sequence $\{i_j\}$ is nondecreasing.

Notice that there exists N, independent of i and j, such that $|\zeta_j| \leqq N$: this can be shown by applying a similar reasoning as in Lemma 5.2 (see also Fig. 4).

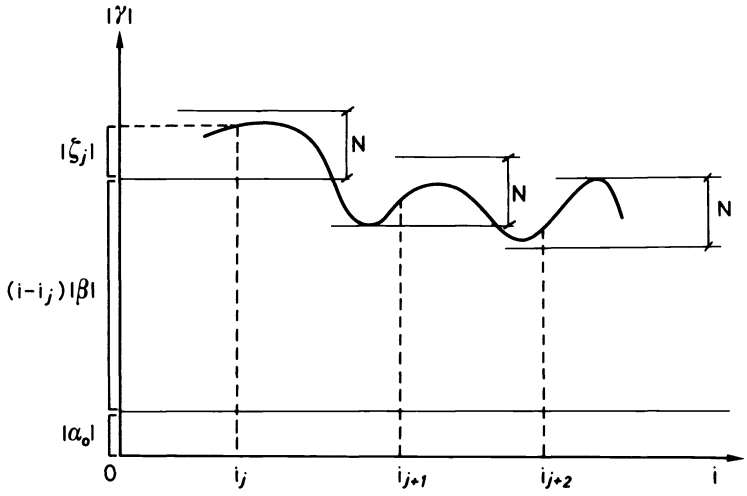


FIG. 4. *Illustration of case* (B2) *of Lemma* 5.3.

Since Q and Γ are finite, for i and r sufficiently large, there exist m, n (with $m < n$) such that:

$$q_m = q_n = \bar{q}, \qquad \zeta_m = \zeta_n = \bar{\zeta}.$$

Thus we determine

$$\langle \bar{q}, x^{n-m}, \alpha_0\beta^{i_n - i_m + 1}\zeta; \varepsilon\rangle \vdash^{d*} \langle \bar{q}, \varepsilon, \alpha_0\beta\bar{\zeta}; y_1\rangle,$$

where $i_n > i_m$ (otherwise we would be in case (A)). Let then be $\hat{k}' = n - m$, $\hat{t} = i_n - i_m$, $\bar{\imath} = i_m$, $\bar{k} = K(\bar{\imath})$.

The above computation is a cycle which removes $\beta^{\hat{\imath}}$ while reading $x^{\hat{k}'}$. If we define w_0 by

$$\langle q, x^{\bar{k}}, a_0\beta^{\bar{\imath}}\alpha_1; \varepsilon\rangle \vdash^{d*} \langle \bar{q}, \varepsilon, \alpha_0\beta\bar{\zeta}; w_0\rangle,$$

$\forall n \geqq 0$, $\forall L$, $0 \leqq L < \hat{k}'$, $\forall l$, $0 \leqq l < \hat{\imath}'$:

$$\langle q, x^{\bar{k}+n\hat{k}'+L}, a_0\beta^{\bar{\imath}+n\hat{\imath}+l+1}\alpha_1; \varepsilon\rangle \vdash^{d*} \langle \bar{q}, x^L, \alpha_0\beta^{l+1}\zeta; w_0 y_1^n\rangle.$$

Notice therefore that, letting $n = (i-\bar{\imath}-l)/\hat{\imath}$, $K(i)$ satisfies the following inequalities:

$$\bar{k}+n\hat{k}' \leqq K(i) < \bar{k}+(n+1)\hat{k}'.$$

Define the linear functions

$$\tilde{K}(i) = \bar{k}+(i-\bar{\imath}-l)/\hat{\imath}-\hat{k}', \qquad f(k; i) = k-\tilde{K}(i),$$

and separately consider the cases (i): $f(k; i) < 0$, and (ii): $f(k; i) \geqq 0$.

(i) $f(k; i) < 0$, i.e. $k < \tilde{K}(i)$. At least $\hat{\imath}+l+1$ β's are left on the stack after reading x^{k-L}, and the output produced for x^k is:

$$\tau(x^k) = w_{0,-1}y_{1,-1}^{h_{1,-1}}w_{1,-1},$$

where $w_{0,-1} = w_0$, $y_{1,-1} = y_1$, $w_{1,-1}$ is a prefix of y_1, depending on L but not on l, and

$$h_{1,-1}(k) = (k-\bar{k}-L)/\hat{k}'.$$

The string $y_{2,-1} = w_{2,-1} = \varepsilon$, which completes the definition of the behaviour in $X_{(-1)}$.

(ii) Consider now the case $f(k; i) \geqq 0$. We apply Lemma 5.2 to the configuration

$$\langle \bar{q}, x^{k-\tilde{K}(i)}, \alpha_0\beta^{\hat{\imath}+l+1}\bar{\zeta}\rangle.$$

After an initial aperiodical computation yielding $w_{1,1}$, for $k-\tilde{K}(i) > \bar{k}$ (i.e. $f(k; i) > \bar{f} = \bar{k}$), a cycle is entered:

$$\langle q', x^{\hat{k}''}, \alpha_0''\delta^r; \varepsilon\rangle \vdash^{d*} \langle q', \varepsilon, \alpha_0''\delta^{r+1}; y_{2,1}\rangle.$$

Notice that the behaviour of the automaton in this phase depends on l; in particular it is $\hat{k}'' = \hat{k}''(l)$. However a common period for (i) and (ii) can be obtained by letting $\hat{k} = \text{l.c.m.}\{\hat{k}', \hat{k}''(l) \mid 0 \leqq l < \hat{\imath}\}$.

The string $w_{2,1}$, a prefix of $y_{2,1}$, depends on the value of $k \pmod{\hat{k}}$. In conclusion, if $(k, i) \in X_{(+1)}$, we have, as in the thesis:

$$\tau(x^k) = w_{0,1}y_{1,1}^{h_{1,1}(\tilde{K}(i))}w_{1,1}y_{2,1}^{h_{2,1}(k)}w_{2,1}$$

where $w_{0,1}$, $y_{1,1}$ and $h_{1,1}$ are the same w_0, y_1, $h_{1,-1}$ as in case (i); $h_{2,1}$ is computed as in Lemma 5.2.

Finally if $(k, i) \in X_{(0)}$, i.e. $0 \leqq f(k; i) \leqq \bar{f} \Leftrightarrow K(i) \leqq k \leqq K(i)+\bar{k}$, the thesis is proved with $y_{2,0} = w_{2,0} = \varepsilon$; the length of $w_{1,0}$, a prefix of $w_{1,1}$, increases with $k-\tilde{K}(i) \leqq \bar{k}$. Q.E.D.

Proof of Lemma 5.4. For simplicity let $\eta_r = \alpha_0\beta_1^{i_1}\alpha_1 \cdots \beta_{r-1}^{i_{r-1}}\alpha_{r-1}$, $1 \leqq r \leqq s$, so that

$$C = \langle q, x^k, \eta_s\beta_s^{i_s}\alpha_s; \varepsilon\rangle.$$

As in Lemma 5.3 there exist two alternatives, denoted as follows:

$(A)_s$ there exists i_s' such that $\forall i_s \geqq i_s'$, $\forall k \geqq 1$, $\forall C_1 = \langle q_1, x_1, \gamma_1\rangle$ such that $C \vdash^* C_1$, there exists $\zeta' \in \Gamma^*$ such that $\zeta_1 = \eta_s\beta_s^{i_s-i_s'}\zeta'$.

$(B)_s$ $\forall i_s$ there exist $k = k_s(i_s)$ and a configuration C_1 reachable from C such that, for no $\zeta' \in \Gamma^*$: $\gamma_1 = \eta_s\beta_s\zeta'$.

CASE $(A)_s$. The lemma substantially coincides with Lemma 5.3, by letting $\eta_s = \alpha_0$. Thus the statement holds with:

$$c = 0 \quad \text{and} \quad X_{(+1)} = \bar{X}.$$

CASE $(B)_s$. Here again the two alternatives $(B1)_s$ and $(B2)_s$ of Lemma 5.3 have to be considered. In both cases, there exists a linear function $\tilde{K}_s(i_s)$ (which becomes a constant in case $(B1)_s$) such that, if $k > \tilde{K}_s(i_s)$:

$$C \vdash^{d*} C_1 = \langle q_1, x^{k-\tilde{K}_s(i_s)}, \eta_s \beta_s^{\hat{\imath}_s + l_s + 1} \zeta_s; w_0 y_1^{h_1(\tilde{K}_s(i_s))} \hat{w}_1 \rangle,$$

where $\hat{w}_1$ is a prefix of y_1 and $0 \leqq l_s < \hat{\imath}$, $|\zeta_s| \leqq N$.

Quite clearly state q_1 and ζ_s, hence the whole subsequent computation, depend on the value l_s $(0 \leqq l_s < \hat{\imath}_s)$ of i_s (mod $\hat{\imath}_s$).

Starting from C_1 we can reapply Lemma 5.3, distinguishing again two cases $(A)_{s-1}$ and $(B)_{s-1}$, and so on.

Let us consider case $(B)_{s-1}$. If $k > \tilde{K}_s(i_s) + \tilde{K}_{s-1}(i_{s-1})$, the previous reasoning applies to the configuration C_2:

$$C_1 \vdash^{d*} C_2 = \langle q_2, x^{k-\tilde{K}_s(i_s)-\tilde{K}_{s-1}(i_{s-1})}, \eta_{s-1} \beta_{s-1}^{\hat{\imath}_{s-1}+l_{s-1}+1} \zeta_{s-1}; w_0 y_1^{h_1(\tilde{K}_s(i_s))} w_1 y_2^{h_2(\tilde{K}_{s-1}(i_{s-1}))} \hat{w}_2 \rangle,$$

and so on until case $(A)_r$ occurs. This happens necessarily at latest for $\gamma = \alpha_0$ (letting by convention $r = 0$; Lemma 5.2).

In conclusion the lemma holds with

$$c = s - r,$$

$$f_t(k, i_s, \cdots, i_t) = k - \tilde{K}_s(i_s) - \cdots - \tilde{K}_t(i_t), \qquad s - c + 1 \leqq t \leqq s;$$

$$\bar{f}_t = \tilde{K}_{t-1}(\tilde{\imath}_{t-1}) \quad \text{for suitable } \tilde{\imath}_{t-1},$$

which intuitively correspond to the number of β_{t-1}'s popped from the stack before entering the corresponding loop. By convention $\tilde{K}_0$ is a constant function.

The strings $w_{i,p}$, $y_{i,p}$ and the functions $h_{i,p}$ are defined in the obvious way.

Notice that, as in the previous lemma, there is no problem in assuming, for the sake of notational simplicity, unique $\hat{\imath}$ and $\hat{k}$ as common periods in the above derivtions. Q.E.D.

Proof of Lemma 5.5. The dpdt reading u_0 performs

$$C_0 \vdash^{d*} \tilde{C}_0 = \langle \tilde{q}_0, x_1^{k_1} \cdots, \tilde{\alpha}_0; \hat{w}_0 \rangle.$$

Now, by Lemma 5.2 for input $x_1^{k_1}$, supposing that k_1 is "large" enough, (i.e. $k_1 \geqq \bar{k}_1$) and $k_1 - \bar{k}_1 \equiv L_1 \pmod{\hat{k}_1}$, the computation takes place:

$$\tilde{C}_0 \vdash^{d*} C_1' = \langle q_1', u_1, \cdots, \tilde{\alpha}_0' \beta_1'^{i_1'(k_1)} \alpha_1'; w_0 y_1^{h_1(k_1)} \hat{w}_1 \rangle,$$

where i_1' and h_1 are linear functions of k_1; $\hat{w}_0$ is a prefix of w_0; q_1', α_1', $\hat{w}_1$ depend on L_1; $\beta_1' \in \Gamma^*$, $y_1 \in \Omega^*$.

Next, reading u_1, one of the two following types of configuration, respectively corresponding to cases (A) and (B1) of Lemma 5.3, is reached:

1) $$C_1 = \langle q, x_2^{k_2}, \cdots, \alpha_0 \beta_1^{i_1} \alpha_1; w_0 y_1^{h_1} \hat{w}_1 \rangle,$$

where, as usual, $i_1'(k_1) - i_1$ and $|\alpha_1| \leqq N$;

2) $$C_1 = \langle q_1, x_2^{k_2}, \cdots, \alpha_0; w_0 y_1^{h_1} w_1 y_2^{h_2(k_1)} \hat{w}_2 \rangle,$$

where h_1 is a linear function of i_1', and consequently of k_1.

Let us now show by induction that, after reading $u_0 x_1^{k_1} u_1 \cdots x_j^{k_j} u_j$, the stack contents is

$$\gamma_j = \alpha_0 \beta_1^{i_1} \alpha_1 \cdots \beta_s^{i_s} \alpha_s, \quad \text{with } s \leqq j,$$

and the output is

$$\tau_j = w_0 y_1^{h_1} w_1 \cdots y_q^{h_q} \hat{w}_q,$$

with $q = 2j - s$, where i_r and h_r are linear functions of $k_1, \cdots, k_j$.

Therefore after reading the whole input string, the output contains at most $2n$ iterated string (or "periods").

Base. For $j = 1$ the statement holds, by virtue of 1) and 2) above.

Induction step. Assume that the statement holds for j, and consider the reading of $x_{j+1}^{k_{j+1}}$. By Lemma 5.4, and supposing that case $(A)_r$ is reached, with $0 \leqq r \leqq s$, the resulting stack is

$$\gamma = \eta_r \beta_r^{i'_r} \alpha'_r \beta_{r+1}'^{i'_{r+1}} \alpha'_{r+1}$$

where, as in the proof of Lemma 5.4,

$$\eta_r = \alpha_0 \beta_1^{i_1} \cdots \beta_{r-1}^{i_{r-1}} \alpha_{r-1}, \qquad s \leqq j,$$

and the output becomes

$$\tau' = w_0 \cdots y_q^{h_q} w_q y_{q+1}^{h_{q+1}} \cdots y_{q+t}^{h_{q+t}} \hat{w}_{q+t}, \quad \text{with } t = s - r + 1.$$

For instance, Fig. 5(a) and (b) resp. show, in an intuitive way, possible evolutions of the stack contents and of the output while reading $a^n b^m$.

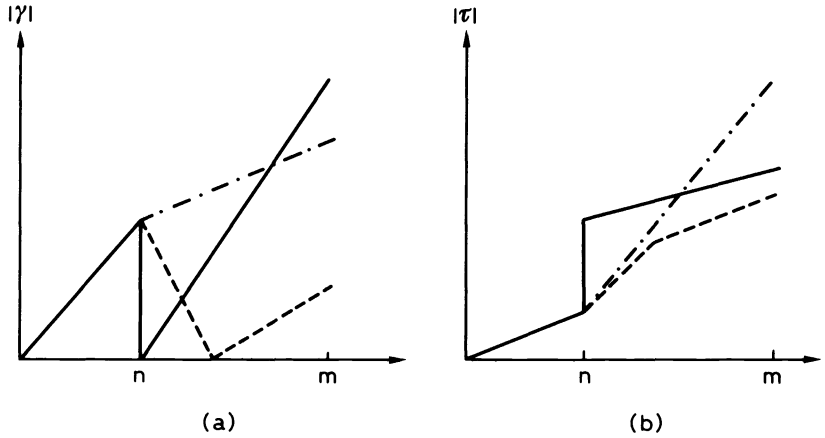


FIG. 5. *Possible growth of stack* (a) *and output* (b) *while reading* $a^n b^m$ (*Lemma* 5.5).

After reading u_{j+1}, the stack becomes

$$\gamma_{j+1} = \eta_{s'} \beta_{s'}^{i_{s'}} \alpha_{s'}, \quad \text{with } 0 \leqq s' \leqq r+1 \leqq s+1 \leqq j+1$$

(notice that if $s' = r+1$, $\beta_{s'} = \beta'_{r+1}$). The output becomes

$$\tau_{j+1} = w_0 y_1^{h_1} \cdots y_{q+t}^{h_{q+t}} w_{q+t} y_{q+t+1}^{h_{q+t+1}} \cdots y_{q'}^{h_{q'}} \hat{w}_{q'}$$

with $q' - (q+t) = r+1-s'$. Thus after the induction step, q takes the new value

$$\begin{aligned} q' &= q + t + r + 1 - s' = q + s - r + 1 + r + 1 - s' \\ &= q + s - s' + 2 = 2j - s + s - s' + 2 = 2(j+1) - s'. \end{aligned}$$

Since s' is the new value of s, and $s' \leqq j+1$ has already been shown, the inductive statement is proved.

As usual $\bar{k}$ and $\hat{k}$ are determined, and, $\forall L_1, \cdots, L_n$, with $0 \leqq L_i < \hat{k}$, the lattice

$$\bar{X} = \{k_i \geqq \bar{k},\ k_i - \bar{k} \equiv L_i \pmod{\hat{k}}\}$$

is considered. Let us show, by induction, that $\bar{X}$ is partitioned into a number R of regions $X_{(p)}$ each one being defined by at most $n-1$ relations of the types

$$0 \leqq f_t \leqq \bar{f}_t, \quad f_t < 0, \quad f_t > \bar{f}_t,$$

where f_t are linear functions of $k_1, \cdots, k_n$ and $\bar{f}_t$ are constants. In general $\{f_t\}$ and $\{\bar{f}_t\}$ depend on $X_{(p)}$.

Base. Recall that the configuration C_1 reached after reading $u_0 x_1^{k_1} u_1$ has the same form throughout $\bar{X}$. Thus for $j=1$ the number d_1 of functions defining $\bar{X}$ is 0, and $\bar{X}$ is partitioned into just one part.

Induction step. Assume that, after reading $u_0 \cdots x_j^{k_j} u_j$, $\bar{X}$ has been partitioned into R_j parts by means of $d_j \leqq j-1$ relations of the above type, and the stack contents is $\gamma_j = \alpha_0 \beta_1^{i_1} \cdots \beta_s^{i_s} \alpha_s$, with $s = s_j \leqq j$ as in the previous induction. Assume also that $d_j + s_j \leqq j$, what is true in the base of induction. Consider the dpdt behaviour in any of the R_j, say $\tilde{X}$. By applying Lemma 5.4 to the reading of $x_{j+1}^{k_{j+1}}$, $\tilde{X}$ is split into $2c_j+1$ parts through c_j linear functions (i.e. at most c_j comparisons have been performed among k_{j+1} and $(k_1, \cdots, k_j)$).

The reading of u_{j+1} does not increase the number of parts.

Thus, after reading u_{j+1}, $\bar{X}$ is partitioned into R_{j+1} parts, with $R_{j+1} \leqq (2j+1)R_j$. Each part is defined by $d_{j+1} \leqq d_j + c_j$ relations. By observing, from the previous induction, that $c_j = s_j - r$ (where r is the integer such that case $(A)_r$ occurs), we have:

$$s_{j+1} \leqq r+1 = s_j - c_j + 1.$$

Thus

$$d_{j+1} + s_{j+1} \leqq d_j + c_j + s_j - c_j + 1 = d_j + s_j + 1 \leqq j+1,$$

and the induction is completed. Q.E.D.

Remark. With a little more insight it could be shown that actually $R_n \leqq 3^{n-1}$. However this fact is not essential for the proof of the main theorem, and therefore is not proved here.

Proof of Lemma 5.6. We proceed by induction on the length of the chain.

Base. By Lemma 5.5, for $x = a^n b^m \perp$, $\tau_1(x) = w_0 y_1^{h_1} \cdots y_4^{h_4} w_4$ has 4 periods $y_1, \cdots, y_4$, which, in general, take different values in 2 angular domains and in one strip of $\bar{X}$.

Induction step. Suppose that the input x' of T_i contains 2^i periods, which, in general, differ in $a_{i-1} = 2^{2^{i-1}-i}$ angular domains and in $a_{i-1}-1$ strips. In fact, at most one strip can lie between two angular domains (see Fig. 6).

By Lemma 5.5, $\tau_i(x')$ contains at most 2^{i+1} periods, which take possibly different values in a number of angular domains and strips to be next computed.

For any angular domain A in $\bar{X}$ consider the region $K = \{k_1, \cdots, k_N\}$ with $N = 2^i$ and $k_j = k_j(n, m)$ with $(n, m) \in A$. When T_i scans the N periods of its input, the region K is split into at most $\prod_{j=1}^{N-1} (2c_j+1)$ subregions, where c_j is defined as in Lemma 5.5, what induces the splitting of A into at most $B = \prod_{j=1}^{N-1} (c_j+1)$ angular domains, and in $B-1$ strips. Notice that the number of possible different subregions of K is not necessarily $2B-1$.

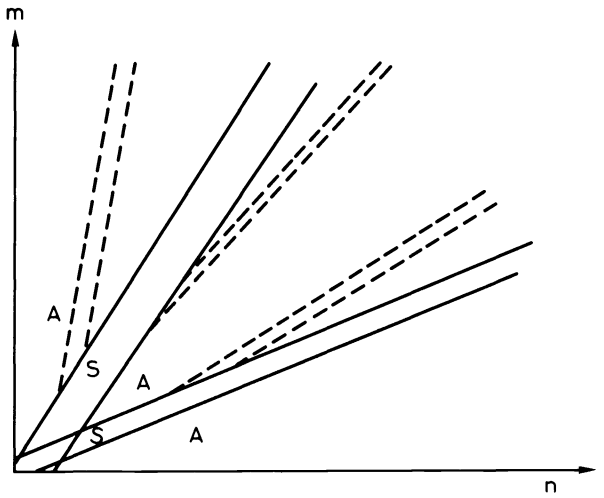


FIG. 6. *Illustration of Lemma* 5.6.

On the other hand we have from Lemma 5.5:

$$\sum_{j=1}^{N-1} c_j = d_N \leqq N-1.$$

This implies that $B = \Pi_{j=1}^{N-1} (c_j+1) \leqq 2^{N-1}$. In fact, since for any integer $x \geqq 0$ it is $\log_2 (x+1) \leqq x$, we have:

$$\log_2 B = \sum_{j=1}^{N-1} \log_2 (c_j+1) \leqq \sum_{j=1}^{N-1} c_j \leqq N-1.$$

Thus the number of possible angular domains determined by T_i is

$$a_i = a_{i-1} \times 2^{(2^i-1)} \quad \text{with } a_1 = 2.$$

Solving the recursion we have

$$\log_2 a_i = \log_2 (a_{i-1}) + 2^i - 1 \qquad (\log_2 a_1 = 1);$$

hence

$$\log_2 a_i = 1 + (2^2-1) + \cdots + (2^i-1) = 2^{i+1} - 2 - i,$$

and

$$a_i = 2^{(2^{i+1}-2-i)},$$

which completes the induction. Finally notice that the final state of T_i only depends on the domain $X_{(p)}$ of the input. Q.E.D.

REFERENCES

[1] A. V. AHO AND J. D. ULLMAN, *The Theory of Parsing, Translation and Compiling, Vols.* I *and* II, Prentice-Hall, Englewood Cliffs, NJ, 1972 and 1973.

[2] C. CITRINI, S. CRESPI-REGHIZZI AND D. MANDRIOLI, *Chains of deterministic push-down transducers,* Dept. Electronics, Politecnico di Milano, Internal Report n. 83-13, 1983, also presented in a shorter version as *Chained deterministic push-down transducers* in Colloquium on Algebra, Combinatorics and Logic in Computer Sciences, Györ, 1983.

[3] J. ENGELFRIET, *Three hierarchies of transducers*, Math. Systems Theory, 15 (1982), pp. 95-125.
[4] S. GINSBURG, *Algebraic and Automata-Theoretic Properties of Formal Languages*, North-Holland, Amsterdam, 1975.
[5] S. GINSBURG AND S. A. GREIBACH, *Deterministic context-free languages*, Inform. Control, 9 (1966), pp. 602-648.
[6] S. GINSBURG AND E. H. SPANIER, *Control sets on grammars*, Math. Systems Theory, 2 (1968), pp. 159-177.
[7] M. A. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.
[8] J. HARTMANIS AND R. E. STEARNS, *Algebraic structure theory of sequential machines*, Prentice-Hall, Englewood Cliffs, NJ, 1966.
[9] J. E. HOPCROFT AND J. D. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, London, 1969.
[10] N. A. KHABBAZ, *Multi-pass precedence analysis*, Acta Inform., 4 (1974), pp. 77-85.
[11] C. M. R. KINTALA, *Refining nondeterminism in context free languages*, Math. Systems Theory, 12 (1979), pp. 1-8.
[12] L. Y. LIU AND W. WEINER, *An infinite hierarchy of intersections of context-free languages*, Math. Systems Theory, 7 (1973), pp. 185-192.
[13] M. PLATEK, *Recognizing of languages by composition of deterministic pushdown transducers*, private communication, 1983.
[14] A. L. ROSENBERG, *Real-time definable languages*, J. Assoc. Comput. Mach., 14 (1967), pp. 645-662.
[15] A. SALOMAA, *Formal Languages*, Academic Press, New York, 1973.
[16] TADASHI AE, *Direct or cascade product of pushdown automata*, J. Comput. System Sci., 20 (1977), pp. 257-265.
[17] M. B. VITÀNYI AND W. J. SAVITCH, *On inverse deterministic pushdown transductions*, J. Comput. System Sci., 16 (1978), pp. 423-444.
[18] D. WOTSCHKE, *Nondeterminism and Boolean operations in pda's*, J. Comput. System Sci., 16 (1978), pp. 456-461.
[19] A. YEHUDAI, *A note on chains of deterministic pushdown transducers*, Internal report, Univ. California, Los Angeles, 1984.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics

THE COMPLEXITY OF RELIABILITY COMPUTATIONS IN PLANAR AND ACYCLIC GRAPHS*

J. SCOTT PROVAN†

Abstract. We show that the problem of computing source-sink reliability is NP-hard, in fact #P-complete, even for undirected and acyclic directed source-sink planar graphs having vertex degree at most three. Thus the source-sink reliability problem is unlikely to have an efficient algorithm, even when the graph can be laid out on a rectilinear grid.

Key words. reliability, complexity, planar graph, acyclic graph, NP-hard, # P-complete

1. Introduction. Connectedness reliability problems on graphs have long constituted a class of computationally intractable problems. Virtually all such problems have been shown to be NP-hard (actually #P-complete) for general graphs [2], [10], [11], [16]. Work has therefore concentrated on finding significant special classes of graphs for which polynomial algorithms do exist for computing network reliability. This has culminated in methods for computing fairly general reliability measures in directed and undirected series-parallel graphs [1], [13]. There are three important classes of graphs for which complexity results have not been obtained, namely, planar graphs, acyclic graphs, and graphs which have bounded vertex degree. There is, moreover, compelling evidence to suggest that reliability problems for these graphs might be computationally easier than for general graphs (see [3], [7], [9], [10], [14]). We show in this paper, however, that the problem of computing source-sink reliability is #P-complete, even for undirected and acyclic directed source sink planar graphs having vertex degree at most three.

2. Preliminaries. Let $G=(V, E)$ be a graph (directed or undirected) with vertex set V of cardinality m and edge set E of cardinality n. The *degree* of a vertex is the number of edges adjacent to that vertex. Two vertices s and t are distinguished as the *source* and *sink* vertices, respectively. The graph G is called *source-sink planar*, or simply (s, t)*-planar*, if it has a planar representation with s and t on the boundary. Now suppose edges in G fail independently, each edge failing with the same probability $1-p$, $0 \leqq p \leqq 1$. For the purposes of this paper, we take p to be rational. Then the (s, t)*-connectedness reliability* of G, $R(G, s, t; p)$, is the probability that there is at least one path of operating edges from s to t, where the path is taken to be directed when G is directed. This probability can be written as

$$R(G, s, t; p)=\sum_{H \in \mathcal{H}} p^{|H|}(1-p)^{n-|H|}$$

where $\mathcal{H}$ is the collection of sets of edges which contain at least one (s, t)-path. The problem of computing $R(G, s, t; p)$ has been shown to be #P-complete for undirected graphs by Valiant [16], and for directed acyclic graphs in [10]. In neither case, however, are the networks planar, nor are the vertex degrees bounded. The two problems considered in this paper are

P3ST. The problem of computing $R(G, s, t; p)$ when G is an undirected (s, t)-planar graph with each vertex having degree at most three;

* Received by the editors April 17, 1984, and in revised form January 2, 1985. This research was supported in part by the Air Force Office of Scientific Research under contract AFOSR-84-0140.

† Operations Research and Systems Analysis, University of North Carolina, Chapel Hill, North Carolina 27514.

PA3ST. The problem of computing $R(G, s, t; p)$ when G is an acyclic directed (s, t)-planar graph with each vertex having degree at most three.

It is at present unknown whether there exists a polynomial algorithm to solve either of these problems.

We explore the complexity of the problems P3ST and PA3ST in the manner proposed by Valiant [16]. We assume that the reader is familiar with the notions of NP and NP-complete problems; see [4] for an excellent account of these concepts and their relationship to #P-completeness. Fix input alphabet Σ and denote by Σ^* the corresponding collection of (finite) strings. We assume Σ^* contains a representation of $Z_+ = \{0, 1, \cdots\}$. Define the class #P to consist of those functions $f: \Sigma^* \to Z_+$ which can be computed by counting the number of accepting computations of some nondeterministic Turing machine of polynomial time complexity. For function $f: \Sigma^* \to Z_+$, define a *f-oracle Turing machine* to be a Turing machine, together with an additional *input* and *output* tape, which at any time during a computation can write string σ on the input tape and in one step receive $f(\sigma)$ on the output tape. A function $g: \Sigma^* \to Z_+$ is *polynomially reducible* to f ($g \propto f$) if there exists a polynomial time complexity f-oracle Turing machine which computes g. A function f is called #P-*complete* if (a) f is in #P and (b) every function g in #P is polynomially reducible to f.

Roughly speaking, the #P-complete problems are those which are polynomially equivalent to the counting problems associated with many NP-complete problems—for example, counting the number of Hamiltonian circuits in a graph. They are therefore at least as hard as NP-complete problems, and so it is unlikely that a polynomial algorithm exists to solve these problems. It should be noted that P3ST and P3AST are technically not #P problems, since, among other things they compute rational numbers rather then integers. However, it will follow from subsequent discussion (specifically Corollary 1) that in both P3ST and PA3ST the problem of computing R can be reduced to that of computing $2^n R(G, s, t; 1/2) = |\mathcal{H}| =$ the number of sets of edges which admit a path from s to t. This function is clearly in #P, since such edge sets are easily recognizable.

We will show that both P3ST and PA3ST are #P-complete problems. This result is somewhat surprising, since several #P-complete enumeration problems associated with (s, t)-connectness—i.e., the number of (s, t)-paths of any given length and the number of minimum cardinality (s, t)-cuts—become polynomially computable when restricted to planar graphs (in the case of minimum cuts) or acyclic graphs (in the case of paths) [3]. The planarity and bounded vertex degree in P3ST and PA3ST also mean that the problem of computing $R(G, s, t; p)$ is #P-complete even when G can be embedded in a rectangular grid with s and t on the perimeter (see [5, Thm. 1] for details).

As a starting point for the results in this paper, we present two known #P-complete functional evaluation problems.

I. Number of Hamiltonian circuits in a planar cubic graph (#HCPC)
Given: planar undirected graph with each vertex of degree three;
Find: the number of closed simple paths going through every vertex of G.

II. Acyclic (s, t)-connectedness reliability (AST)
Given: acyclic graph G, source s, sink t, and rational probability p.
Find: $R(G, s, t; p)$,

As with P3ST and PA3ST, one can think of AST as computing $2^n R(G, s, t; \frac{1}{2}) = |\mathcal{H}|$ so as to make it a member of #P. The problem AST was shown to be #P-complete in [10]. The problem #HCPC has essentially been shown to be #P-complete in [6]. Specifically, in the construction for the m clause, 3-conjunctive normal form expression

F in that paper, if the "required-edge" graph is placed into the successive figures as oriented, then the resulting graph has exactly $(8^7 \cdot 18)^m \cdot 8^{6a} \cdot 8^b \cdot 36$ Hamiltonian circuits for each assignment satisfying F, where a is the number of "crossing exclusive-or" graphs and b is the number of "noncrossing exclusive-or" graphs added in the final construction. Since the number of satisfying assignments for a 3-conjunctive normal form expression is known to be #P-complete [15], then #HCPC is also #P-complete.

We next present a key formula used to prove both of the main results of this paper. It is due to Satyanarayana and Prabhakar ([12, eq. (3)], with a correct proof in [17]), and is restated here as it applies to P3ST and PA3ST. For directed graph G and specified vertices s and t, define an (s, t)-*subgraph* of G to be an acyclic subgraph H of G having the property that every edge of H lies in at least one path in H from s to t. Equivalently, an (s, t)-subgraph can be characterized as an acyclic subgraph H such that s is the only vertex with no edges of H pointing into it and t is the only vertex with no edges of H pointing out of it. Satyanarayana and Prabhakar showed that the (s, t)-connectedness reliability for G can be written

$$R(G, s, t; p) = \sum_{i,j} \sum_{H \in \mathcal{H}_{ij}} (-1)^{j-i+1} p^j \tag{1}$$

where $\mathcal{H}_{ij}$ is the set of (s, t)-subgraphs H with i vertices and j edges. This formula can be applied as well when G is undirected. To do this, we first construct directed graph G' by replacing each undirected edge of G by two oppositely directed edges, each with probability p. Then, as proved in [2, Thm. 2], $R(G', s, t; p)$ is equal to $R(G, s, t; p)$. The (s, t)-subgraphs of G now correspond to acyclic *orientations* of undirected subgraphs of G for which every edge is in at least one (s, t)-path, so that an (s, t)-subgraph for an undirected graph will always refer to the corresponding oriented subgraph. With this modification, equation (1) holds also when G is undirected.

Finally, we present three general lemmas. The first is due to Valiant [16, Fact 5]. Define the *size* of a rational number r to be the total number of (binary) digits in the numerator and denominator, when r is presented as a fraction in lowest terms.

LEMMA 1. *If $g(x)$ is an nth degree polynomial with rational coefficients and its value is known at each of the distinct rational points $x_1, \cdots, x_{n+1}$ each of size at most d, then the coefficients of g can be deduced in time polynomial in n, d, and the maximum size of the values of $g(x_i)$.*

To give the second lemma, we need some additional notation. Let G, s, t, and p be given, and let S be a subset of edges of G. For $l = 0, \cdots, |S|$ define

$$R^l(G, s, t, S; p) = \sum_{i,j} \sum_{H \in \mathcal{H}^l_{ij}} (-1)^{j-i+1} p^j$$

where $\mathcal{H}^l_{ij}$ is the set of (s, t)-subgraphs of G with i vertices, j edges of $E - S$, and l edges of S.

LEMMA 2. *$R^l \propto R$, even when restricted to graphs which are (s, t)-planar, acyclic, or with vertex degree at most three.*

Proof. Let the arguments G, s, t, S and p be given, with m the number of vertices of G, n the number of edges of G, k the cardinality of S, and d the size of p. For $r = 0, \cdots, k$ construct the graph G_r by replacing each edge $e = (u, v)$ of S by the path of $r+1$ edges $(u, u_{e,1})(u_{e,1}, u_{e,2}), \cdots, (u_{e,r}, v)$, where $u_{e,1}, \cdots, u_{e,r}$ are new vertices. This graph has $m + kr$ vertices and $n + k(r+1)$ edges, and is planar, acyclic, or with vertex degree at most three, respectively, if G is. Further, an (s, t)-subgraph in G_r corresponds to an (s, t)-subgraph of G with each (oriented) edge of S replaced by the appropriate

(oriented) path. Equation (1) now becomes

$$R(G_r, s, t; p) = \sum_{l=0}^{k} \sum_{i,j} \sum_{H \in \mathcal{H}_{ij}^l} (-1)^{j+l(r+1)-(i+lr)+1} p^{j+l(r+1)}$$

$$= \sum_{l=0}^{k} (-p^{r+1})^l R^l(G, s, t, S; p).$$

This is a polynomial in $-p^{r+1}$. The coefficients $R^l(G, s, t, S; p)$ consist of sums of at most 2^{2n} terms (an upper bound on the number of (oriented) (s, t)-subgraphs of G) with size at most nd, so that the size of $R^l(G, s, t, S; p)$ is bounded by $n(d+2)$. Similarly, the size of each $R(G_r, s, t; p)$ is bounded by $(n+k(r+1))(d+2)$. Using Lemma 1 we can compute the $k+1$ coefficients $R^l(G, s, t, S; p)$, $l=0, \cdots, k$, from the $k+1$ values $R(G_r, s, t; p)$, $r=0, \cdots, k$, in time polynomial m, n, and d. This proves the lemma. □

Note that the reduction in Lemma 1 can be performed when p is fixed at any value other than 0 or 1, specifically for $p=1/2$. Furthermore, if we consider the case when $S=E$, then for $l=0, \cdots, n$ we have

$$R^l(G, s, t, E; p) = \sum_i \sum_{H \in \mathcal{H}_{i0}^l} (-1)^{i-1}$$

where $\mathcal{H}_{i0}^l$ is the set of (s, t)-subgraphs of G with i vertices and l edges, i.e. $\mathcal{H}_{i0}^l = \mathcal{H}_{il}$. But now we can write

$$R(G, s, t; p) = \sum_l (-p)^l R^l(G, s, t, E; p).$$

From the above discussion we obtain the following corollary.

COROLLARY 1. $R \propto R(\cdot; p=1/2)$, *even when restricted to graphs which are* (s, t)*-planar, acyclic, or with vertex degree at most three.*

As stated when defining P3ST, PA3ST, and AST, then, there is no loss of generality in considering the #P-problem of computing $|\mathcal{H}|$ rather than the reliability problem of computing R.

Lemma 2 provides a second useful corollary. Let G, s, t, and S be as in Lemma 2, and for $l=0, \cdots, |S|$, $k=0, \cdots, |E-S|$ define

$$R^{lk}(G, s, t, S) = \sum_i \sum_{H \in \mathcal{H}_{ik}^l} (-1)^{i-1} \tag{2}$$

where $\mathcal{H}_{ik}^l$ is as defined for R^l. By applying Lemma 2 twice, once to S and once to $E-S$, we have the following result:

COROLLARY 2. $R^{lk} \propto R$, *even when restricted to graphs which are* (s, t)*-planar, acyclic, or with vertex degree at most three.*

Again, the restriction to $p=1/2$ provides no loss of generality.

3. #P-complete results. We first prove the result for the undirected case.

THEOREM 1. P3ST *is* #P*-complete, in particular,* #HCPC $\propto$ P3ST.

Proof. Let $G=(V, E)$ be a planar undirected cubic graph with m vertices and n edges. Chose any edge (u, v), and form the graph $G'=(V', E')$ by replacing (u, v) by the two edges (s, u) and (v, t), where s and t are new vertices. The graph G' is clearly (s, t)-planar. Now form the graph $G''=(V'', E'')$ by replacing each (degree 3) vertex except s and t with a triangle as shown in Fig. 1. Then $|V''|=3m+2$ and $|E''|=n+3m+1$, and each edge of E' can be identified with the appropriate edge of E''. Let S be the set of edges of E'' associated with E' so that $T=E''-S$ is the set of triangle

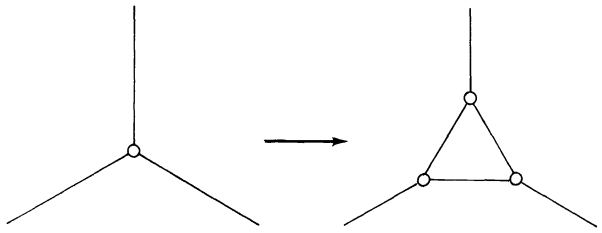


FIG. 1

edges. By applying Corollary 2, we conclude that the function

$$R^{m+1,m}(G'', s, t, S) = \sum_{i,j} \sum_{H \in \mathcal{H}^{m+1}_{i,m}} (-1)^{i-1}$$

—where $\mathcal{H}^{m+1}_{i,m}$ is the set of all oriented (s, t)-subgraphs of G'' with i vertices, $m+1$ edges S and m edges of T—is polynomially reducible to P3ST. Now for any H in $\mathcal{H}^{m+1}_{i,m}$, consider the oriented subgraph H' of G' corresponding to the $m+1$ edges of H in S. These must comprise an (s, t)-subgraph of G', which means that each vertex in H' except s and t must have degree either 2 or 3. Let V_i be the set of vertices of H' of degree i, $i = 2,3$. Again, since H is an (s, t)-subgraph then corresponding to each vertex in V_2 there must be at least one edge of $T \cap H$ and corresponding to each vertex in V_3 there must be at least two edges of $T \cap H$. Let k be the total number of edges of $T \cap H$ corresponding to vertices in V_3, so that $m-k$ is the total number of edges of $T \cap H$ corresponding to vertices in V_2. By summing the degrees of vertices in H', we have

$$2(m+1) = 2|V_2| + 3|V_3| + 2 \leqq 2(m-k) + 3k/2 + 2 = 2(m+1) - k/2$$

implying that $k = 0$. This means that H' must be a Hamiltonian path in G, and that H is obtained from H' by adding the unique set of m edges of T connecting the appropriate edges of $S \cap H$ at each vertex. Thus each H has exactly $2m+2$ vertices and corresponds in one-to-one fashion with the Hamiltonian paths from s to t in G. It follows that

$$R^{m+1,m}(G'', s, t, S) = -|\mathcal{H}^{m+1}_{2m+2,m}|$$

and we therefore obtain the number of Hamiltonian paths from s to t in G', which corresponds to the number of Hamiltonian circuits in G' containing the edge (u, v). In a similar manner, we can obtain the number of Hamiltonain circuits containing each of the other two edges of G adjacent to v, and the sum of these three values is exactly twice the number of Hamiltonian circuits in G. This completes the proof of the theorem. □

To prove that PA3ST is #P-complete, we provide an intermediate reduction. Define a directed (s, t)-planar graph G to be *contiguously directed* if it has a planar presentation for which each vertex v has its adjacent edges arranged so that in a clockwise sweep around v the edges into v and the edges out of v lie in two unbroken sequences. Define the problem PCAST to be that of computing $R(G, s, t; p)$ in an (s, t)-planar contiguously directed acyclic graph. We note that Lemma 2 and its corollaries can easily be seen to include the restriction to contiguously directed graphs.

THEOREM 2. AST$\propto$PCAST.

Proof. Suppose we are given acyclic directed graph $G = (V, E)$ with m vertices, n edges, source s, and sink t. Since G is acyclic, it follows that we can number the

vertices of G $v_1, \cdots, v_m$ such that $(v_i, v_j) \in E$ if and only if $i<j$. We can therefore place the vertices of G in the plane, representing the edges as straight lines, in such a way that each vertex v_i has x-coordinate i, $i=1, \cdots, m$, no three edges cross at the same point, and no edge crosses a nonadjacent vertex. Further, this construction can be done in polynomial time. This realization of G has the properties that no simple path can cross itself in the realization, and that the number of edge crossings is $c \leqq n^2$. Now construct the graph $\bar{G}=(\bar{V}, \bar{E})$ by successively replacing each pair of crossing edges in G by the planar subgraph shown in Fig. 2. We define the set S to comprise the collection of heavy-lined edges as shown in the figure and we set $T=\bar{E}-S$. Then $\bar{G}$ is planar with $m+6c$ vertices, $6c$ edges in S and $n+3c$ edges in T. Further, with the given realization of G it follows that $\bar{G}$ remains acyclic, and that $\bar{G}$ is contiguously directed since all edges come into a vertex from the left and leave to the right.

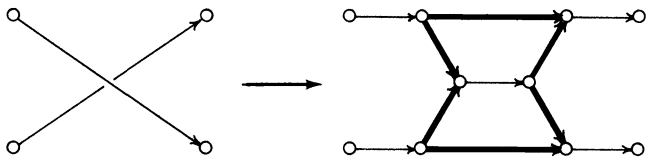


FIG. 2

Now for $l=1, \cdots, 6c$, $k=1, \cdots, n+3c$, define R^{lk} as in (2):

$$R^{lk}(\bar{G}, s, t, S)=\sum_{i=0}^{m} \sum_{H \in \bar{\mathcal{H}}_{ik}^{l}} (-1)^{i-1}$$

where $\bar{\mathcal{H}}_{ik}^{l}$ is the number of (s, t)-subgraphs of $\bar{G}$ with i vertices, l edges of S and k edges of T. It follows from Corollary 2 that $R^{lk}(\bar{G}, s, t, S)$ is reducible to R with R restricted to (s, t)-planar contiguously directed acyclic graphs. The theorem is an immediate consequence of Corollary 2 and the following claim.

CLAIM.

$$R(G, s, t; 1/2)=\sum_{l,k} (-1)^{l+k} R^{lk}(\bar{G}, s, t, S)(1/2)^{k-l}. \tag{3}$$

Proof of claim. We first rewrite (3) as

$$R(G, s, t; 1/2)=\sum_{i,l,k} \sum_{H \in \bar{\mathcal{H}}_{ik}^{l}} (-1)^{l+k-i+1}(1/2)^{k-l}. \tag{4}$$

We prove the claim by induction on the number r of replacements of the type shown in Fig. 2. The case $r=0$ follows from (1). Now suppose that (4) holds after r replacements with $\bar{G}$ the resulting graph. Suppose we replace the pair (u, v), (w, z) of crossing edges by the appropriate subgraph F to form graph $\bar{G}'$. Consider an (s, t)-subgraph H' of $\bar{G}'$. This subgraph falls into one of 10 classes, depending on which of the four edges of F adjacent to u, v, w, and z appear in H'. By symmetry we need to consider only five classes, namely class 1—in which none of these four edges appears—or one of the four classes shown in Fig. 3. For each class, consider the possible configurations of internal edges of F which could form an (s, t)-subgraph with the given outside edges. By summing $(-1)^{l'+k'-i'+1}(1/2)^{k'-l'}$ over all configurations in a class—where i', k', and l' are, respectively, the number of vertices, number of edges of S, and number of edges of T appearing in the configuration (including outside edges and vertices)—we get:

(i) the sum over all configurations in class 2 and 3 is 0;

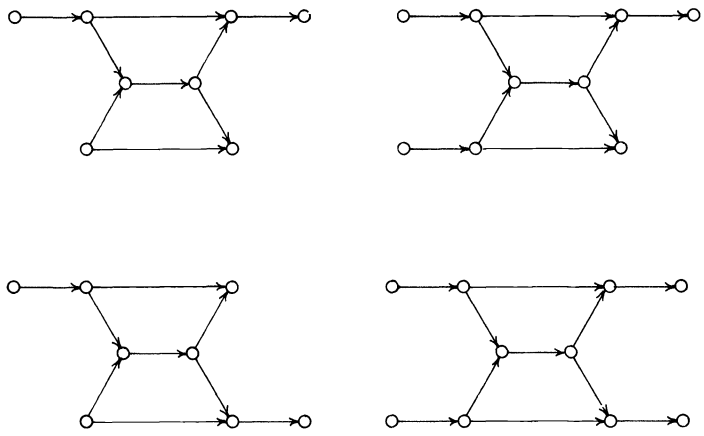


FIG. 3

(ii) the sum over all configurations in class 4 is $\frac{1}{2}$;
(iii) the sum over all configurations in class 5 is $-\frac{1}{4}$.

Now consider an (s, t)-subgraph H in $\bar{G}$. Then H falls into one of four classes, depending on which subset of the edges (u, v) and (w, z) appear in H. If H contains neither (u, v) nor (w, z) then H corresponds to exactly one subgraph of $\bar{G}'$, which is in class 1; if H contains exactly one of the two edges, say (u, v), then H corresponds to the class of (s, t)-subgraphs of $\bar{G}'$ comprised of the edges $H - F$ along with the set of class 4 configurations of F; and if H contains both of the edges (u, v) and (w, z), then H corresponds to the class of (s, t)-subgraphs of $\bar{G}'$ comprised of the edges $H - F$ along with the set of class 5 configurations of F. In each case, the net contribution to (4) of configurations in classes 1, 4, and 5 is exactly equal to contribution of the edges (u, v) and (w, z), and the net contribution of configurations in classes 2 and 3 is zero. It follows that (4) continues to hold for the graph $\bar{G}'$, and the claim follows by induction. This completes the proof of the theorem.

We can now present the result for the directed case.

THEOREM 3. PA3ST *is* #P-*complete, in particular* PCAST$\propto$ PA3ST.

Proof. Let $G = (V, E)$ be an (s, t)-planar acyclic contiguously directed graph with m vertices and n edges. For each vertex v, let

$$(u_1(v), v), (u_2(v), v), \cdots, (u_{l(v)}(v), v), (v, u_{l(v)+1}(v)), \cdots, (v, u_{k(v)}(v))$$

be the edges adjacent to v, listed in clockwise order. Now define the directed graph $G' = (V', E')$ with vertex set

$$V' = \{w_1(v), \cdots, w_{k(v)}(v);\ v \in V\}$$

and edge set $E' = S \cup T$ with

$$S = \{(w_1(v), w_2(v)), \cdots, (w_{k(v)-1}(v), w_{k(v)}(v)) : v \in V\},$$

$$T = \{(w_i(u), w_j(v)) : (u, v) \in E \text{ with } u = u_i(v) \text{ and } v = u_j(u)\}$$

The edges can be positioned as shown in Fig. 4. Finally, set $s' = w_1(s)$ and $t' = w_{k(t)}(t)$. It is clear that G' is (s', t')-planar and acyclic, that it has at most three edges adjacent to each vertex, and that $|V'| = 2n$, $|S| = 2n - m$, and $|T| = n$. Further, there is a one-to-one correspondence between the (s, t)-subgraphs of G and the (s', t')-subgraphs of G',

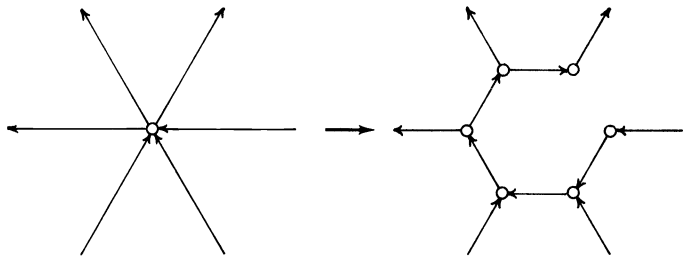


FIG. 4

obtained by associating with each (s, t)-subgraph H of G the subgraph H' of G' with edge set $A \cup B$ where

$$A = \{(w_i(u), w_j(v)) : (u, v) \in H \text{ with } u = u_i(v),\ v = u_j(u)\} \subseteq T,$$

$$B = \left\{ (w_i(v), w_{i+1}(v)), (w_{i+1}(v), w_{i+2}(v)), \cdots, (w_{j-1}(v), w_j(v)) : v \in H, \right.$$
$$\left. i = \begin{cases} 1, & v = s, \\ \min\{p : (u_p(v), v) \in H\}, & v \neq s, \end{cases} \quad j = \begin{cases} k(v), & v = t, \\ \max\{p : (v, u_p(v)) \in H\}, & v \neq t \end{cases} \right\} \subseteq S$$

so that H' has exactly $|B|$ more vertices than H. From Lemma 2 we get that $R^l(G', s', t', S; p)$ is polynomially reducible to R, with R restricted to (s, t)-planar, acyclic graphs with vertex degree at most 3. But now

$$R(G, s, t; p) = \sum_{i,j} \sum_{H \in \mathcal{H}_{ij}} (-1)^{j-i+1} p^j$$
$$= \sum_{l} (-1)^l R^l(G', s', t', S; p)$$

and the theorem follows.

4. Conclusion. We have shown in this paper that planarity, acyclicness, and bounded vertex degree, even taken together, are not sufficient to keep the problem of computing (s, t)-connectedness reliability from being NP-hard. This is true even though several counting problems associated with (s, t)-connectedness reliability become polynomial when restricted to these classes of graphs. A second important connectedness reliability problem for which the same type of analysis could be performed is the *source-to-all*, or *connectedness*, reliability problem. This is the problem of computing for a graph G—under the same edge failure distribution used in this paper—the probability that a given source vertex can reach *all* other vertices of G through paths of operating edges. For this measure, the acyclic property alone is sufficient to construct a polynomial evaluation algorithm (see [3, Thm. 4(v)]). Whether planarity alone is sufficient remains an open problem, although again several related NP-hard problems can be solved in polynomial time on planar graphs (see [3, Thm. 4(iv)], [10], and [9, Corollary 2.7 and succeeding discussion]). Further, the class of planar graphs for which this reliability can be computed efficiently has been extended slightly from series-parallel graphs to include "Cube-Free" graphs [8]. The problem for planar graphs with bounded vertex degree also remains open.

Acknowledgment. The author would like to thank Mark Jerrum for comments which significantly improved the results and presentation of this paper.

REFERENCES

[1] A. AGRAWAL AND A. SATYANARAYANA, *An $O(|E|)$-time algorithm for computing the reliability of a class of directed networks*, Oper. Res., 32 (1984), pp. 493–515.

[2] M. O. BALL, *The complexity of network reliability computations*, Networks, 10 (1977), pp. 153–165.

[3] M. O. BALL AND J. S. PROVAN, *Calculating bounds on reachability and connectedness in stochastic networks*, Networks, 13 (1983), pp. 253–278.

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

[5] ———, *The rectilinear Steiner tree problem is* NP-*complete*, SIAM J. Appl. Math., 32 (1977), pp. 826–834.

[6] M. R. GAREY, D. S. JOHNSON AND R. E. TARJAN, *The planar Hamiltonian circuit problem is* Np-*complete*, this Journal, 5 (1976), pp. 704–714.

[7] M. G. LUBY, *Monte-carlo methods for estimating system reliability*, Tech. Rep. 84/168, Computer Science Division, Univ. California, Berkeley, CA, 1982, Chapter 6.

[8] T. POLITOF AND A. SATYANARAYANA, *A linear time algorithm to compute the reliability of planar cube-free graphs*, Tech. Rept., Dept. Electrical Engineering and Computer Science, Stevens Institute of Technology, Hoboken, NJ, 1985.

[9] J. S. PROVAN, *Polyhedral combinatorics and network reliability*, Math. Oper. Res., to appear.

[10] J. S. PROVAN AND M. O. BALL, *The complexity of counting cuts and computing the probability that a graph is connected*, this Journal, 12 (1983), pp. 777–788.

[11] A. ROSENTHAL, *Computing the reliability of complex networks*, SIAM J. Appl. Math., 32 (1977), pp. 384–393.

[12] A. SATYANARAYANA AND A. PRABHAKAR, *New topological formula and rapid algorithm for reliability analysis of complex networks*, IEEE Trans. Reliability, R-27 (1978), pp. 82–100.

[13] A. SATYANARAYANA AND R. K. WOOD, *Polygon-to-chain reductions and network reliability*, this Journal, 14 (1985), pp. 818–832.

[14] A. W. SHOGAN, *Sequential bounding of the reliability of a stochastic network*, Oper. Res., 24 (1976), pp. 1027–1044.

[15] J. SIMON, *On some central problems in computational complexity*, Ph.D. Thesis (Tech. Rep. TR75-224), Department of Computer Science, Cornell University, Ithaca, NY, 1975.

[16] L. G. VALIANT, *The complexity of enumeration and reliability problems*, this Journal, 8 (1979), pp. 410–421.

[17] R. R. WYLLIE, *A theorem concerning directed graphs with applications to network reliability*, Networks, 10 (1980), pp. 71–78.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986

© 1986 Society for Industrial and Applied Mathematics
006

FILTERING SEARCH: A NEW APPROACH TO QUERY-ANSWERING*

BERNARD CHAZELLE†

Abstract. We introduce a new technique for solving problems of the following form: preprocess a set of objects so that those satisfying a given property with respect to a query object can be listed very effectively. Well-known problems that fall into this category include *range search*, *point enclosure*, *intersection*, and *near-neighbor* problems. The approach which we take is very general and rests on a new concept called *filtering search*. We show on a number of examples how it can be used to improve the complexity of known algorithms and simplify their implementations as well. In particular, *filtering search* allows us to improve on the worst-case complexity of the best algorithms known so far for solving the problems mentioned above.

Key words. computational geometry, database, data structures, filtering search, retrieval problems

AMS (MOS) subject classifications. CR categories 5.25, 3.74, 5.39

1. Introduction. A considerable amount of attention has been recently devoted to the problem of preprocessing a set S of objects so that all the elements of S that satisfy a given property with respect to a query object can be listed effectively. In such a problem (traditionally known as a *retrieval problem*), we assume that queries are to be made in a repetitive fashion, so preprocessing is likely to be a worthwhile investment. Well-known retrieval problems include *range search*, *point enclosure*, *intersection*, and *near-neighbor* problems. In this paper we introduce a new approach for solving retrieval problems, which we call *filtering search*. This new technique is based on the observation that the complexity of the *search* and the *report* parts of the algorithms *should* be made dependent upon each other—a feature absent from most of the algorithms available in the literature. We show that the notion of filtering search is versatile enough to provide significant improvements to a wide range of problems with seemingly unrelated solutions. More specifically, we present improved algorithms for the following retrieval problems:

1. *Interval Overlap*: Given a set S of n intervals and a query interval q, report the intervals of S that intersect q [8], [17], [30], [31].

2. *Segment Intersection*: Given a set S of n segments in the plane and a query segment q, report the segments of S that intersect q [19], [34].

3. *Point Enclosure*: Given a set S of n d-ranges and a query point q in $\mathfrak{R}^d$, report the d-ranges of S that contain q (a *d-range* is the Cartesian product of d intervals) [31], [33].

4. *Orthogonal Range Search*: Given a set S of n points in $\mathfrak{R}^d$ and a query d-range q, report the points of S that lie within q [1], [2], [3], [5], [7], [9], [21], [22], [23], [26], [29], [31], [35], [36].

5. *k-Nearest-Neighbors*: Given a set S of n points in the Euclidean plane E^2 and a query pair (q, k), with $q \in E^2$, $k \leqq n$, report the k points of S closest to q [11], [20], [25].

6. *Circular Range Search*: Given a set S of n points in the Euclidean plane and a query disk q, report the points of S that lie within q [4], [10], [11], [12], [16], [37].

* Received by the editors September 15, 1983, and in final revised form April 20, 1985. This research was supported in part by the National Science Foundation under grants MCS 83-03925, and the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014-83-K-0146 and ARPA order no. 4786.

† Department of Computer Science, Brown University, Providence, Rhode Island 02912.

For all these problems we are able to reduce the worst-case *space* × *time* complexity of the best algorithms currently known. A summary of our main results appears in a table at the end of this paper, in the form of complexity pairs (*storage, query time*).

2. Filtering search. Before introducing the basic idea underlying the concept of filtering search, let us say a few words on the applicability of the approach. In order to make filtering search feasible, it is crucial that the problems specifically require the exhaustive enumeration of the objects satisfying the query. More formally put, the class of problems amenable to filtering search treatment involves a finite set of objects S, a (finite or infinite) query domain Q, and a predicate P defined for each pair in $S \times Q$. The question is then to preprocess S so that the function g defined as follows can be computed efficiently:

$$g: Q \mapsto 2^S; [g(q) = \{v \in S | P(v, q) \text{ is true}\}].$$

By computing g, we mean reporting each object in the set $g(q)$ exactly once. We call algorithms for solving such problems *reporting* algorithms. Database management and computational geometry are two areas where retrieval problems frequently arise. Numerous applications can also be found in graphics, circuit design, or statistics, just to pick a few items from the abundant literature on the subject. Note that a related, yet for our purposes here, fundamentally different class of problems, calls for computing a single-valued function of the objects that satisfy the predicate. *Counting* instead of *reporting* the objects is a typical example of this other class.

On a theoretical level it is interesting to note the dual aspect that characterizes the problems under consideration: *preprocessing resources* vs. *query resources.* Although the term "resources" encompasses both space and time, there are many good reasons in practice to concentrate mostly on *space* and *query time.* One reason to worry more about storage than preprocessing time is that the former cost is permanent whereas the latter is temporary. Another is to see the preprocessing time as being amortized over the queries to be made later on. Note that in a parallel environment, *processors* may also be accounted as resources, but for the purpose of this paper, we will assume the standard sequential RAM with infinite arithmetic as our only model of computation.

The worst-case query time of a reporting algorithm is a function of n, the input size, and k, the number of objects to be reported. For all the problems discussed in this paper, this function will be of the form $O(k+f(n))$, where f is a slow-growing function of n. For notational convenience, we will say that a retrieval problem admits of an $(s(n), f(n))$-algorithm if there exists an $O(s(n))$ space data structure that can be used to answer any query in time $O(k+f(n))$. The hybrid form of the query time expression distinguishes the *search* component of the algorithm (i.e. $f(n)$) from the *report* part (i.e. k). In general it is rarely the case that, chronologically, the first part totally precedes the latter. Rather, the two parts are most often intermixed. Informally speaking, the computation usually resembles a partial traversal of a graph which contains the objects of S in separate nodes. The computation appears as a sequence of two-fold steps: (*search*; *report*).

The purpose of this work is to exploit the hybrid nature of this type of computation and introduce an alternative scheme for designing reporting algorithms. We will show on a number of examples how this new approach can be used to improve the complexity of known algorithms and simplify their implementations. We wish to emphasize the fact that this new approach is absolutely general and is not a priori restricted to any particular class of retrieval problems. The traditional approach taken in designing

reporting algorithms has two basic shortcomings:

1. The search technique is independent of the number of objects to be reported. In particular, there is no effort to balance k and $f(n)$.

2. The search attempts to locate the objects to be reported and *only* those.

The first shortcoming is perhaps most apparent when the entire set S is to be reported, in which case no search at all should be required. In general, it is clear that one could take great advantage of an *oracle* indicating a lower bound on *how many* objects were to be reported. Indeed, this would allow us to restrict the use of a sophisticated structure only for small values of k. Note in particular that if the oracle indicates that $n/k = O(1)$ we may simply check all the objects naively against the query, which takes $O(n) = O(k)$ time, and is asymptotically optimal. We can now introduce the notion of filtering search.

A reporting algorithm is said to use *filtering search* if it attempts to match searching and reporting times, i.e. $f(n)$ and k. This feature has the effect of making the search procedure *adaptively* efficient. With a search increasingly slow for large values of k, it will be often possible to reduce the size of the data structure substantially, following the general precept: *the larger the output, the more naive the search.*

Trying to achieve this goal points to one of the most effective lines of attack for filtering search, also thereby justifying its name: this is the prescription to report more objects than necessary, the excess number being at most proportional to the actual number of objects to be reported. This suggests including a postprocessing phase in order to *filter out* the extraneous objects. Often, of course, no "clean-up" will be apparent.

What we are saying is that $O(k+f(n))$ extra reports are permitted (as long, of course, as any object can be determined to be good or bad in constant time). Often, when k is much smaller than $f(n)$, it will be very handy to remember that $O(f(n))$ extra reports are allowed. This simple-minded observation, which is a particular case of what we just said, will often lead to striking simplifications in the design of reporting algorithms. The main difficulty in trying to implement filtering search is simulating the oracle which indicates a lower bound on k beforehand. Typically, this will be achieved in successive steps, by guessing higher and higher lower bounds, and checking their validity as we go along. The time allowed for checking will of course increase as the lower bounds grow larger.

The idea of filtering search is not limited to algorithms which admit of $O(k+f(n))$ query times. It is general and applies to *any* retrieval problem with nonconstant output size. Put in a different way, the underlying idea is the following: let $h(q)$ be the time "devoted" to searching for the answer $g(q)$ and let $k(q)$ be the output size; the reporting algorithm should attempt to match the functions h and k as closely as possible. Note that this induces a highly nonuniform cost function over the query domain.

To summarize our discussion so far, we have sketched a data structuring tool, *filtering search*; we have stated its underlying philosophy, mentioned one means to realize it, and indicated one useful implementation technique.

1. The *philosophy* is to make the data structure cost-adaptive, i.e. slow and economical whenever it can afford to be so.

2. The *means* is to build an oracle to guess tighter and tighter lower bounds on the size of the output (as we will see, this step can sometimes be avoided).

3. The *implementation technique* is to compute a coarse superset of the output and filter out the undesirable elements. In doing so, one must make sure that the set computed is at most proportional in size to the actual output.

Besides the improved complexity results which we will describe, the novelty of filtering search lies in its abstract setting and its level of generality. In some broad sense, the steps guiding the computation (search) and those providing the output (report) are no longer to be distinguished. Surprisingly, even under this extended interpretation, relatively few previous algorithms seem to use anything reminiscent of filtering search. Noticeable exceptions are the scheme for *circular range search* of Bentley and Maurer [4], the *priority search tree* of McCreight [31], and the algorithm for *fixed-radius neighbor search* of Chazelle [10].

After these generalities, we are now ready to look at filtering search in action. The remainder of this paper will be devoted to the retrieval problems mentioned in the introduction.

3. Interval overlap problems. These problems lead to one of the simplest and most illustrative applications of filtering search. Let $S = \{[a_1, b_1], \cdots, [a_n, b_n]\}$; the problem is reporting all the intervals in S that intersect a query interval q. Optimal solutions to this problem can be found in [8] (w.r.t. query time) and [17], [30], [31] (w.r.t. both space and query time). All these methods rely on fairly sophisticated tree structures with substantial implementation overhead. We can circumvent these difficulties by using a drastically different method based on filtering search. This will allow us to achieve both optimal space ($s(n) = n$) and optimal query time ($f(n) = \log_2 n$), as well as much greater simplicity. If we assume that the query endpoints are taken in a range of $O(n)$ elements, our method will give us an $(n, 1)$-algorithm, which constitutes the only optimal algorithm known for the discrete version of the problem. One drawback of our method, however, is that it does not seem to be easily dynamized.

To begin with, assume that all query and data set endpoints are real numbers. We define a *window-list*, $W(S)$, as an ordered sequence of linear lists (or *windows*), $W_1, \cdots, W_p$, each containing a number of intervals from S (Fig. 1). Let I_j be the interval (in general not in S) spanned by W_j. For simplicity, we will first look at the case where the query interval is reduced to a single point x. Note that this restriction is precisely the one-dimensional point enclosure problem. Let $S(x) = \{[a_i, b_i] \in S | a_i \leqq x \leqq b_i\}$ be the set of intervals of S that contain x. The idea is to ensure that the window enclosing x contains a superset of $S(x)$, but this superset contains at most $\delta - 1$ times too many intervals, where δ is a parameter to be defined later on. We can then simply read out the entire contents of the window, keeping only the desired intervals. More precisely, let $\alpha_1, \cdots, \alpha_{2n}$ be the set of endpoints in S in ascending order. Each window W_j has associated with it an *aperture*, i.e. an open interval $I_j = (\alpha_{i_j}, \alpha_{i_{j+1}})$ with $\alpha_{i_j} = \alpha_1$, $\alpha_{i_{p+1}} = \alpha_{2n}$ and $\alpha_{i_{j+1}} \leqq \alpha_{i_{j+1}}$. As can be seen, the windows of

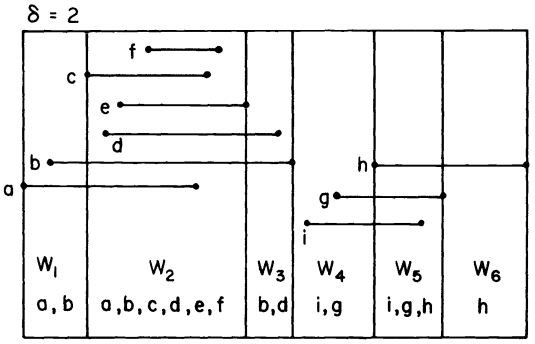


FIG. 1

$W(S)$ induce a partition of $[\alpha_1, \alpha_{2n}]$ into contiguous intervals $I_1, \cdots, I_p$, and their endpoints.

Let us assume for the time being that it is possible to define a window-list which satisfies the following density-condition: (δ is a constant > 1)

$$\sum_{1 \leq j \leq p} |W_j| < \frac{2\delta}{\delta - 1} n$$

and

$$\forall x \in I_j, \quad S(x) \subseteq W_j \quad \text{and} \quad 0 < |W_j| \leq \delta \max(1, |S(x)|).$$

It is then easy to solve the point enclosure problem in optimal time and space. To do so, we first determine which aperture I_j contains the query x and then simply check the intervals in the window W_j, reporting those containing x. If x falls exactly on the boundary between two apertures, we check both of their windows. Duplicates are easily detected by keeping a flag bit in the table where S is stored as a list of pairs. Access to windows is provided by a sorted array of cells. Each cell contains two pointers: one to α_{i_j} (for binary search) and one to W_j (or a null pointer if $j = p+1$). Each window W_j is an array of records, each pointing to an entry in S; the end is marked with a sentinel. The storage needed is $3p + 2\delta/(\delta - 1)n + O(1)$ words of roughly $\log_2 n$ bits each. This is to be added to the $O(n)$ input storage, whose exact size depends on the number representation. Because of the density-condition, the algorithm requires $O(\delta|S(x)| + \log p)$ comparisons per query, which gives an $O(k + \log n)$ query time. Note that the parameter δ allows us to trade off time and space. We next show that window-lists can be easily constructed. We first give an algorithm for setting up the data structure, and then we prove that it satisfies the density-condition.

```
Procedure Window-Make
W_1 = ∅; j := 1; T := 0; low := 1; cur := 0
for i := 1 to 2n do
  if α_i = left-point then
    cur := cur + 1
    T := T + 1
    if δ × low < T
      then  Clear (i, j)
            W_j := W_j ∪ "current interval"
            low := T := cur
      else  W_j := W_j ∪ "current interval"
  else "α_i = right-point"
    cur := cur − 1
    low := min (low, cur)
    if δ × low < T then
              Clear (i, j)
              T := cur; low := max (1, T)

Procedure Clear (i, j)
j := j + 1
W_j := {intervals [x, y] of W_{j−1} s.t. y > α_i}
```

We assume that the α_i are sorted in preprocessing; ties are broken arbitrarily. This requires $O(n \log n)$ operations, which actually dominates the linear running time of the construction phase Window-Make. This part involves scanning each α_i in ascending order and setting up the windows from left to right. We keep inserting new intervals into the current window W_j as long as T, the number of intervals so far inserted in W_j, does not exceed $\delta \times$ low, where "low" is the size of the smallest $S(x)$

found so far in W_j. The variable "cur" is only needed to update "low". It denotes the number of intervals overlapping the current position. Whenever the condition is no longer satisfied, the algorithm initializes a new window (Clear) with the intervals already overlapping in it.

A few words must be said about the treatment of endpoints with equal value. Since ties are broken arbitrarily, the windows of null aperture which will result from shared endpoints may carry only partial information. The easiest way to handle this problem is to make sure that the pointer from any shared endpoint (except for α_{2n}) leads to the first window to the right with *nonnull* aperture. If the query falls right on the shared endpoint, both its left and right windows with nonnull aperture are to be examined; the windows in between are ignored, and for this reason, need not be stored.

By construction, the last two parts of the density-condition, $\forall x \in I_j$, $S(x) \subseteq W_j$ and $0 < |W_j| \leqq \delta \max(1, |S(x)|)$, are always satisfied, so all that remains to show is that the overall storage is linear.

LEMMA 1. $\sum_{1 \leqq j \leqq p} |W_j| < 2\delta/(\delta - 1)\, n$.

Proof. Any interval $]a_i, b_i]$ can be decomposed into its window-parts, i.e. the parts where it overlaps with the I_j's. Parts which fully coincide with some I_j are called A-parts; the others are called B-parts. In Fig. 1, for example, segment b has one B-part in W_1, one A-part in W_2, and one A-part in W_3. Let A_j (resp. B_j) be the number of A-parts (resp. B-parts) in the window W_j. Note that for the purposes of this proof we substitute ranks for endpoints. This has the effect that no interval can now share endpoints (although they may in S); this implies in particular that $A_p \leqq 1$. Since a new window W_{j+1} is constructed only when $\min_{x \in I_j}(\delta |S(x)|) < |W_j| + 1$, if scanning a start-point, and $\delta(\min_{x \in I_j} |S(x)| - 1) < |W_j|$, if scanning an endpoint, we always have $\delta A_j \leqq \min_{x \in I_j}(\delta |S(x)|) < A_j + B_j + \delta$, for $j < p$. From this we derive

$$\sum_{1 \leqq j \leqq p} (A_j + B_j) < \frac{\delta}{\delta - 1} \sum_{1 \leqq j < p} B_j + \frac{\delta}{\delta - 1}(p - 1) + A_p + B_p.$$

Each endpoint a_i, b_i gives rise to at most one B-part (although some might be shared), but exactly $p+1$ endpoints do not give rise to any (falling on the boundary of a window). This implies that $\sum_{1 \leqq j \leqq p} B_j \leqq 2n - p - 1$, and since $A_p \leqq 1$ and $\delta > 1$, we have

$$\sum_{1 \leqq j \leqq p} |W_j| = \sum_{1 \leqq j \leqq p} (A_j + B_j) < \frac{2\delta}{\delta - 1} n - \frac{2\delta + B_p}{\delta - 1} + A_p < \frac{2\delta}{\delta - 1} n. \qquad \square$$

A major asset of window-lists is their simplicity of implementation. Another advantage which, at first sight, may seem paradoxical, is that window-lists are *not* based on a tree-structure. This allows us to solve the discrete version of the point enclosure problem extremely efficiently. This variant of the problem accepts only queries from a set of integers Q of range $O(n)$. We can then dispense with the preliminary binary search and find the corresponding window in constant time. To do so, we simply have to provide a table providing the correspondence point/window, which requires only $O(n)$ additional words of storage. This is a new result in that it departs from the structures in [8], [17], [30], [31], where roughly $\log_2 n$ comparisons are always required, even in the discrete case.

For the *interval overlap problem*, where the query now becomes an interval I, and all the $]a_i, b_i]$ overlapping with I are to be reported, we can still use the window-list by checking all the windows overlapping with I. An argument similar to the proof of Lemma 1 would show that, there again, a superset of $O(k)$ intervals will be reported. We omit the proof.

THEOREM 1. *There exist an* $(n, \log n)$*-algorithm, based on window-lists, for solving the* interval overlap *problem, and an* $(n, 1)$*-algorithm for solving the discrete version of the problem. Both algorithms are optimal.*

4. Segment intersection problems. *Given a set S of n segments in the Euclidean plane and a query segment q in arbitrary position, report all the segments in S that intersect q.*

For simplicity, we will assume (in § 4.1, § 4.2, § 4.3) that the n segments may intersect only at their endpoints. This is sometimes directly satisfied (think of the edges of a planar subdivision), but at any rate we can always ensure this condition by breaking up intersecting segments into their separate parts. Previous work on this problem includes an $(n \log n, \log n)$-algorithm for solving the orthogonal version of the problem in two dimensions, i.e. when query and data set segments are mutually orthogonal [34], and an $(n^3, \log n)$-algorithm for the general problem [19].

We show here how to improve both of these results, using filtering search. We first give improved solutions for the orthogonal problem (§ 4.1) and generalize our results to a wider class of intersection problems (§ 4.2). Finally we turn our attention to the general problem (§ 4.3) and also look at the special case where the query is an infinite line (§ 4.4).

4.1. The hive-graph. We assume here that the set S consists of horizontal segments and the query segment is vertical. We present an optimal $(n, \log n)$-algorithm for this problem, which represents an improvement of a factor of $\log n$ in the storage requirement of the best method previously known [34]. The algorithm relies on a new data structure, which we call a *hive-graph.*

To build our underlying data structure, we begin by constructing a planar graph G, called the *vertical adjacency map.* This graph, first introduced by Lipski and Preparata [27], is a natural extension of the set S. It is obtained by adding the following lines to the original set of segments: for each endpoint M, draw the longest vertical line passing through M that does not intersect any other segment, except possibly at an endpoint. Note that this "line" is always a segment, a half-line, or an infinite line. It is easy to construct an adjacency-list representation of G in $O(n \log n)$ time by applying a standard sweep-line algorithm along the x-axis; we omit the details. Note that G trivially requires $O(n)$ storage.

Next, we suggest a tentative algorithm for solving our intersection problem: preprocess G so that any point can be efficiently located in its containing region. This planar point location problem can be solved for general planar subdivisions in $O(\log n)$ query time, using $O(n)$ space [15], [18], [24], [28]. We can now locate, say, the lowest endpoint, (x, y_1), of the query segment q, and proceed to "walk" along the edges of G, following the direction given by q. Without specifying the details, it is easy to see that this method will indeed take us from one endpoint to the other, while passing through all the segments of G to be reported. One fatal drawback is that many edges traversed may not contribute any item to the output. Actually, the query time may be linear in n, even if few intersections are to be reported (Fig. 2).

To remedy this shortcoming we introduce a new structure, the *hive-graph* of G, denoted $H(G)$. $H(G)$ is a refinement of G supplied with a crucial *neighboring* property. Like G, $H(G)$ is a planar subdivision with $O(n)$ vertices, whose bounded faces are rectangles parallel to the axes. It is a "supergraph" of G, in the sense that it can be constructed by adding only vertical edges to G. Furthermore, $H(G)$ has the important property that each of its faces is a rectangle (possibly unbounded) with at most two extra vertices, one on each horizontal edge, in addition to its 4 (or fewer) corners. It

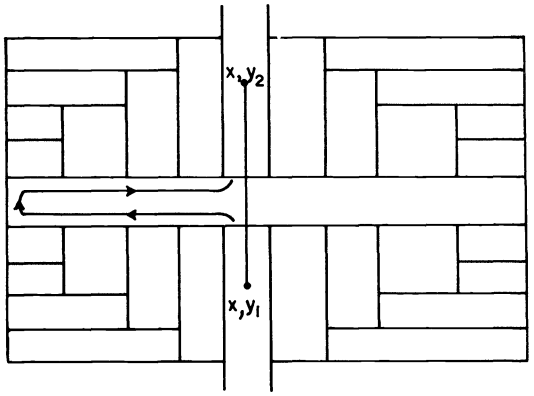


FIG. 2

is easy to see that the existence of $H(G)$ brings about an easy fix to our earlier difficulty. Since each face traversed while following the query's direction contains an edge supported by a segment of S to be reported, the presence of at most 6 edges per face ensures an $O(k)$ traversal time, hence an $O(k+\log n)$ query time.

A nice feature of this approach is that it reports the intersections in sorted order. Its "on-line" nature allows questions of the sort: report the first k intersections with a query half-line. We now come back to our earlier claim.

LEMMA 2. *There exists a hive-graph of G; it requires $O(n)$ storage, and can be computed in $O(n \log n)$ time.*

Proof. We say that a rectangle in G has an upper (resp. lower) *anomaly* if its upper (resp. lower) side consists of at least three edges. In Fig. 3, for example, the upper anomalies are on s_3, s_7, and the lower anomalies on s_1, s_2, s_4. In order to produce $H(G)$, we augment the graph G in two passes, one pass to remove each type of anomaly. If G is given a standard adjacency-list representation and the segments of S are available in decreasing y-order, $s_1, \cdots, s_n$, the graph $H(G)$ can be computed in $O(n)$ time. Note that ensuring these conditions can be done in $O(n \log n)$ steps.

The two passes are very similar, so we may restrict our investigation to the first one. *W* log, we assume that all y-coordinates in S are distinct. The idea is to sweep

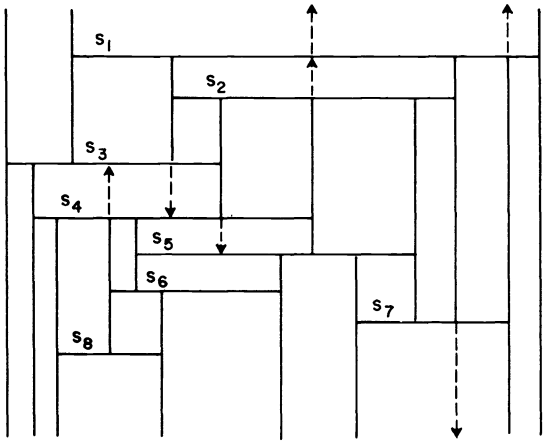


FIG. 3

an infinite horizontal line L from top to bottom, stopping at each segment in order to remove the upper anomalies. Associated with L we keep a data structure $X(L)$, which exactly reflects a cross-section of G as it will appear after the first pass.

$X(L)$ can be most simply (but not economically) implemented as a doubly linked list. Each entry is a pointer to an *old* vertical edge of G or to a *new* vertical edge. We can assume the existence of flags to distinguish between these two types. Note the distinction we make between edges and segments. The notion of edges is related to G, so for example, each segment s_i is in general made of several edges. Similarly, vertical segments adjacent to endpoints of segments of S are made of two edges. Informally, L performs a "combing" operation: it scans across G from top to bottom, dragging along vertical edges to be added into G. The addition of these new segments results in $H(G)$.

Initially L lies totally above G and each entry in $X(L)$ is *old* (in Fig. 3, $X(L)$ starts out with three entries). In the following, we will say that a vertical edge is *above* (resp. *below*) s_i if it is adjacent to it and lies completely above (resp. below) s_i. For $i = 1, \cdots, n$, perform the following steps.

1. *Identify relevant entries.* Let a_1 be the leftmost vertical edge above s_i. From a_1 scan along $X(L)$ to retrieve, in order from left to right, all the edges $A = \{a_1, a_2, \cdots, a_m\}$ above s_i.

2. *Update adjacency lists.* Insert into G the vertices formed by the intersections of s_i and the *new* edges of A. Note that if $i = 1$, there are no such edges.

3. *Update* $X(L)$. Let $B = \{b_1, \cdots, b_p\}$ be the edges below s_i, in order from left to right (note that a_1 and b_1 are collinear). Let F be a set of edges, initially empty. For each $j = 1, \cdots, p-1$, retrieve the edges of A that lie strictly between b_j and b_{j+1}. By this, we mean the set $\{a_u, a_{u+1}, \cdots, a_v\}$ such that the x-coordinate of each $a_k (u \leqq k \leqq v)$ is strictly larger (resp. smaller) than the x-coordinate of b_j (resp. b_{j+1}). Include in F *every other* edge in this set, i.e. $\{a_{u+1}, a_{u+3}, \cdots, a_{u+w}\}$, where w is the largest odd integer not exceeding $v - u$ (whenever these indices are meaningful)—see Fig. 4. Finally, delete from $X(L)$ all entries in A, insert all edges in B and in F, marking the latter *new*.

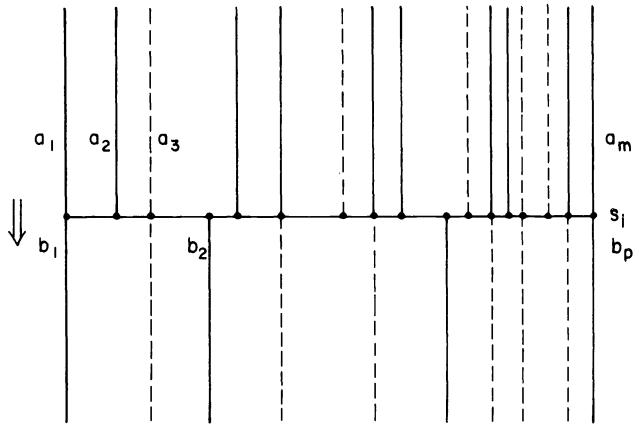


FIG. 4

This part of the algorithm can be easily implemented to run in time proportional to the size of the resulting graph. Access to a_1 in step 1 is provided in constant time by keeping a table associating with each s_i the relevant entry in $X(L)$ (if any). Steps 2 and 3 require a total of $O(m+p)$ operations, which proves our claim.

It is not difficult to see that G, in its final state, is free of upper anomalies. For every s_i, in turn, the algorithm considers the upper sides of each rectangle attached to s_i below, and subdivides these rectangles (if necessary) to ensure the presence of at most one extra vertex per upper side. Note that the creation of new edges may result in new anomalies at lower levels. This propagation of anomalies does not make it obvious that the number of edges in G should remain linear in n. This will be shown in the next paragraph.

The second pass consists of pulling L back up, applying the same algorithm with respect to lower sides. It is clear that the new graph produced, $H(G)$, has no anomalies since the second pass cannot add new upper anomalies. To see that the transformation does not add too many edges, we can imagine that we start with $O(n)$ objects which have the power to "regenerate" themselves (which is different from "duplicate", i.e. the total number of objects alive at any given time is $O(n)$ by definition). Since we extend only every other anomaly, each regeneration implies the "freezing" of at least another object. This limits the maximum number of regenerations to $O(n)$, and each pass can at most double the number of vertical edges. This proves that $|H(G)| = O(n)$, which completes the proof. □

THEOREM 2. *It is possible to preprocess n horizontal segments in $O(n \log n)$ time and $O(n)$ space, so that computing their intersections with an arbitrary vertical query segment can be done in $O(k+\log n)$ time, where k is the number of intersections to be reported. The algorithm is optimal.*

Note that if the query segment intersects the x-axis we can start the traversal from the intersection point, which will save us the complication of performing planar point location. Indeed, it will suffice to store the rectangles intersecting the x-axis in sorted order to be able to perform the location with a simple binary search. This gives us the following result, which will be instrumental in § 5.

COROLLARY 1. *Given a set S of n horizontal segments in the plane and a vertical query segment q intersecting a fixed horizontal line, it is possible to report all k segments of S that intersect q in $O(k+\log n)$ time, using $O(n)$ space. The algorithm involves a binary search in a sorted list ($O(\log n)$ time), followed by a traversal in a graph ($O(k)$ time).*

We wish to mention another application of hive-graphs, the *iterative search* of a database consisting of a collection of sorted lists $S_1, \cdots, S_m$. The goal is to preprocess the database so that for any triplet (q, i, j), the test value q can be efficiently looked up in each of the lists $S_i, S_{i+1}, \cdots, S_j$ (assuming $i<j$). If n is the size of the database, this can be done in $O((j+i)\log n)$ time by performing a binary search in each of the relevant lists. We can propose a more efficient method, however, by viewing each list S_i as a chain of horizontal segments, with y-coordinates equal to i. In this way, the computation can be identified with the computation of all intersections between the segment $[(q, i), (q, j)]$ and the set of segments. We can apply the hive-graph technique to this problem.[1]

COROLLARY 2. *Let C be a collection of sorted lists $S_1, \cdots, S_m$, of total size n, with elements chosen from a totally ordered domain U. There exists an $O(n)$ size data structure so that for any $q \in U$ and $i, j (1 \leqq i \leqq j \leqq m)$, the elements of $S_i, S_{i+1}, \cdots, S_j$ immediately following q (if any) can be found in $O(j-i+\log n)$ time. The algorithm is optimal.*

4.2. Generalizing the hive-graph. We consider now the case where the segments of S may assume an arbitrary position and the query segment has its supporting line passing through a fixed point O. We can adapt the hive-graph to handle this particular

[1] Chazelle and Guibas [13] have recently extended the notion of hive-graph into a general technique for iterative search problems, called *fractional cascading*.

situation, as we proceed to show. Let J be the planar subdivision defined as follows: for each endpoint p in S, draw the longest line collinear with the origin O, passing through p, that does not intersect any other segment except possibly at an endpoint (Fig. 5). We ensure that this "line" actually stops at point O if passing through it, so that it is always a segment or a half-line.

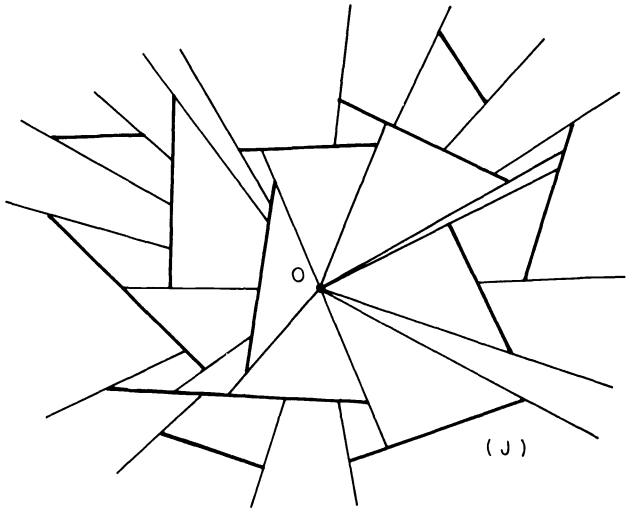


FIG. 5

It is easy to construct the adjacency-list of J in $O(n \log n)$ time, by using a standard sweep-line technique. The sweep-line is a "radar-beam" centered at O. At any instant, the segments intersecting the beam are kept in sorted order in a dynamic balanced search tree, and segments are either inserted or deleted depending on the status (first endpoint or last endpoint) of the vertex currently scanned. We omit the details. In the following, we will say that a segment is *centered* if its supporting line passes through O. As before, the hive-graph H is a planar subdivision with $O(n)$ vertices built on top of J. It has the property that each of its faces is a quadrilateral or a triangle (possibly unbounded) with two centered edges and at most two extra vertices on the noncentered edges (Fig. 6). As before, H can be easily used to solve the intersection problem at hand; we omit the details.

We can construct H efficiently, proceeding as in the last section. All we need is a new partial order among the segments of S. We say that $s_i \leqq s_j$ if there exists a ray (i.e. a half-line) emanating from O that intersects both segments, s_i before s_j. Unfortunately, as shown in Fig. 7, the directed graph induced by this relation may contain cycles. We can easily remedy this shortcoming, however, by breaking up into their two subparts each segment intersecting, say, the vertical ray emanating upwards from O.

LEMMA 3. *The relation* $\leqq$ *can be embedded in a total order.*

Proof. It suffices to prove that the induced graph does not contain any cycle. Suppose that it does, and let $s_{i_1}, \cdots, s_{i_k}, s_{i_1}$ be the shortest cycle in the graph. Call P_{i_j} the smallest wedge centered at O containing s_{i_j}. Since the cycle is the shortest, it is easy to see that the wedges $P_{i_1}, \cdots, P_{i_k}$ overlap two by two, but 1) three never overlap at the same point and 2) one never contains another totally. This shows that the sequence of wedges $P_{i_1}, \cdots, P_{i_k}$ is monotonically rotating, either clockwise or counterclockwise. This must stop before crossing the vertical ray, however, since there is no possible relation between two edges on opposite sides of the ray, hence a contradiction. □

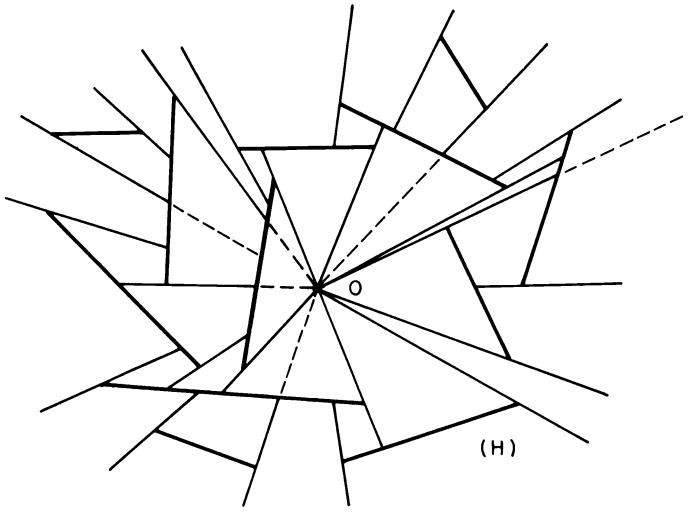


FIG. 6

Lemma 3 allows us to embed the relation $\leqq$ into a total order, called *ray-order.* To do so, we retrieve the subset of the partial order provided by J. This consists of all pairs s_i, s_j such that there exists a ray from O that intersects s_i before s_j, and nothing in between. It is easily shown that this order, denoted $\leqq^*$, contains $O(n)$ pairs, and its transitive closure coincides with that of $\leqq$. Therefore we can embed $\leqq$ into a total order by performing a topological sort on $\leqq^*$, which can be done in $O(n)$ time. Next, we compute H by proceeding as described in § 4.1, the only difference coming from the fact that the order in which to consider the edges is now given by the ray-order. We omit the details and directly conclude:

LEMMA 4. *The graph H exists, requires $O(n)$ storage, and can be computed in $O(n \log n)$ time.*

We observe that setting O at infinity (in the projective plane) gives us an algorithm for the case where the query segment has a fixed slope.

THEOREM 3. *It is possible to preprocess n segments in $O(n \log n)$ time and $O(n)$ space, so that computing their intersections with a query segment which either has a fixed slope or has its supporting line passing through a fixed point can be done in $O(k+\log n)$ time (k is the number of intersections to be reported). It is assumed that the interior of the segments are pairwise disjoint.*

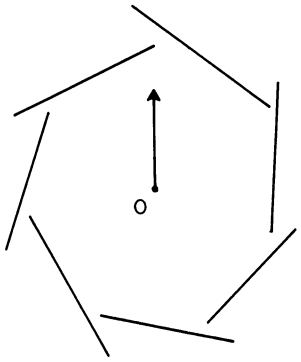


FIG. 7

4.3. The algorithm for the general case. Consider the set of all lines in the Euclidean plane. It is easy to see that the n segments of S induce a partition of this set into connected regions. A mechanical analogy will help to understand this notion. Assume that a line L, placed in arbitrary position, is free to move continuously anywhere so long as it does not cross any endpoint of a segment in S. The range of motion can be seen as a region in a "space of lines", i.e. a *dual space.* To make this notion more formal, we introduce a well-known geometric transform, T, defined as follows: a point p: (a, b) is mapped to the line T_p: $y = ax + b$ in the dual space, and a line L: $y = kz + d$ is mapped to the point T_L: $(-k, d)$. It is easy to see that a point p lies above (resp. on) a line L if and only if the point T_L lies below (resp. on) the line T_p. Note that the mapping T excludes vertical lines. Redefining a similar mapping in the projective plane allows us to get around this discrepancy. Conceptually simpler, but less elegant, is the solution of deciding on a proper choice of axes so that no segment in S is vertical. This can be easily done in $O(n \log n)$ time.

The transformation T can be applied to segments as well, and we can easily see that a segment s is mapped into a *double wedge,* as illustrated in Fig. 8. Observe that since a double wedge cannot contain vertical rays, there is no ambiguity in the definition of the double wedge once the two lines are known. We can now express the intersection of segments by means of a dual property. Let $W(s)$ and $C(s)$ denote, respectively, the double wedge of segment s and the intersection point thereof (Fig. 8).

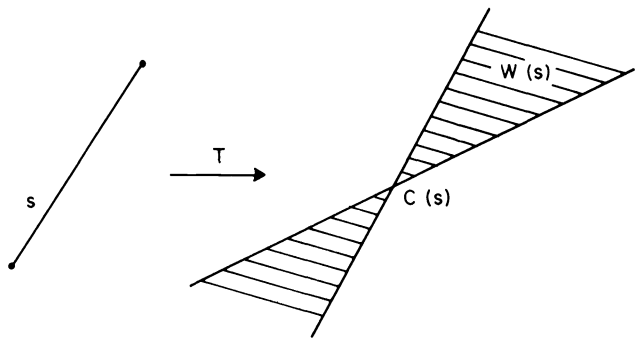


FIG. 8

LEMMA 5. *A segment s intersects a segment t if and only if $C(s)$ lies in $W(t)$ and $C(t)$ lies in $W(s)$.*

Proof. Omitted. □

This result allows us to formalize our earlier observation. Let G be the subdivision of the dual plane created by the n double wedges $W(s_i)$, with $S = \{s_1, \cdots, s_n\}$. The faces of G are precisely the dual versions of the regions of lines mentioned above. More formally, each region r in G is a convex (possibly unbounded) polygon, whose points are images of lines that all cross the same subset of segments, denoted $S(r)$. Since the segments of S are nonintersecting, except possibly at their endpoints, the set $S(r)$ can be totally ordered, and this order is independent of the particular point in r.

Our solution for saving storage over the $(n^3, \log n)$-algorithm given in [19] relies on the following idea. Recall that q is the query segment. Let $p(q)$ denote the vertical projection of $C(q)$ on the first line encountered below in G. If there is no such line, no segment of S can intersect q (or even the line supporting q) since double wedges cannot totally contain any vertical ray. Since the algorithm is intimately based on the

fast computation of $p(q)$, we will devote the next few lines to it before proceeding with the main part of the algorithm.

From [14], [20], we know how to compute the planar subdivision formed by n lines in $O(n^2)$ time. This clearly allows us to obtain G in $O(n^2)$ time and space. The next step is to organize G into an optimal planar point location structure [15], [18], [24], [28]. With this preprocessing in hand, we are able to locate which region contains a query point in $O(\log n)$ steps. In order to compute $p(q)$ efficiently, we notice that since each face of G is a convex polygon, we can represent it by storing its two chains from the leftmost vertex to the rightmost one. In this way, it is possible to determine $p(q)$ in $O(\log n)$ time by binary search on the x-coordinate of $C(q)$.

We are now ready to describe the second phase of the algorithm. Let H designate the line $T^{-1}_{p(q)}$, and let K denote the line supporting the query segment q (note that $K = T^{-1}_{C(q)}$). The lines H and K are parallel and intersect exactly the same segments of S, say, $s_{i_1}, \cdots s_{i_k}$, and in the same order. Let $s_{i_a}, s_{i_{a+1}}, \cdots, s_{i_{b-1}}, s_{i_b}$ be the list of segments intersecting $q = \alpha\beta$, with $a \leqq b$ (Fig. 9). Let p be the point whose transform T_p is the line of G passing through $p(q)$ (note that p is an endpoint of some segment in S). It is easy to see that 1) if p lies between s_{i_a} and s_{i_b}, the segments to be reported are exactly the segments of S that intersect either $p\alpha$ or $p\beta$ (Fig. 9-A). Otherwise, 2) $s_{i_a}, \cdots, s_{i_b}$ are the first segments of S intersected by $p\alpha$ or $p\beta$ starting at α or β, respectively (Fig. 9-B). Assume that we have an efficient method for enumerating the segments in S intersected by $p\alpha$ (resp. $p\beta$), in the order they appear from α (resp. β). We can use it to report the intersections of S with $p\alpha$ and $p\beta$ in case 1), as well as the relevant subsequence of intersections of S with $p\alpha$ or $p\beta$ in case 2). Note that in the latter case it takes constant time to decide which of $p\alpha$ or $p\beta$ should be considered. Our problem is now essentially solved since we can precompute a generalized hive-graph centered at each endpoint in S, as described in § 4.2, and apply it to report the intersections of S with $p\alpha$ and $p\beta$.

THEOREM 4. *It is possible to preprocess n segments in $O(n^2 \log n)$ time and $O(n^2)$ space, so that computing their intersections with an arbitrary query segment can be done in $O(k+\log n)$ time, where k is the number of intersections to be reported. It is assumed that the interior of the segments are pairwise disjoint.*

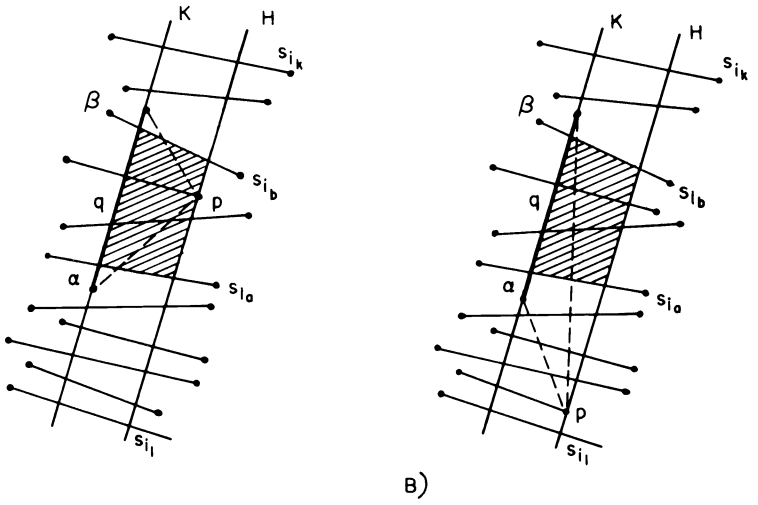


FIG. 9

4.4. The special case of a query line. We will show that if the query q is an infinite line, we can reduce the preprocessing time to $O(n^2)$ and greatly simplify the algorithm as well. At the same time, we can relax the assumption that the segments of S should be disjoint. In other words, the segments in S may now intersect. All we have to do is compute the set R of double wedges that contain the face F where $C(q)(=T_q)$ lies. To do so, we reduce the problem to one-dimensional point enclosure. We need two data structures for each line L in G. Let us orient each of the $\leqq 2n$ infinite lines in G, and designate by L_r (resp. L_l) the right (resp. left) side of line L. We construct $D(L_r)$, the data structure associated with L_r by proceeding as follows: let I be the set of intersections between L and each of the n double wedges. These intersections consist of points, segments, pairs of collinear rays, or lines. If L coincides with one of the boundary lines of a double wedge, we do not include the whole line, but only the ray on the boundary of the unique wedge lying in L_r. Finally, we organize I into a window-list, which can be done in $O(n)$ time, given the sorted order provided by G.

The data structure $D(L_l)$ is defined in a similar fashion. It is identical to $D(L_r)$, except for its treatment of wedges sharing a line with L. It is now easy to solve our intersection problem. First, assume that $C(q)$ does not lie on any edge of G. Let e be an edge of F, and let L be its supporting line. W log, assume that $C(q)$ lies in L_r. The set R is obtained by probing the window-list $D(L_r)$ with respect to the midpoint of e. If now $C(q)$ lies on e, we probe both $D(L_r)$ and $D(L_l)$, and eliminate duplicate reports with table look-up. Of course, the data structure, as presented, is redundant and can be made more economical. We will not concern ourselves with these implementation issues, however.

THEOREM 5. *It is possible to preprocess n arbitrary segments in $O(n^2)$ time and space, so that computing their intersections with an arbitrary query line can be done in $O(k+\log n)$ time, where k is the number of intersections to be reported.*

5. Point enclosure problems. Given a set S of n d-ranges and a query point q in $\Re^d$, report the d-ranges of S that contain q. Recall that a d-range is the Cartesian product of d intervals. We are able to use filtering search to improve on the most efficient algorithms previously known for the two-dimensional version of the problem: $(n \log^2 n, \log n)$ [33] and $(n, \log^3 n)$ [31]. We present an optimal $(n, \log n)$-algorithm for this problem; this solution readily generalizes into an $(n \log^{d-2} n, \log^{d-1} n)$-algorithm in $\Re^d (d > 1)$. Note that if $d = 1$ we have a special case of the interval overlap problem treated in § 3.

We look at the case $d = 2$, first. S consists of n rectangles whose edges are parallel to the axes. Let L be an infinite vertical line with $|S|$ vertical rectangle edges on each side. The line L partitions S into three subsets: S_l (resp. S_r) contains the rectangles completely to the left (resp. right) of L, and S_m contains the rectangles intersecting L. We construct a binary tree T of height $O(\log n)$ by associating with the root the set $R(\text{root}) = S_m$, and defining the left (resp. right) subtree recursively with respect to S_l (resp. S_r).

Let v be an arbitrary node of T. Solving the point enclosure problem with respect to the set $R(v)$ can be reduced to a generalized version of a one-dimensional point enclosure problem. W log, assume that the query $q = (x, y)$ is on the left-hand side of the partitioning line $L(v)$ associated with node v. Let h be the ray $[(-\infty, y), (x, y)]$; the rectangles of $R(v)$ to be reported are exactly those whose left vertical edge intersects h. These can be found in optimal time, using the hive-graph of § 4.1. Note that by carrying out the traversal of the graph from $(-\infty, y)$ to (x, y) we avoid the difficulties

of planar point location (Corollary 1). Indeed, locating $(-\infty, y)$ in the hive-graph can be done by a simple binary search in a sorted list of y-coordinates, denoted $Y_l(v)$. Dealing with query points to the right of $L(v)$ leads to another hive-graph and another set of y-coordinates, $Y_r(v)$.

The data structure outlined above leads to a straightforward $(n, \log^2 n)$-algorithm. Note that answering a query involves tracing a search path in T from the root to one of its leaves. This method can be improved by embedding the concept of hive-graph into a tree structure.

To speed up the searches, we augment $Y_l(v)$ and $Y_r(v)$ with sufficient information to provide constant time access into these lists. The idea is to be able to perform a single binary search in $Y_l(\text{root}) \cup Y_r(\text{root})$, and then gain access to each subsequent list in a constant number of steps. Let $Y(v)$ be the union of $Y_l(v)$ and $Y_r(v)$. By endowing $Y(v)$ with two pointers per entry (one for $Y_l(v)$ and the other for $Y_r(v)$), searching either of these lists can be reduced to searching $Y(v)$. Let w_1 and w_2 be the two children of v, and let W_1 (resp. W_2) be the list obtained by discarding every other element in $Y(w_1)$ (resp. $Y(w_2)$). Augment $Y(v)$ by merging into it both lists W_1 and W_2. By adding appropriate pointers from $Y(v)$ to $Y(w_1)$ and $Y(w_2)$, it is possible to look up a value y in $Y(w_1)$ or $Y(w_2)$ in constant time, as long as this look-up has already been performed in $Y(v)$.

We carry out this preprocessing for each node v of T, proceeding bottom-up. Note that the cross-section of the new data structure obtained by considering the lists on any given path is similar to a hive-graph defined with only one refining pass (with differences of minor importance). The same counting argument shows that the storage used is still $O(n)$. This leads to an $(n, \log n)$ algorithm for point enclosure in two dimensions. We wish to mention that the idea of using a hive-graph within a tree structure was suggested to us by R. Cole and H. Edelsbrunner, independently, in the context of planar point location. We conclude:

THEOREM 6. *There exists an* $(n, \log n)$*-algorithm for solving the* point enclosure *problem in* $\mathfrak{R}^2$.

The algorithm can be easily generalized to handle d-ranges. The data structure $T_d(S)$ is defined recursively as follows: project the n d-ranges on one of the axes, referred to as the x-axis, and organize the projections into a segment tree [8]. Recall that this involves constructing a complete binary tree with $2n-1$ leaves, each leaf representing one of the intervals delimited by the endpoints of the projections. With this representation each node v of the segment-tree "spans" the interval $I(v)$, formed by the union of the intervals associated with the leaves of the subtree rooted at v. Let w be an internal node of the tree and let v be one of its children. We define $R(v)$ as the set of d-ranges in S whose projections cover $I(v)$ but not $I(w)$. We observe that the problem of reporting which of the d-ranges of $R(v)$ contain a query point q whose x-coordinate lies in $I(v)$ is really a $(d-1)$-point enclosure problem, which we can thus solve recursively. For this reason, we will store in v a pointer to the structure $T_{d-1}(V)$, where V is the projection of $R(v)$ on the hyperplane normal to the x-axis. Note that, of course, we should stop the recursion at $d=2$ and then use the data structure of Theorem 6. A query is answered by searching for the x-coordinate in the segment tree, and then applying the algorithm recursively with respect to each structure $T_{d-1}(v)$ encountered. A simple analysis shows that the query time is $O(\log^{d-1} n + \text{output size})$ and the storage required, $O(n \log^{d-2} n)$.

THEOREM 7. *There exists an* $(n \log^{d-2} n, \log^{d-1} n)$*-algorithm for solving the point enclosure problem in* $\mathfrak{R}^d (d > 1)$.

6. Orthogonal range search problems.

6.1. Two-dimensional range search. A considerable amount of work has been recently devoted to this problem [1], [2], [3], [5], [7], [9], [21], [22], [23], [26], [29], [31], [35], [36]. We propose to improve upon the $(n \log^{d-1} n, \log^{d-1} n)$-algorithm of Willard [35], by cutting down the storage by a factor of $\log \log n$ while retaining the same time complexity. The interest of our method is three-fold: theoretically, it shows that, say, in two dimensions, logarithmic query time can be achieved with $o(n \log n)$ storage, an interesting fact in the context of lower bounds. More practically, the algorithm has the advantage of being conceptually quite simple, as it is made of several independent building blocks. Finally, it has the feature of being "parametrized", hence offering the possibility of trade-offs between time and space.

When the query rectangle is constrained to have one of its sides lying on one of the axes, say the x-axis, we are faced with the *grounded 2-range search* problem, for which we have an efficient data structure, the *priority search tree* of McCreight [31]. We will see how filtering search can be used to derive other optimal solutions to this problem (§ 6.3, § 6.4). But first of all, let's turn back to the original problem. We begin by describing the algorithm in two dimensions, and then generalize it to arbitrary dimensions.

In preprocessing, we sort the n points of S by x-order. We construct α vertical slabs which partition S into α equal-size subsets (called *blocks*), denoted $B_1, \cdots, B_\alpha$, from left to right. Each block contains $\lceil n/\alpha \rceil$ points, except for possibly B_α (we assume $1 < \alpha \leqq n$). The underlying data structure T is an α-ary tree, defined recursively on the following pattern. Each node v of T points to a data structure $D(v)$, constructed with respect to a certain subset $B(v)$ of S. To the root corresponds the set S itself, and to its ith child from the left corresponds the block B_i. The data structure $D(v)$ pointed to by each node v consists of either the list of points in $B(v)$ if it contains fewer than α points, or else the following items: if v_i is the ith child of v from the left,

1. each $B(v_i)$ is organized into two priority search trees, each grounded on a distinct side of the slab;

2. each $B(v_i)$ is sorted by y-coordinates and the points of $B(v_i)$ are linked together in this order. This forms a polygonal line monotone with respect to the y-axis. Consider the projections on the y-axis of the $|B(v)| - \alpha$ segments thus defined, and organize these projections into a window-list (see § 3).

The tree T being defined recursively in this manner, it is then easy to answer a query $R = \{(x_1, y_1), (x_2, y_2)\}$; $x_1 \leqq x_2$ and $y_1 \leqq y_2$. Searching for the values x_1 and x_2 in T induces a decomposition of the rectangle R into one, two, or three canonical parts; one, if it fits entirely into a slab; two, if it overlaps with two consecutive ones; three, if it overlaps with more than two. In the general case, R can be rewritten as the juxtaposition of R_1, R^*, R_2. Both R_1 and R_2 partly overlap with a slab, which gives us two instances of the *grounded 2-range search* problem for which we use the priority search trees. The next step involves determining the polygonal lines intersected by a horizontal side of R, say, $s = [(x_1, y_2), (x_2, y_2)]$. Using the window-list, this can be done in time $O(\alpha + \log n)$. Indeed, it suffices to check the window whose aperture covers y_2, which will give at most $\alpha - 1$ spurious segments. Finally, for each of the segments intersected by s, we scan the polygonal line to which it belongs, downwards, stopping as soon as we encounter a point below the lower side of R.

It is clear that each level of the tree entails $O(n)$ storage; furthermore the partition of S into equal-size blocks leads to a height of at most $\lceil \log n / \log \alpha \rceil$, therefore the total amount of storage required is $O(n(\log n / \log \alpha))$. Decomposing a query rectangle

R into its $\leqq 3$ canonical parts can be simply done in $O(\log n)$ time by performing an $O(\log \alpha)$ time binary search at each node traversed ($\lceil \log n/\log \alpha \rceil + O(1)$ nodes are visited). Finally, solving the two grounded 2-range search problems and checking all the appropriate windows can be done in time $O(\alpha + \log n + k)$, which leads to an $(n(\log n/\log \alpha), \alpha + \log n)$ algorithm. Note that setting $\alpha = n^{\epsilon}$, for $\epsilon < 1$ leads to an (n, n^{ϵ})-algorithm which matches the performance given in [5]. More interestingly, matching the first two terms in the expression of the query time leads to an $(n(\log n/\log\log n), \log n)$-algorithm, which outperforms the algorithm given in [35].

6.2. Orthogonal range search in $\mathfrak{R}^d$. We can use known techniques to extend our method to higher dimensions [2]. The basic idea is to reduce a problem in $\mathfrak{R}^d$ to a similar problem in $\mathfrak{R}^{d-1}$. The underlying data structure, $T_d(S)$, is a complete binary tree with n leaves. Let x be one of the axes; the ith leaf from left to right corresponds to the ith point of S in x-order. In this manner, each node v "spans" the set $S(v)$, formed by the points associated with leaves of the subtree rooted at v. As usual, we keep in v a pointer to a data structure $D(v)$, defined as follows: $D(v) = T_{d-1}(S^*(v))$, where $S^*(v)$ is the projection of the set $S(v)$ on the hyperplane $(x = 0)$. Of course, $T_2(S^*(v))$ is defined by using the method described above. To answer a query, we start by searching for the two x-values of the query d-range in the tree $T_d(S)$. This induces a canonical decomposition of the d-range into at most $2\lfloor \log_2 n \rfloor$ parts, each of them dealt with by projection on the hyperplane $(x = 0)$ and reduction to an orthogonal range search problem in $\mathfrak{R}^{d-1}$.

Each increment in dimensionality adds a factor of $\log n$ in the time and space complexity of the algorithm, therefore since for $d = 2$, we have an $(n(\log n/\log\log n), \log n)$-algorithm, we conclude with

THEOREM 8. *There exists an $(n(\log^{d-1} n/\log\log n), \log^{d-1} n)$-algorithm for solving the orthogonal range search problem in $\mathfrak{R}^d$ $(d > 1)$.*

6.3. Grounded 2-range search. Before closing this section, we wish to show how filtering search leads to efficient algorithms for the *grounded 2-range search* problem. The problem is to preprocess a set S of n points $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, so that for any query triple (a, b, c) with $a \leqq b$ and $0 \leqq c$, it is possible to report all the points of $\{(x_i, y_i) | a \leqq x_i \leqq b \text{ and } y_i \leqq c\}$ effectively.

Recall that this problem can be solved optimally using a priority search tree [31]. This structure is essentially a balanced binary tree with one point stored at each node. The tree is a heap with respect to the y-order, i.e. each node stores a point with smaller y-coordinate than all of its descendants. As is the case with most balanced tree-structures, although insertions and deletions can be performed in logarithmic time, overhead in implementation and running times occurs because rotations (or equivalent operations) are often necessary.

Filtering search can be used to alleviate the average cost of these operations. After sorting the n points in x-order, partition S into p blocks of roughly $\log_2 n$ points each, denoted $B_1, \cdots, B_p$, from left to right. Each block B_j is arranged as a linked-list sorted in ascending y-order. With each B_j we associate the vertical slab $V_j = \{(x, y) | b_j \leqq x \leqq b_{j+1}\}$, where the b_j's correspond to vertical lines separating the blocks. Let (α_j, β_j) be the point of B_j with smallest y-coordinate. We construct a priority search tree T with respect to the p points $(\alpha_1, \beta_1), \cdots, (\alpha_p, \beta_p)$. Answering a query (a, b, c) involves:

1. finding the two slabs V_l, $V_r (l \leqq r)$ containing $(a, 0)$ and $(b, 0)$, respectively;
2. if $l + 1 < r$, answering the query (b_{l+1}, b_r, c), using T;

3. for each point found in step 2, scanning the corresponding block upwards as long as the y-coordinate does not exceed c, and reporting all the points visited, including the starting ones;

4. checking all the points in B_l and B_r, and reporting those satisfying the query.

Note that, since the blocks B_i correspond to fixed slabs, their cardinality may vary from log n to 0 in the course of dynamic operations. We assume here that the candidate points are known in advance, so that slabs can be defined beforehand. This is the case if we use this data structure for solving, say, the all-rectangle-intersection problem (see [30] for the correspondence between the two problems). This new algorithm clearly has the same time and space complexity as the previous one, but it should require much fewer dynamic operations on the priority search tree, on the average. The idea is that to insert/delete a point, it suffices to locate its enclosing slab, and then find where it fits in the corresponding block. Since a block has at most log n points, this can be done by sequential search. Note that access to the priority search tree is necessary only when the point happens to be the lowest in its block, an event which we should expect to be reasonably rare.

6.4. Other optimal schemes. In light of our previous discussion on the hive-graph, it is fairly straightforward to devise another optimal algorithm for the grounded 2-range search problems. Simply extend a vertical ray upwards starting at each point of S. The problem is now reduced to intersecting a query segment, $[(a, c), (b, c)]$ with a set of nonintersecting rays; this can be solved optimally with the use of a hive-graph (§ 4).

Incidentally, it is interesting to see how, in the static case, filtering search allows us to reduce the space requirement of one of the earliest data structures for the ECDF (empirical cumulative distribution function) [7] and the grounded 2-range search problems, taking it from $O(n \log n)$ to $O(n)$. Coming back to the set of points $(\alpha_1, \beta_1), \cdots, (\alpha_p, \beta_p)$, we construct a complete binary tree T, as follows: the root of the tree is associated with the y-sorted list of all the p points. Its left (resp. right) child is associated with the y-sorted list $\{(\alpha_1, \beta_1), \cdots, (\alpha_{\lfloor p/2 \rfloor}, \beta_{\lfloor p/2 \rfloor})\}$ (resp. $\{(\alpha_{\lfloor p/2 \rfloor+1}, \beta_{\lfloor p/2 \rfloor i+1}), \cdots, (\alpha_p, \beta_p)\}$). The other lists are defined recursively in the same way, and hence so is the tree T. It is clear that searching in the tree for a and b induces a canonical partition of the segment $[a, b]$ into at most $2\lfloor \log_2 p \rfloor$ parts, and for each of them all the points lower than c can be obtained in time proportional to the output size. This allows us to use the procedure outlined above, replacing step 2 by a call to the new structure. The latter takes $O(p \log p) = O(n)$ space, which gives us yet another optimal $(n, \log n)$-algorithm for the grounded 2-range search problem.

7. Near-neighbor problems. We will give a final illustration of the effectiveness of filtering search by considering the *k-nearest-neighbors* and the *circular range search* problems. Voronoi diagrams are—at least in theory—prime candidates for solving neighbor problems. The Voronoi diagram of order k is especially tailored for the former problem, since it is defined as a partition of the plane into regions all of whose points have the same k nearest neighbors. More precisely, $\mathrm{Vor}_k(S)$ is a set of regions R_i, each associated with a subset S_i of k points in S which are exactly the k closest neighbors of any point in R_i. It is easy to prove that $\mathrm{Vor}_k(S)$ forms a straight-line subdivision of the plane, and D. T. Lee has shown how to compute this diagram effectively [25]. Unfortunately, his method requires the computation of all Voronoi diagrams of order $\leqq$k, which entails a fairly expensive $O(k^2 n \log n)$ running time, as well as $O(k^2(n-k))$ storage. However, the rewards are worth considering, since $\mathrm{Vor}_k(S)$ can be preprocessed to be searched in logarithmic time, using efficient planar point location methods [15], [18], [24], [28]. This means that, given any query point,

we can find the region R_i where it lies in $O(\log n)$ time, which immediately gives access to the list of its k neighbors. Note that the $O(k^2(n-k))$ storage mentioned above accounts for the list of neighbors in each region; the size of $\mathrm{Vor}_k(S)$ alone is $O(k(n-k))$. A simple analysis shows that this technique will produce an $(n^4, \log n)$-algorithm for the *k-nearest-neighbors* problem. Recall that this problem involves preprocessing a set S of n points in E^2, so that for any query $(q, k)(q \in E^2, k \leqq n)$ the k points of S closest to q can be retrieved efficiently [11], [20], [25].

Using the basic principle of filtering search: "*be naive when k is large*", we can save a factor of n space over this algorithm. The method consists of computing $\{\mathrm{Vor}_{2^i}(S) | 0 \leqq i \leqq \lceil \log_2 n \rceil\}$ (let $\mathrm{Vor}_j(S) = \mathrm{Vor}_n(S)$, if $j \geqq n$). This requires $S(n)$ storage, with

$$S(n) = O\left(\sum_{0 \leqq i \leqq \lceil \log n \rceil} 2^{2i}(n - 2^i)\right) = O(n^3).$$

All we then have to do is search the diagram $\mathrm{Vor}_{2^j}(S)$, where $2^{j-1} < k \leqq 2^j$, and apply a linear-time selection algorithm to retrieve from the corresponding set of points the k closest to the query point. Since 2^j is at most twice k, answering a query takes $O(k + \log n)$ time. We conclude to the existence of an $(n^3, \log n)$-algorithm for solving the *k-nearest-neighbors problem.* This result matches the performance of the (very different) algorithm given in [20], which is based on a compact representation of Voronoi diagrams.

We must mention that a similar technique has been used by Bentley and Maurer [4] to solve the *circular range search* problem. This problem is to preprocess a set S of n points in E^2, so that for any query disk q, the points of S that lie within q can be reported effectively. The approach taken in [4] involves searching $\mathrm{Vor}_{2^i}(S)$ for $i = 0, 1, 2, \cdots$, until we first encounter a point in S further than r away from the query point. At worst we will have checked twice as many points as needed, therefore the query time will be $O(k + \log k \log n)$, which as noticed in [4] is also $O(k + \log n \log\log n)$. The space complexity of the algorithm, stated to be $O(n^4)$ in [4], can be shown to be $O(n^3)$ using Lee's results on higher-order Voronoi diagrams [25]. For this problem, see also the elegant reduction to a three-dimensional problem used in the sublinear solution of [37].

If the radius of the query disk is fixed, we have the so-called *fixed-radius neighbor problem.* Chazelle and Edelsbrunner [12] have used filtering search to obtain an optimal $(n, \log n)$-algorithm for this problem. As regards the general problems of computing k-nearest-neighbors and performing circular range search, Chazelle, Cole, Preparata and Yap [11] have recently succeeded in drastically lowering the space requirements for both problems. The method involves a more complex and intensive use of filtering search.

THEOREM 9. [11] *There exists an* $(n (\log n \log\log n)^2, \log n)$*-algorithm for solving the* k-nearest-neighbors *problem as well as the* circular range search *problem. With probabilistic preprocessing, it is possible to obtain an* $(n \log^2 n, \log n)$*-algorithm for both problems.*

8. Conclusions and further research. We have presented several applications of a new, general technique for dealing with retrieval problems that require exhaustive enumeration. This approach, called *filtering search,* represents a significant departure from previous methodology, as it is primarily based on the interaction between *search* and *report* complexities. Of course, the idea of balancing mixed complexity factors is not new *per se.* Our basic aim in this paper, however, has been to introduce the idea of balancing all the various complexity costs of a problem as a *systematic* tool for improving on known *reporting* algorithms. To implement this idea, we have introduced

the general scheme of *scooping* a superset of candidates and then *filtering out* the spurious items. To demonstrate the usefulness of filtering search, we have shown its aptitude at reducing the complexity of several algorithms. The following chart summarizes our main results; the left column indicates the best results previously known, and the right the new complexity achieved in this paper.

problem	previous complexity	filtering search
discrete interval overlap	$(n, \log n)$[17, 30, 31]	$(n, 1)$
segment intersection	$(n^3, \log n)$ [19]	$(n^2, \log n)$
segment intersection[2]	$(n \log n, \log n)$ [34]	$(n, \log n)$
point enclosure in $\Re^d (d > 1)$	$(n \log^d n, \log^{d-1} n)$ [33]	$(n \log^{d-2} n, \log^{d-1} n)$
orthogonal range query in $\Re^d (d > 1)$	$(n \log^{d-1} n, \log^{d-1} n)$ [35]	$\left(n \frac{\log^{d-1} n}{\log \log n}, \log^{d-1} n\right)$
k-nearest-neighbors	$(n^3, \log n)$ [20]	$(n(\log n \log \log n)^2, \log n)$ [11]
circular range query	$(n^3, \log n \log \log n)$ [4]	$(n(\log n \log \log n)^2, \log n)$ [11]

In addition to these applications, we believe that filtering search is general enough to serve as a stepping-stone for a host of other retrieval problems as well. Investigating and classifying problems that lend themselves to filtering search might lead to interesting new results. One subject which we have barely touched upon in this paper, and which definitely deserves attention for its practical relevance, concerns the dynamic treatment of retrieval problems. There have been a number of interesting advances in the area of dynamization lately [6], [32], [34], and investigating how these new techniques can take advantage of filtering search appears very useful. Also, the study of upper or lower bounds for orthogonal range search in two dimensions is important, yet still quite open. What are the conditions on storage under which logarithmic query time (or a polynomial thereof) is possible? We have shown in this paper how to achieve $o(n \log n)$. Can $O(n)$ be realized?

Aside from filtering search, this paper has introduced, in the concept of the *hive-graph*, a novel technique for batching together binary searches by propagating fractional samples of the data to neighboring structures. We refer the interested reader to [13] for further developments and applications of this technique.

Acknowledgments. I wish to thank Jon Bentley, Herbert Edelsbrunner, Janet Incerpi, and the anonymous referees for many valuable comments.

REFERENCES

[1] Z. AVID AND E. SHAMIR, *A direct solution to range search and related problems for product regions*, Proc. 22nd Annual IEEE Symposium on Foundations of Computer Science, 1981, pp. 123-216.

[2] J. L. BENTLEY, *Multidimensional divide-and-conquer*, Comm. ACM, 23 (1980), pp. 214-229.

[3] J. L. BENTLEY AND J. H. FRIEDMAN, *Data structures for range searching*, Comput. Surveys, 11 (1979) 4, pp. 397-409.

[4] J. L. BENTLEY AND H. A. MAURER, *A note on Euclidean near neighbor searching in the plane*, Inform. Proc. Lett., 8 (1979), pp. 133-136.

[5] ———, *Efficient worst-case data structures for range searching*, Acta Inform., 13 (1980), pp. 155-168.

[2] With supporting line of query segment passing through a fixed point, or with fixed-slope query segment.

[6] J. L. BENTLEY AND J. B. SAXE, *Decomposable searching problems. I. Static-to-dynamic transformation*, J. Algorithms, 1 (1980), pp. 301-358.

[7] J. L. BENTLEY AND M. I. SHAMOS, *A problem in multivariate statistics: Algorithm, data structure and applications*, Proc. 15th Allerton Conference on Communication Control and Computing, 1977, pp. 193-201.

[8] J. L. BENTLEY AND D. WOOD, *An optimal worst-case algorithm for reporting intersections of rectangles*, IEEE Trans. Comput., C-29 (1980), pp. 571-577.

[9] A. BOLOUR, *Optimal retrieval algorithms for small region queries*, this Journal, 10 (1981), pp. 721-741.

[10] B. CHAZELLE, *An improved algorithm for the fixed-radius neighbor problem*, IPL 16(4) (1983), pp. 193-198.

[11] B. CHAZELLE, R. COLE, F. P. PREPARATA AND C. K. YAP, *New upper bounds for neighbor searching*, Tech. Rep. CS-84-11, Brown Univ., Providence, RI, 1984.

[12] B. CHAZELLE AND H. EDELSBRUNNER, *Optimal solutions for a class of point retrieval problems*, J. Symb. Comput., 1 (1985), to appear, also in Proc. 12th ICALP, 1985.

[13] B. CHAZELLE AND L. J. GUIBAS, *Fractional cascading: a data structuring technique with geometric applications*, Proc. 12th ICALP, 1985.

[14] B. CHAZELLE, L. J. GUIBAS AND D. T. LEE, *The power of geometric duality*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, Nov. 1983, pp. 217-225.

[15] R. COLE, *Searching and storing similar lists*, Tech. Rep. No. 88, Computer Science Dept., New York Univ., New York, Oct. 1983.

[16] R. COLE AND C. K. YAP, *Geometric retrieval problems*, Proc. 24th Annual IEEE Symposium on Foundations of Computer Science, Nov. 1983, pp. 112-121.

[17] H. EDELSBRUNNER, *A time- and space- optimal solution for the planar all-intersecting-rectangles problem*, Tech. Rep. 50, IIG Technische Univ. Graz, April 1980.

[18] H. EDELSBRUNNER, L. GUIBAS AND J. STOLFI, *Optimal point location in a monotone subdivision*, this Journal, 15 (1986), pp. 317-340.

[19] H. EDELSBRUNNER, D. G. KIRKPATRICK AND H. A. MAURER, *Polygonal intersection searching*, IPL, 14 (1982), pp. 74-79.

[20] H. EDELSBRUNNER, J. O'ROURKE AND R. SEIDEL, *Constructing arrangements of lines and hyperplanes with applications*, Proc. 24th Annual IEEE Symposium on Foundation of Computer Science, Nov. 1983, pp. 83-91.

[21] R. A. FINKEL AND J. L. BENTLEY, *Quad-trees: a data structure for retrieval on composite keys*, Acta Informat. (1974), pp. 1-9.

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

[23] ———, *Lower bounds on the complexity of some optimal data structures*, this Journal, 10 (1981), pp. 1-10.

[24] D. G. KIRKPATRICK, *Optimal search in planar subdivisions*, this Journal, 12 (1983), pp. 28-35.

[25] D. Y. LEE, *On k-nearest neighbor Voronoi diagrams in the plane*, IEEE Trans. Comp., C-31 (1982), pp. 478-487.

[26] D. T. LEE AND C. K. WONG, *Quintary trees: A file structure for multidimensional data base systems*, ACM Trans. Database Syst., 1 (1980), pp. 339-353.

[27] W. LIPSKI AND F. P. PREPARATA, *Segments, rectangles, contours*, J. Algorithms, 2 (1981), pp. 63-76.

[28] R. J. LIPTON AND R. E. TARJAN, *Applications of a planar separator theorem*, this Journal, 9 (1980), pp. 615-627.

[29] G. LUEKER, *A data structure for orthogonal range queries*, Proc. 19th Annual IEEE Symposium on Foundations of Computer Science, 1978, pp. 28-34.

[30] E. M. MCCREIGHT, *Efficient algorithms for enumerating intersecting intervals and rectangles*, Tech. Rep. Xerox PARC, CSL-80-9, 1980.

[31] ———, *Priority search trees*, Tech. Rep. Xerox PARC, CSL-81-5, 1981; this Journal 14 (1985), pp. 257-276.

[32] M. H. OVERMARS, *The design of dynamic data structures*, Ph.D. thesis, Univ. Utrecht, 1983.

[33] V. K. VAISHNAVI, *Computing point enclosures*, IEEE Trans. Comput., C-31 (1982), pp. 22-29.

[34] V. K. VAISHNAVI AND D. WOOD, *Rectilinear line segment intersection, layered segment trees and dynamization*, J. Algorithms, 3 (1982), pp. 160-176.

[35] D. E. WILLARD, *New data structures for orthogonal range queries*, this Journal, 14 (1985), pp. 232-253.

[36] A. C. YAO, *Space-time trade-off for answering range queries*, Proc. 14th Annual ACM Symposium on Theory of Computing, 1982, pp. 128-136.

[37] F. F. YAO, *A 3-space partition and its applications*, Proc. 15th Annual ACM Symposium on Theory of Computing, April 1983, pp. 258-263.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
007

ON A MULTIDIMENSIONAL SEARCH TECHNIQUE AND ITS APPLICATION TO THE EUCLIDEAN ONE-CENTRE PROBLEM*

M. E. DYER†

Abstract. The paper is divided into two main sections. The first deals with a multidimensional search technique of Megiddo [J. Assoc. Comput. Mach., 31 (1984), pp. 114–127], and suggests an improvement. The second gives an application of the technique to the Euclidean one-centre problem in $\mathbb{R}^d$. An algorithm of time-complexity $O(3^{(d+2)^2}n)$ is derived for this problem. This improves the best previous bound even in the case $d=2$.

Key words. multidimensional search, linear time algorithms, Euclidean one-centre problem, computational geometry, location problems

1. Introduction. This paper is divided into two main sections which are closely related, but also to some extent independent. The first section, § 2, discusses in some detail a multidimensional search procedure due to Megiddo [11], and gives a clarification of, and an improvement upon, Megiddo's idea. Lower bounds on the efficiency of the technique are also briefly examined.

The other main section, § 3, deals with an application of the search technique to a well-known problem in computational geometry, the weighted Euclidean one-centre problem in d-dimensional space. We show that there exists an algorithm which runs in $O(3^{(d+2)^2}n)$ time for this problem. This time-bound is linear in n for any fixed d, and improves the best previously known bounds even in the planar case $d=2$.

2. A multidimensional search technique.

2.1. Introduction. In [11], Megiddo presented an ingenious technique for a multidimensional search problem. The exact problem, and the nature of Megiddo's solution, is described in § 2.2 below. The technique is a major component of an algorithm for linear programming which runs in time which grows only linearly with the size of the problem in any fixed dimension d. Megiddo's demonstration of the existence of such an algorithm is clearly an important development in the theory of linear programming. Unfortunately, however, the "constant factor" for this algorithm grows as fast as $2^{O(2^d)}$. Here we describe an improvement to the technique which reduces the order of growth of this constant to $O(3^{(d+1)^2})$. Megiddo [11] discusses the problem from a geometrical point of view, and gives only a fairly informal development of his method. However, his arguments (which rest at one point on an appeal to a diagram) appear to be entirely correct only in nondegenerate cases. These points will be detailed further below. The paper [11] also contains a method for handling other types of degeneracy which is shown in the present paper to be unnecessary. In § 2.3 we give a more detailed exposition of the method, incorporating the above-mentioned improvement, and show that the degenerate cases are, if anything, an advantage and need little special attention. The problem and technique are discussed from an algebraic standpoint for ease of rigorous development.

2.2. The search problem. Suppose we are given n affine functions in $\mathbb{R}^d$:

$$h_i(x) = a_i^T x + b_i \qquad (i = 1, 2, \cdots, n). \tag{2.1}$$

* Received by the editors September 10, 1984, and in revised form April 12, 1985.

† Department of Mathematics and Statistics, Teesside Polytechnic, Middlesbrough, Cleveland TS1 3BA, United Kingdom.

Let $x_0 \in \mathbb{R}^d$ be any point. We wish to determine the sign of $h_i(x_0)$ for some fixed proportion p of the h_i, i.e. for at least pn values of i. By the sign of a number, we mean the obvious mapping into the set $\{-1, 0, +1\}$. We denote this by sign $(\cdot)$. We will also abbreviate sign $(h_i(x_0))$ to sign (h_i) where no ambiguity arises.

An *enquiry* will mean an evaluation of sign (h) for some given affine function in $\mathbb{R}^d$. (This need not be one of the h_i.) We assume that the only access to information about x_0 is through an "oracle" which can decide arbitrary enquiries. Thus we present an h to it, and it returns us sign (h). The problem is then, for some given p, to achieve the above objective while, in some sense, minimizing the number of enquiries. It is obvious that, for any given p and n, some finite number of enquiries suffices, since we can simply choose any $\lceil pn \rceil$ of the h_i and enquire about these. Rather more surprisingly, however, Megiddo showed in [11] that there exists a p for which a number of enquiries suffices which is independent of n. To be specific, he showed that if $p = 2^{1-2^d}$, then 2^{d-1} enquiries will suffice. Here we generalise his technique, and produce an improvement which, in the context of its applications, is shown to be advantageous.

2.3. The search technique. First note that if any h_i in (2.1) is a constant function (i.e. $a_i = 0$) then sign (h_i) can be determined without any enquiries. Thus we may assume that all such h_i are removed initially. Their removal will obviously increase the proportion of h_i whose signs are discovered, for, supposing there are n_0 such h_i,

$$n_0 + p(n - n_0) = (1-p)n_0 + pn > pn.$$

Now let l_i be the leading (in order of variable indexing) coefficient in h_i. Since sign $(h_i) =$ sign $(l_i) \cdot$ sign (h_i/l_i), we may assume that $l_i = +1$ without loss of generality.

First consider the case $d = 1$. We have then n functions of the form

$$h_i(x) = x_1 + b_i \qquad (i = 1, 2, \cdots, n).$$

Let β be the median of the b_i, i.e. the $\lfloor \frac{1}{2}(n+1) \rfloor$th largest. Then $x_1 + b_i = (x_1 + \beta) + (b_i - \beta)$. Now, in one enquiry, we can determine sign (h) for $h(x) = x_1 + \beta$. Clearly if

(a) sign $(h) = +1$, then we know sign $(h_i) = +1$ for the $\lceil \frac{1}{2}(n+1) \rceil > \frac{1}{2}n$ values of i for which $b_i \geqq \beta$;

(b) sign $(h) = -1$, then we know sign $(h_i) = -1$ for the $\lfloor \frac{1}{2}(n+1) \rfloor \geqq \frac{1}{2}n$ values of i for which $b_i \leqq \beta$;

(c) sign $(h) = 0$, then we know sign (h_i) for all i.

Thus, in all cases, with one enquiry we know the sign of at least half of the h_i.

Now, suppose $d \geqq 2$. First we select all h_i for which the coefficient of x_1 is zero. Let these form the set S_1, i.e. $h_i \in S_1$ if $a_{i1} = 0$. We will use lower-case letters to denote the cardinality of sets represented by the corresponding upper-case letter. Thus, for example, $s_1 = |S_1|$. Now the $h_i \notin S_1$ have the form:

$$h_i(x) = x_1 + a_{i2}x_2 + h_i'(x),$$

where $h_i'(x)$ is an affine function of $x_3, x_4, \cdots, x_d$.

We now make (roughly) half of the a_{i2} nonnegative and half nonpositive by a change of variable, as follows.

Let α be the median of the a_{i2}, and substitute $x_1' = x_1 + \alpha x_2$. Then $h_i(x) = x_1' + a_{i2}'x_2 + h_i'(x)$, where $a_{i2}' = a_{i2} - \alpha$.

The change of variables should, in principle, be reversed before presenting the enquiries to the oracle, but otherwise makes no real difference to the problem. Thus we will now drop the primes on x_1' and a_{i2}' for convenience.

We now select all h_i for which $a_{i2}=0$ (or, in the original variables, $a_{i2}=\alpha$), and let these form the set S_2.

Now consider the remaining $h_i \notin S_1 \cup S_2$. We form these into pairs such that, in each pair, there is one a_{i2} of each sign $+1$ or -1. The pairs are otherwise arbitrary.

Let i, j be a typical pair, so that

$$\begin{aligned} h_i(x) &= x_1 + a_{i2}x_2 + h_i'(x) \quad \text{with } a_{i2} > 0, \\ h_j(x) &= x_1 + a_{j2}x_2 + h_j'(x) \quad \text{with } a_{j2} < 0. \end{aligned} \tag{2.2}$$

Note that we will be able to form at least $\frac{1}{2}(n-s_1)-s_2$ such pairs. If any h_i are left unpaired, these are now simply discarded from further consideration for the moment. Now, for each pair i, j, we can eliminate either x_1 or x_2 from h_i and h_j to give two affine functions (with leading coefficient $+1$). Thus from (2.2) these are:

$$\begin{aligned} h_{ij}^{(1)}(x) &= (h_i(x) - h_j(x))/(a_{i2} - a_{j2}), \\ h_{ij}^{(2)}(x) &= (-a_{j2}h_i(x) + a_{i2}h_j(x))/(a_{i2} - a_{j2}). \end{aligned} \tag{2.3}$$

Note that $a_{i2} - a_{j2} \neq 0$, since $a_{i2} > 0 > a_{j2}$. Now $h_{ij}^{(1)}$ involves only $x_2, x_3, \cdots, x_d$ and $h_{ij}^{(2)}$ involves only $x_1, x_3, \cdots, x_d$. Thus each is an affine function of only $(d-1)$ variables. Let S_{12} be the set of $h_{ij}^{(1)}$ and S_{21} the set of $h_{ij}^{(2)}$. The pairing gives a one-to-one mapping of S_{12} onto S_{21}. In particular,

$$s_{12} = s_{21} \geqq \tfrac{1}{2}(n - s_1) - s_2. \tag{2.4}$$

Now, we may invert the system (2.3) to give:

$$\begin{aligned} h_i(x) &= a_{i2}h_{ij}^{(1)}(x) + h_{ij}^{(2)}(x) \quad \text{with } a_{i2} > 0, \\ h_j(x) &= a_{j2}h_{ij}^{(1)}(x) + h_{ij}^{(2)}(x) \quad \text{with } a_{j2} < 0. \end{aligned} \tag{2.5}$$

Note here that it is crucial that $a_{i2} \neq a_{j2}$, i.e. they are not both zero. It is here that a minor error occurs in [11]. Megiddo asserts that each pair requires only a nonnegative and a nonpositive a_{i2}. It is easy to see that this will not do since, if $a_{i2} = a_{j2} = 0$, the transformation between (2.3) and (2.5), or vice versa, is singular. Because of the informal geometric development, this seems to have become a little confused in [11] with h_i, h_j being linearly dependent, which would only be true in $\mathbb{R}^2$. The difficulty is not apparent in [11] since, at this point, an appeal is made to a diagram which represents only the nonsingular case (2.3). Now, from (2.5), it is easy to see that if we know sign $(h_{ij}^{(1)})$ and sign $(h_{ij}^{(2)})$, then we can determine sign (h_i) or sign (h_j) or both. Specifically,

(i) If sign $(h_{ij}^{(1)}) = 0$, then sign $(h_i) =$ sign $(h_j) =$ sign $(h_{ij}^{(2)})$,

(ii) If sign $(h_{ij}^{(1)}) = +1$, then
  if $h_{ij}^{(2)}(x_0) \geqq 0$, then sign $(h_i) = +1$,
  if $h_{ij}^{(2)}(x_0) \leqq 0$, then sign $(h_j) = -1$;

(iii) If sign $(h_{ij}^{(1)}) = -1$, then
  if $h_{ij}^{(2)}(x_0) \leqq 0$, then sign $(h_i) = -1$,
  if $h_{ij}^{(2)}(x_0) \geqq 0$, then sign $(h_j) = +1$.

In all cases, we can deduce the sign of at least one of the pair h_i, h_j. We can use this to recursively define a procedure for the search problem.

Formally, a *procedure* for the search problem is an algorithm $A(d, q, p)$ with an oracle subroutine for enquiries. It accepts as input a d-dimensional system (2.1) of size n and outputs the signs of at least pn of the $h_i(x_0)$, after making at most q calls

to the oracle. The time-complexity of $A(d, q, p)$ will be denoted by $E_A(n)$, excluding calls to the oracle.

Suppose now we have two procedures $A_1(d-1, q_1, p_1)$, $A_2(d-1, q_2, p_2)$ for the search problem in $\mathbb{R}^{d-1}$. It is possible $A_1 \equiv A_2$. We combine these to form a procedure $A(d, q, p)$ in $\mathbb{R}^d$ as follows. We assume that an oracle is given for the point $x_0 \in \mathbb{R}^d$.

(a) Apply A_1 to $S_1 \cup S_{12}$. Note that all functions in $S_1 \cup S_{12}$ involve only $x_2, \cdots, x_d$. The oracle for A_1 must respond to enquiries about x_0 restricted to these coordinates. It is clear that the oracle for A can do this by restricting enquiries so that the first coefficient of h is zero. Thus the oracle for A_1 is a restriction of the oracle supplied for A. Procedure A must handle some simple encoding and decoding of A_1's enquiries, but otherwise they are simply calls to A's oracle. Let T_1 be the set of functions in $S_1 \cup S_{12}$ whose sign is discovered by A_1. Write $M_1 = T_1 \cap S_1$, $M' = T_1 \cap S_{12}$. We have, after q_1 enquiries,

$$m_1 + m' \geqq p_1(s_1 + s_{12}). \tag{2.6}$$

(b) Apply A_2 to $c(M') \cup S_2$, where $c(M') \subseteq S_{21}$ is the set of $h^{(2)}$ corresponding to the $h^{(1)}$ in M'. These functions involve only $x_1, x_3, \cdots, x_d$. Similar remarks to those in (a) about the oracle for A_2 apply here also. Let T_2 be the set of functions whose sign is discovered by A_2. Write $M_2 = T_2 \cap S_2$, $M_{12} = T_2 \cap c(M')$. Then we have, after a further q_2 enquiries,

$$m_2 + m_{12} \geqq p_2(s_2 + m'). \tag{2.7}$$

At the end of A_2, we will therefore have discovered the sign of h_i for all $h_i \in M_1 \cup M_2$. For each $h^{(2)} \in M_{12}$, we will know its sign and also that of its corresponding $h^{(1)}$, since this is in M'. Thus we will know the sign of at least one h_i for every $h^{(2)} \in M_{12}$, applying (2.5). Thus A can deduce sign (h_i) for at least

$$\begin{aligned} m &\geqq m_1 + m_2 + m_{12} \quad \text{values of } i, \text{ with } q_1 + q_2 \text{ enquiries in total} \\ &\geqq m_1 + p_2(s_2 + m') \quad \text{from (2.7)} \\ &\geqq m_1 + p_2(s_2 + p_1(s_1 + s_{12}) - m_1) \quad \text{from (2.6)} \\ &\geqq m_1 + p_2(s_2 + p_1(s_1 + \tfrac{1}{2}(n - s_1) - s_2) - m_1) \quad \text{from (2.4)}, \end{aligned}$$

i.e.

$$m \geqq \tfrac{1}{2}p_1p_2(n + s_1) + p_1(1 - p_2)s_2 + (1 - p_2)m_1. \tag{2.8}$$

Thus,

$$m \geqq \tfrac{1}{2}p_1p_2n. \tag{2.9}$$

Therefore we have a procedure $A(d, q_1 + q_2, \frac{1}{2}p_1p_2)$.

Note, from (2.8), that the existence of nonempty S_1, S_2 actually increases the proportion of signs which are discovered. In [11], to avoid degenerate cases, Megiddo proposes a sequence of operations designed to destroy this advantageous situation should it exist. From the above, however, it seems that sparsity is useful here, as in linear programming by the Simplex Method. Under certain assumptions on the coefficients, we may be able to guarantee higher values of p than are given by (2.9). We will not explore this idea here, but merely observe that it has implications for the application to linear programming.

We observe further that the work done by A over and above that done in A_1 and A_2 is $O(nd)$, since it involves only simple matrix manipulations and median-finding operations. The median-finding can be done in $O(n)$ time [2].

The above construction is used recursively, basing the recursion on the procedure in $\mathbb{R}^1$, which is an $A(1, 1, \frac{1}{2})$, described earlier. We explore this idea below.

Megiddo's procedure [11] involves taking $q_1 = q_2 = 1$ in every dimension. Used recursively he shows that this gives $q = 2^{d-1}$ and $p = 2^{1-2^d}$. Thus, while the number of enquiries is relatively small, the proportion discovered is vanishingly small for (say) $d > 3$. However, the number of enquiries does not seem to be the most appropriate measure of the merit of a procedure. A more relevant measure is what we will call its *efficiency*, $e = p/q$. Note that e measures the average proportion discovered per enquiry, and we would obviously wish to maximize this.

To see why this definition of efficiency is appropriate, note that the applications of this technique ([11] and § 3 below in particular) lead to recurrence equations of the form:

$$T(n, d) = T((1-\alpha p)n, d) + rq(T(n, d-1) + K_1 nd) + K_2 nd \tag{2.10}$$

where T is the time bound for some algorithm, r is a constant integer, α is a constant proportion and K_1, K_2 are constants.

We are looking for a solution of the form $T(n, d) \leqq C(d)n$, where $C(d)$ is a constant depending only on d. (Thus the algorithm is required to be linear-time in fixed dimension.) Clearly, from (2.10), we must have

$$C(d)n \geqq C(d)n(1-\alpha p) + rq(C(d-1) + K_1 nd) + K_2 nd,$$

i.e.

$$C(d) \geqq (r/\alpha e)(C(d-1) + K_1 d) + K_2 d/(p\alpha). \tag{2.11}$$

In order to keep the rate of growth of $C(d)$ with d as small as possible, we obviously require e to be as large as possible. To a lesser extent we want p to be large, but it is not this term which dominates the behaviour of $C(d)$.

Now Megiddo's procedure, which is an $A(d, 2^{d-1}, 2^{1-2^d})$, has $e = 2^{1-2^d}/2^{d-1} = 2^{2-d-2^d}$. This is doubly exponentially small in d. Since the number of enquiries, using a recursion of the type indicated above, will grow exponentially with d, we might hope to be able to obtain efficiencies which are singly exponentially small, but no larger. We show below that this can indeed be achieved.

The device required is extremely simple, and was already used in a rudimentary form by Megiddo in his Approach II in [11]. That is, suppose we have r procedures $A(d, q_k, p_k)$, $k = 1, 2, \cdots, r$. Then we may construct a procedure $A(d, q, p)$ with

$$p = 1 - \prod_{k=1}^{r} (1-p_k) \quad \text{and} \quad q = \sum_{k=1}^{r} q_k, \tag{2.12}$$

simply by applying A_1, removing the h_i whose signs are now known, applying A_2 to the remainder, and so on. In fact, the symmetry of (2.12) shows that it is irrelevant (in the worst case) which order is used for the r procedures. This idea could be exploited in a number of ways. We will examine only the most obvious avenues of approach. Suppose we have some fixed procedure $A(d-1, q_{d-1}, p_{d-1})$ in $\mathbb{R}^{d-1}$. Then we may repeat this procedure k times to give a procedure $A'(d-1, kq_{d-1}, 1-(1-p_{d-1})^k)$ by the above observation. We may combine two such procedures A', A'' with k, l, by the above methods to give a procedure $A_1(d, q, p)$ with

$$q = (k+l)q_{d-1} \quad \text{and} \quad p = \tfrac{1}{2}\{1-(1-p_{d-1})^k\}\{1-(1-p_{d-1})^l\}. \tag{2.13}$$

We will call this a $[k, l]$ procedure in $\mathbb{R}^d$. The symmetry of (2.13) shows that it is sufficient to consider $k \leqq l$. Also, it is easily shown that for fixed $(k+l)$ the value of p

in (2.13) is maximized when $k = l$ or $k = l-1$. Thus we really only need consider $[k, k]$ or $[k, k+1]$ procedures.

We may now combine a number of such procedures in $\mathbb{R}^d$, using (2.12). A *scheme* will comprise a list of r $[k_i, l_i]$ procedures for $i = 1, 2, \cdots, r$. Such a scheme will guarantee a proportion

$$p_d = 1 - \prod_{i=1}^{r} (1 - \tfrac{1}{2}\{1-(1-p_{d-1})^{k_i}\}\{1-(1-p_{d-1})^{l_i}\}) \tag{2.14}$$

with

$$q_d = \sum_{i=1}^{r} (k_i + l_i) q_{d-1} \tag{2.15}$$

enquiries.

We now suppose that a fixed scheme is used recursively, based on the $A(1, 1, \frac{1}{2})$ procedure. Thus

$$q_d = \left(\sum_{i=1}^{r} (k_i + l_i) \right)^{d-1}$$

and we will generate a sequence of procedures $A(d, q_d, p_d)$, with p_d being expressed by the recurrence (2.14), subject to the initial condition $p_1 = \frac{1}{2}$. Let the right-hand side of (2.14) be denoted by $f(p_{d-1})$. Then it is clear that f is an increasing function from $[0, 1]$ into itself. We then have the recurrence

$$p_d = f(p_{d-1}) \quad \text{with } p_1 = \tfrac{1}{2}. \tag{2.16}$$

Recurrences of the form (2.16) have been well studied in numerical analysis, as simple iteration methods for finding the root of an equation $p = f(p)$. The asymptotic behaviour of (2.16) is wholly determined by the roots of this equation. Now clearly $p = 0$ is always a root of $p = f(p)$, from (2.14). However, since it is easily shown that

$$f(p) = \tfrac{1}{2}\left(\sum_{i=1}^{r} k_i l_i \right) p^2 + O(p^3),$$

it follows that convergence to this root will always be doubly exponentially fast. Thus we will not improve effectively on Megiddo's $[1, 1]$ scheme unless there is a root of $p = f(p)$ with $p > 0$. By exhaustive checking, it may be shown that 9 is the smallest value of $\sum_{i=1}^{r} (k_i + l_i)$ for which such a nonzero root exists. There are several schemes for which the root then exists. However, since $p_1 = \frac{1}{2}$, we would like a scheme which guarantees $p_d \geqq \frac{1}{2}$ for all d. The only scheme with $\sum_{i=1}^{r} (k_i + l_i) = 9$ for which this is true is a $[2, 2]$, $[2, 3]$ scheme. If $p_{d-1} \geqq \frac{1}{2}$, then from (2.14),

$$p_d \geqq 1 - (1 - \tfrac{1}{2}(\tfrac{3}{4})^2)(1 - \tfrac{1}{2} \cdot \tfrac{3}{4} \cdot \tfrac{3}{8}) = 1059/2048 > \tfrac{1}{2}.$$

Thus, used recursively, this scheme guarantees $p = \frac{1}{2}$ with $q = 9^{d-1}$ enquiries, i.e. it generates a sequence of procedures $A(d, 9^{d-1}, \frac{1}{2})$. This has efficiency $e \geqq 1/(2 \times 9^{d-1})$, which is superior to that of Megiddo's $[1, 1]$ scheme for every $d \geqq 3$. Furthermore, its singly exponential behaviour enables us to reduce the constant in Megiddo's linear programming algorithm from $2^{O(2^d)}$ to $O(3^{(d+1)^2})$. (See § 3 below for an analogous argument given in detail.) Megiddo's Approach II of [11] may be viewed as a repeated $[1, 1]$ with $r = \lceil \log_2 n \rceil$. Again this does not produce as good a bound for linear programming. (In fact a repeated $[1, 1]$ with $r = 6$ is sufficient to guarantee $p = \frac{1}{2}$, and would then give a superior bound to either of the approaches of [11].)

In $\mathbb{R}^2$, a [1, 1] procedure is more efficient than the above [2, 2], [2, 3] scheme. However, the most efficient scheme in $\mathbb{R}^2$ turns out to be a [2, 2] procedure. This has $q=4$ enquiries, and guarantees $p=9/32$. Its efficiency is therefore $9/128>1/16$, the efficiency of a [1, 1] scheme.

We must also examine the time-complexity $E_A(n)$ of the procedures we have constructed. Let $E(n, d)$ be the time-complexity for $A(d, q_d, p_d)$, where the A's are generated by a scheme $[k_i, l_i]$ $(i=1, 2, \cdots, r)$. Since $A(d-1, q_{d-1}, p_{d-1})$ is used $c=\sum_{i=1}^{r}(k_i+l_i)$ times, and is never applied to more than n functions,

$$E(n, d) \leqq cE(n, d-1)+Knd \tag{2.17}$$

for some constant K. Note that $q_d=c^{d-1}$.

If we suppose K is large enough that $E(n, 1) \leqq Kn$, then (2.17) has solution

$$E(n, d) \leqq Kc^{d-1}dn = Kqdn \tag{2.18}$$

as may be verified by induction. Thus $E(n, d)=O(qdn)$. In specific cases, by a more careful analysis, we can lower the bound on $E(n, d)$ somewhat, but (2.18) is sufficient for most purposes. It is, in fact, better than Megiddo's estimate [11, p. 121] for his own procedure. His value is $2^{O(2^d)}n$, although it is not clear in [11] how he obtains this estimate. From the above it is clear that his scheme only requires $O(d2^{d-1}n)$ effort. (In fact, we believe that a bound of only $O(nd^2)$ can be derived for Megiddo's scheme.)

Finally, it is possible to improve slightly on the growth of q from 9^{d-1}, while still guaranteeing $p=\frac{1}{2}$. This can be done by using a [2, 2], [2, 2] scheme for odd $d>1$, and a [2, 2], [2, 3] scheme only for even d. It may then be checked that this guarantees $p=\frac{1}{2}$ with only $q \leqq 9(\sqrt{72})^{d-2}$ enquiries.

2.4. Lower bounds. In the above we have used algebraic, rather than geometric, terminology. We can, as did Megiddo [11], cast the problem in the geometric form of locating x_0 with respect to hyperplanes $h_i(x)=0$ and the half-spaces they determine. In this section we will adopt this terminology, since it provides a more natural language for the proof of the result given below. However, it may be observed that there is a dual geometric interpretation of the search problem, in terms of known points and an unknown hyperplane, which is equally intuitive.

For procedures $A(d, q, p)$, we might ask what minimum number of enquiries q is necessary to achieve some given p. The methods of § 2.3 give an exponential upper bound on q as a function of d. The following gives a much weaker lower bound, that q must increase at least linearly with d.

PROPOSITION 2.1 *For* $0<p<1$, $q \geqq d \max\{1, -\log_2(1-p)\}$.

Proof. First we prove $q \geqq d$. Suppose the n hyperplanes are in general position in $\mathbb{R}^d$. If $q<d$, then x_0 is only located within a convex polyhedron which contains at least one infinite line. This line will intersect all but, at most, $(d-1)$ hyperplanes in a single point. Thus x_0 cannot be located with respect to more than $(d-1)$ hyperplanes. Since $(d-1)/n \to 0$ as $n \to \infty$, we have the result.

To prove $q \geqq -d \log_2(1-p)$, we note that $A(d, q, p)$ is a "linear decision tree", in the terminology of Dobkin and Lipton [4]. This tree has at most 2^q leaves, which partition $\mathbb{R}^d$ into disjoint convex polyhedra. Now the n hyperplanes also partition $\mathbb{R}^d$ into $\sum_{i=0}^{d}\binom{n}{i}$ convex polyhedra. We will call these regions, to distinguish them from the polyhedra, which are the leaves of the decision tree. Now each polyhedron fails to intersect at least $\lceil pn \rceil$ hyperplanes, since x_0 is located with respect to at least this number of hyperplanes. Thus, even if the polyhedron intersects all the $\sum_{i=0}^{d}\binom{n-\lceil pn \rceil}{i}$ regions formed by the remaining $(n-\lceil pn \rceil)$ hyperplanes, it can intersect at most this

number of the regions formed by the n hyperplanes. However, both the regions and the polyhedra fill $\mathbb{R}^d$, so we must have

$$2^q \sum_{i=0}^{d} \binom{n-\lceil pn \rceil}{i} \geqq \sum_{i=0}^{d} \binom{n}{i}, \quad \text{i.e.} \quad 2^q \geqq \sum_{i=0}^{d} \binom{n}{i} \Big/ \sum_{i=0}^{d} \binom{n-\lceil pn \rceil}{i}. \tag{2.19}$$

Letting $n \to \infty$, for fixed d, the right-hand side of (2.19) is easily shown to tend to the limit $(1-p)^d$. Thus

$$2^q \geqq (1-p)^{-d}. \tag{2.20}$$

Taking logs to base 2 in (2.20) gives the result.

Even the weak bound $q \geqq d$ has implications for the application of the search problem to algorithms where equations like (2.10) determine their time-complexity. If we look for a solution to (2.10) of the form $T(n, d) \leqq C(d)n^k$, for any constant k, we must have

$$C(d)n^k \geqq C(d)(1-\alpha p)^k n^k + dC(d-1)n^k, \quad \text{since } rq \geqq d.$$

That is, $C(d) \geqq dC(d-1)/(1-(1-\alpha p)^k) > dC(d-1)$. Thus $C(d) = \Omega(d!)$.

Therefore the constant must grow super-exponentially with d. Thus, for example, the approach of [11] is incapable of providing an algorithm for linear programming which is even $O(A^d n^k)$ for any constants A, k. It is therefore highly unlikely that this approach can lead to a "genuinely" polynomial algorithm for linear programming [11], other than for very slowly growing values of d.

3. The Euclidean one-centre problem.

3.1. Introduction. The weighted Euclidean one-centre problem has been considered by various authors. See, for example, [5], [8], [12]. The problem may be formulated as follows:

$$\begin{aligned} &\text{Determine } c \text{ such that } F(c) = \min_x F(x) \\ &\text{where } F(x) = \max_{1 \leqq i \leqq n} w_i^2 (v_i - x)^2, \end{aligned} \tag{3.1}$$

where the v_i are n points in $\mathbb{R}^d$ with weights $w_i > 0$, and $c \in \mathbb{R}^d$ is the centre to be selected. (Note $z^2 \equiv z^T z$, $z \in \mathbb{R}^d$.)

It is well known that F is a convex function of x [5], and this property enables efficient algorithms for (3.1) to be constructed. In the planar case $d = 2$, to which most attention has been directed in view of its application to distribution problems, it is straightforward to give $O(n^3)$ time algorithms, but it is possible to do much better than this. Megiddo has developed an $O(n \log^2 n)$ algorithm, and Megiddo and Zemel an $O(n \log n)$ time randomised algorithm. (See [10], [12], [13].) Here we improve these results by exhibiting an algorithm which is $O(n)$ in any fixed dimension d, although the constant grows as fast as $O(3^{(d+2)^2})$. The algorithm applies the results of § 2, and techniques developed by the author [6], and independently by Megiddo [10] for other problems involving convexity. In [10] Megiddo has shown that the planar *unweighted* one-centre problem (i.e. when all w_i are equal) can be solved in $O(n)$ time, and his method extends to higher dimensions using the techniques of [11], or better § 2 above. His method does not however extend to the weighted problem, since he uses special properties of the unweighted problem. However, the present section may be viewed as a generalisation of his technique.

3.2. The algorithm. The method is iterative. At each iteration, the size of the problem is reduced by a fixed proportion. To control the iterations, we use the search technique of § 2. The algorithm is rather complicated, and thus will only be outlined rather than described formally. Its justification and time-analysis will be presented along with its development. We do not attempt to minimise the constant factor for the time bound in any dimension, but merely note that such improvements can be made, for example by maximizing the efficiency of the search procedure.

We first rewrite (3.1) as

$$F(x) = \max_i (w_i^2 x^T x - 2w_i^2 v_i^T x + w_i^2 v_i^T v_i).$$

We now express F in a more "linear" form, by introducing an additional variable x_{d+1}, and write $y = (x_1, x_2, \cdots, x_{d+1})^T$. Thus $y \in \mathbb{R}^{d+1}$. Then minimising $F(x)$ is equivalent to solving the following mathematical programming problem:

$$\begin{aligned} &\min_y G(y) = \max_i (w_i^2 x_{d+1} - 2w_i^2 v_i^T x + w_i^2 v_i^2) \\ &\text{subject to} \quad x^T x - x_{d+1} \leqq 0. \end{aligned} \tag{3.2}$$

The equivalence follows easily from the fact that the constraint must obviously be binding in any optimal solution. Now (3.2) may be regarded as a special case of the following, more general, problem:

$$\begin{aligned} &\min_y G(y) = \max_{1 \leqq i \leqq n} (a_i^T y + b_i) \\ &\text{subject to} \quad f(y) = \tfrac{1}{2} y^T C y + a^T y + b \leqq 0 \end{aligned} \tag{3.3}$$

where C is a symmetric positive semidefinite matrix, the a's are constant vectors and the b's numerical constants. In (3.2) we will have, partitioning matrices on their $(d+1)^{th}$ position:

$$C = \left[\begin{array}{c|c} 2I_d & 0 \\ \hline 0 & 0 \end{array}\right], \quad a = \left[\begin{array}{c} 0 \\ \hline -1 \end{array}\right], \quad b = 0,$$

$$a_i = \left[\begin{array}{c} -2w_i^2 v_i \\ \hline w_i^2 \end{array}\right], \qquad b_i = w_i^2 v_i^2.$$

We will also write $g_i(y) = a_i^T y + b_i$, so that $G(y) = \max_i g(y)$.

Now (3.3) is a convex programming problem. This follows directly from the facts that:

(i) any affine function is convex,
(ii) a positive semidefinite quadratic form is convex,
(iii) a maximum of convex functions is convex,
(iv) a sum of convex functions is convex.

Note that (3.3) is very similar to a convex quadratic programming problem, in that such a problem has the form

$$\min_y f(y) \quad \text{subject to } G(y) \leqq 0,$$

in the above notation. From this relationship, and the arguments relating convexity and polynomial solvability developed in [7], it would seem to follow that, to any specified length of approximation to the solution, there exists a polynomial time algorithm for the weighted Euclidean one-centre problem. (Chandrasekaran [3] has

arrived at the same conclusion using a different formulation.) The qualification concerning length of approximation is necessary since, as shown below, the solution point may be irrational for rational data. We conjecture further that it may be possible to give a polynomial time algorithm to determine the solution exactly, if the square-root symbol is allowed, using the methods in [9]. However, we will not pursue these issues here, concentrating instead on the problem in fixed-dimensional space.

The advantage of (3.3) over (3.1) is that the objective function has been "linearised". However, observe that to achieve this we have had to introduce a nonlinear constraint and move up one dimension. Thus, for convenience, we will redefine d for the moment so that (3.3) is in $\mathbb{R}^d$, i.e. we set $d \leftarrow d+1$.

We now consider how to solve (3.3). Let y_0 be its optimal solution. We use the observation [6], [10] that for any $1 \leqq i, j \leqq n$, if $g_i(y_0) > g_j(y_0)$, then g_j can be removed from the definition of G without affecting its optimal solution.

First, we divide the n affine functions g_j arbitrarily into $\lfloor \frac{1}{2}n \rfloor$ pairs. Let i, j be any such pair, and let $h_{ij}(y) = g_i(y) - g_j(y)$. Now we attempt to determine the sign of $h_{ij}(y_0)$ for some proportion of the h_{ij}. If $h_{ij}(y_0) > 0$, then g_j can be deleted, and if $h_{ij}(y_0) < 0$, then g_i can be deleted. If $h_{ij}(y_0) = 0$, then it follows that the problem can be reduced to one of the same form in the (lower-dimensional) space in which $h_{ij} \equiv 0$. In this space $g_i \equiv g_j$, and hence either can be deleted. (In fact, it will follow that, if this case arises, the problem has already been solved.) In all cases, at least one of g_i, g_j can be deleted if the sign of $h_{ij}(y_0)$ is known.

Using the ideas of § 2, an enquiry is then a problem of the form: for a given affine function $h(y)$, determine the sign of $h(y_0)$. Then it follows that $9^{d-1} = 3^{2(d-1)}$ such enquiries are sufficient to determine the sign of at least half of the h_{ij}. Thus at least $\frac{1}{2}\lfloor \frac{1}{2}n \rfloor \geqq \frac{1}{6}n$ of the g's can be deleted, and the procedure can be iterated, while $n > 1$. Eventually, we will reach a point where G is defined by a single g. We will deal with this case below. Also, we show below that each enquiry can be achieved by solving at most three problems of the form (3.3) in $\mathbb{R}^{d-1}$. In addition there will be some manipulations associated with each enquiry, which can be time-bounded by Knd for some constant K. (See § 2.) Therefore, let $T(n, d)$ be the time to solve (3.3) by this algorithm. We have (c.f. (2.10)),

$$T(n, d) \leqq T(\tfrac{5}{6}n, d) + 3 \times 3^{2(d-1)}\{T(n, d-1) + Knd\}. \tag{3.4}$$

Consider $T(n, 0)$. In this case, solving (3.3) amounts to checking whether $b \leqq 0$, and finding the maximum of the n values b_i. This can clearly be done in $O(n)$ time. Let us assume that K is large enough that $T(n, 0) < 3Kn$. Inductively assume that $T(n', d') < K3^{(d'+1)^2}n'$ for all pairs $(d', n') < (d, n)$ in lexicographic order. Then, from (3.4),

$$\begin{aligned} T(n, d) &\leqq \tfrac{5}{6}K3^{(d+1)^2}n + 3^{2d-1}(K3^{d^2}n + Knd) \\ &= K3^{(d+1)^2}(5/6 + 1/9 + d/3^{d^2+2})n \\ &\leqq K3^{(d+1)^2}(5/6 + 1/9 + 1/27)n \\ &= \tfrac{53}{54}K3^{(d+1)^2}n \\ &< K3^{(d+1)^2}n. \end{aligned} \tag{3.5}$$

Thus the inductive hypothesis is maintained, and we may conclude that for all n, d, $T(n, d) = O(3^{(d+1)^2}n)$.

We observe that the algorithm is, in fact, polynomial for $d = O(\sqrt{\log n})$, in addition to being linear-time in any fixed dimension. The constant grows rather fast, but less quickly than the constants of order $2^{O(2^d)}$ in [11].

We still have to verify the correctness of some assertions made above. Let us first consider the solution of (3.3) when $n=1$, which occurs at the termination of the iterative cycle, i.e.

$$\begin{aligned} &\min && a_1^T y + b_1 \\ &\text{subject to} && \tfrac{1}{2} y^T C y + a^T y + b \leqq 0. \end{aligned}$$

By rational operations, using a "completing the square" process (see, e.g., [1]), this can be reduced to a problem of the form:

$$\begin{aligned} &\min && \alpha_1^T y_1 + \alpha_2^T y_2 + \beta \\ &\text{subject to} && \tfrac{1}{2} y_1^T C_1 y_1 + \gamma^T y_2 + \delta \leqq 0 \end{aligned} \tag{3.6}$$

in variables (y_1, y_2) which are obtained by a nonsingular affine transformation from y. Here C_1 is a diagonal positive definite matrix. Now (3.6) is obviously infeasible if and only if $\delta > 0$ and $\gamma = 0$. (If the condition is satisfied, no solution exists because C_1 is positive definite. If it is not satisfied then, if $\delta \leqq 0$, $y_1 = y_2 = 0$ is feasible, otherwise $y_1 = 0$, $y_2 = -\delta\gamma/\gamma^2$ is feasible.) Thus we can easily check feasibility, and obtain a feasible point. Now, if $\alpha_1 = \alpha_2 = 0$, any feasible point is optimal in (3.6). Otherwise, putting $y_1 = 0$, we have

$$\begin{aligned} &\min && \alpha_2^T y_2 + \beta \\ &\text{subject to} && \gamma^T y_2 + \delta \leqq 0. \end{aligned}$$

This is clearly unbounded below unless either (i) $\alpha_2 = 0$ or (ii) $\alpha_2 = -\mu\gamma$ for some $\mu > 0$.

Consider case (ii) first. Let $t = -(\gamma^T y_2 + \delta)$. Then (3.6) becomes

$$\begin{aligned} &\min && \alpha_1^T y_1 + \mu t + \beta_1 \\ &\text{subject to} && \tfrac{1}{2} y_1^T C_1 y_1 \leqq t. \end{aligned}$$

The solution to this, since $\mu > 0$, is clearly given by $t = \frac{1}{2} y_1^T C_1 y_1$. Thus the problem reduces to

$$\min\ (\alpha_1^T y_1 + \tfrac{1}{2}\mu y_1^T C_1 y_1 + \beta_1).$$

By elementary calculus, this has solution $y_1 = -\mu^{-1} C_1^{-1} \alpha_1$. We then choose any y_2 to satisfy

$$-(\gamma^T y_2 + \delta) = t = \tfrac{1}{2}\mu^{-2} \alpha_1^T C_1^{-1} \alpha_1.$$

It remains to consider case (i). Note that so far we require only rational operations. We have $\alpha_2 = 0$. Now unless $\gamma = 0$, (3.6) is unbounded below, since we can choose, for example, $y_1 = -M\alpha_1$, $y_2 = -(\frac{1}{2} M^2 \alpha_1^T C_1 \alpha_1 + \delta)\gamma/\gamma^2$ for arbitrarily large M. Thus we may assume $\gamma = 0$, and hence $\delta \leqq 0$. Now (3.6) has the form

$$\begin{aligned} &\min && \alpha_1^T y_1 + \beta \\ &\text{subject to} && \tfrac{1}{2} y_1^T C_1 y_1 + \delta \leqq 0. \end{aligned}$$

The constraint determines a bounded region for y_1, since C_1 is positive definite. Thus the constraint must be binding (since $\alpha_1 \neq 0$) in the optimal solution. Thus the problem reduces to

$$\begin{aligned} &\min && \alpha_1^T y_1 + \beta \\ &\text{subject to} && \tfrac{1}{2} y_1^T C_1 y_1 = -\delta. \end{aligned}$$

Again using elementary calculus, it follows that this problem has the solution $y_1 = \lambda C_1^{-1}\alpha_1$ where λ satisfies

$$\lambda^2 = -2\delta/\alpha_1^T C_1^{-1}\alpha_1. \tag{3.7}$$

The value of y_2 is arbitrary.

Up to this point, only rational operations have been needed. However, the computation of y_1 requires the square-root function to determine λ from (3.7). For the moment we will assume this is available, but we will return to this below. Thus, to summarise the above discussion, when $n = 1$ we can readily decide the feasibility, boundedness, and optimal solution of (3.3).

To complete the description of the algorithm, we have to consider the enquiry process in more detail. We will assume that (3.3) is feasible, and that we know a feasible point y^*. This can be achieved by the same process used in connection with (3.6) above. (If (3.3) is infeasible, there is clearly nothing more to do.)

Now, given an affine function $h(y)$, if $h(y)$ is a constant then we can determine sign (h) immediately. Otherwise, we first solve the problem (3.3) subject to the additional constraint $h(y) = 0$. By using this affine constraint to eliminate any variable having a nonzero coefficient in its description, we can reduce (3.3) to a problem of identical form in $(d-1)$ variables. (Note that positive semidefiniteness is preserved, trivially, by such a substitution.) We recursively solve this lower-dimensional problem by our algorithm, basing the recursion on the case $d = 0$ discussed above. If it has an unbounded solution, then so, obviously, does (3.3). There is then nothing more to do. If it is infeasible, then $h(y_0)$ obviously has the same sign as $h(y^*)$, since the feasible set is convex. Thus sign(h) can be determined in a further $O(d)$ time. Otherwise, this problem has a bounded optimal solution at a value y_1, say, with $h(y_1) = 0$. Now, by convexity, all feasible points to (3.3) with $G(y) < G(y_1)$ will have the same sign for $h(y)$. Moreover, if there is any such point, there are points arbitrarily close to y_1. Thus we look for a point $y_2 = y_1 + \varepsilon p$ where p, which is subject to an arbitrary normalisation, is a "direction of descent", and $\varepsilon > 0$ can be chosen arbitrarily small.

Now let I be the set of values of i such that $g_i(y_1) = G(y_1)$. Then, for ε small enough, it follows that

$$G(y_2) = \max_{i \in I} (a_i^T y_2 + b_i) = G(y_1) + \varepsilon \max_{i \in I} (a_i^T p).$$

Thus $G(y_2) < G(y_1)$ if and only if $\max_{i \in I} (a_i^T p) < 0$.

If $f(y_1) < 0$, this is the condition that such a p exists. However, if $f(y_1) = 0$, we must ensure that p is directed into the feasible region. Now

$$\begin{aligned} f(y_2) &= y_2^T C y_2 + a^T y_2 + b = f(y_1) + \varepsilon\{\tfrac{1}{2}\varepsilon p^T C p + (a + Cy_1)^T p\} \\ &= \varepsilon\{\tfrac{1}{2}\varepsilon p^T C p + (a + Cy_1)^T p\}. \end{aligned}$$

So $f(y_2) \leqq 0$ if and only if, for arbitrarily small ε,

$$\tfrac{1}{2}\varepsilon p^T C p + (a + Cy_1)^T p \leqq 0. \tag{3.8}$$

This condition can clearly only be satisfied if $(a + Cy_1)^T p \leqq 0$.

We consider three cases. These give rise to subproblems for checking the existence of p. Note that we will never have to solve more than two of them in order to do this. With the determination of y_1, this gives a total of three, as asserted.

(i) $f(y_1) < 0$, or $a = 0$, $C = 0$. Then p exists if and only if there is a solution to

$$\max_{i \in I} (a_i^T p) < 0.$$

We may normalise p arbitrarily so that, for some $j \in I$, $a_j^T p = -1$. Clearly we cannot

have $a_j = 0$, thus we may eliminate one variable using this equation. We obtain a problem of the form (3.3) in $(d-1)$ variables, but having a trivial constraint (i.e. C, a, b are all zero). In fact, this problem is simply a linear programming problem, but for symmetry we treat it as a special case of (3.3).

(ii) Case (i) does not hold, but $(a + Cy_1)^T p = 0$. Then clearly $p^T Cp = 0$, which is equivalent to $Cp = 0$. Thus $a^T p = 0$. Thus there is a direction of descent of this form if and only if there is a solution to

$$\max_{i \in I} (a_i^T p) < 0 \quad \text{such that } Cp = 0, \text{ and } a^T p = 0.$$

We can use the equations to eliminate at least one variable. We again obtain a problem of the form (3.3) with a trivial constraint.

(iii) Case (i) does not hold, and $(a + Cy_1)^T p < 0$. Thus we may normalise p so that $(a + Cy_1)^T p = -1$. Then the constraint (3.8) is obvously satisfied for any ε if $p^T Cp = 0$, or otherwise for all $\varepsilon < 2/p^T Cp$. Thus there is a direction of descent of this form if and only if there is a solution to

$$\max_{i \in I} (a_i^T p) < 0 \quad \text{such that } (a + Cy_1)^T p = -1.$$

Again the equation can be used to eliminate a variable. (Note that if $a + Cy_1 = 0$ this case cannot occur.) Once again we obtain a lower-dimensional problem of the form (3.3) with a trivial constraint.

Solving the subproblems described in (i), (ii) and (iii) allows us to decide the existence of a direction of descent, and to find one if it exists. If none exists, then clearly y_1 is also optimal without the constraint $h(y) = 0$. Thus we may take $y_0 = y_1$ and stop. Otherwise, suppose $h(y) = \alpha^T y + \beta$. Then

$$h(y_2) = \alpha^T (y_1 + \varepsilon p) + \beta = h(y_1) + \varepsilon \alpha^T p = \varepsilon \alpha^T p \neq 0,$$

since y_1 is optimal subject to $h(y) = 0$. Therefore we have

$$\text{sign } (h(y_0)) = \text{sign } (h(y_2)) = \text{sign } (\alpha^T p),$$

and we can readily determine sign (h) as required.

This completes the description of the algorithm. However, there is one more point which requires attention. We have not described a *rational* algorithm for this problem, since we have, possibly, to compute square-roots in (3.7). But note that it is only in the subproblem of case (iii) above that such an irrationally-derived quantity would enter within the algorithm, since this involves y_1. Since this subproblem has a trivial constraint, however, no further irrational operations would be needed in its solution.

Let S be the set of all the real-numerical values in the a's, b's and C. Then a rational algorithm means that we must compute wholly within the ordered field $R = Q(S)$, where Q is the field of rational numbers, and we are using the notation for a field extension given, for example, in [1]. In order to solve the subproblems referred to above, we wish occasionally to compute in $R(\lambda)$, where $\lambda^2 \in R$ (i.e. in a quadratic extension of R). However, we can simulate the computations of $R(\lambda)$ in R by using ordered pairs $(r_1, r_2) \equiv r_1 + \lambda r_2$, where $r_1, r_2 \in R$. It is well-known algebra [1] that the field operations for $R(\lambda)$ can be conducted using these ordered pairs. It is also straightforward to show that comparisons in $R(\lambda)$ can be simulated by a mixture of field operations and comparisons on the elements of such ordered pairs. Moreover, each operation in $R(\lambda)$ is simulated by $O(1)$ operations in R. It now follows that we can construct a completely rational algorithm for solving (3.3) which has the same time-bound up to a constant factor independent of d. The output from this algorithm will, in general, be a d-vector of ordered pairs (r_1, r_2) together with a number λ^2. Then

y_0 is the corresponding d-vector of real numbers $r_1+\lambda r_2$. We must either accept the solution in this form, or compute y_0 after making a single call to a square-root function.

To summarise the discussion, we have shown that we can construct a rational algorithm, with time-complexity (3.5), for solving (3.3) provided that we accept the solution values in the form $r_1+\lambda r_2$, where the r's and λ^2 are rational. Note that, for the Euclidean one-centre problem, the $(d+1)$ in the exponent of (3.5) must be replaced by $(d+2)$, since we redefined the value of d in (3.3).

Finally, to show that the solution of (3.3) can be irrational in terms of the data, consider the following Euclidean one-centre problem in the plane $\mathbb{R}^2$. Let $v_1=(0,0)$, $v_2=(1,0)$, $v_3=(0,1)$ with weights $w_1=9$, $w_2=w_3=8$. Then it is straightforward to show that the optimal Euclidean one-centre is the point $\lambda(1,1)$, where $\lambda=8/(8+7\sqrt{2})$. Clearly all the data is rational, but the solution involves the irrational number $\sqrt{2}$. Thus the solution value could not be obtained by a purely rational algorithm. Note that the difficulty does not arise in the unweighted case, since then it is easy to show the algorithm is always rational, since irrational λ never occur. In this case, the algorithm is close to Megiddo's [10], [11] algorithm for the unweighted case. Thus Megiddo's algorithm may be viewed as a special case of the one given here.

Note. During revision of this paper, I learned that K. Clarkson has obtained an improvement of Megiddo's linear programming algorithm similar to the main result of § 2 of this paper. In the notation of § 2, he proposes a [4, 5] scheme. This achieves the same number of enquiries as the [2, 2], [2, 3] scheme given here, but with a rather worse (though nonzero) lower bound for p. I am grateful to Nimrod Megiddo and one of the referees for bringing Clarkson's work to my attention.

Acknowledgment. I am grateful to Alan Frieze for discussions concerning the content of this paper.

REFERENCES

[1] G. BIRKHOFF AND S. MACLANE, *A Survey of Modern Algebra*, MacMillan, New York, 1977 (fourth edition).

[2] M. BLUM, R. W. FLOYD, V. PRATT, R. L. RIVEST AND R. E. TARJAN, *Time bounds for selection*, J. Comput. System Sci., 7 (1973), pp. 448-461.

[3] R. CHANDRASEKARAN, *The weighted Euclidean 1-center problem*, Oper. Res. Lett., 1 (1982), pp. 111-112.

[4] D. P. DOBKIN AND R. J. LIPTON, *On the complexity of computations under varying sets of primitives*, Lecture Notes in Computer Science, 33, Springer-Verlag, New York, 1975, pp. 110-117.

[5] Z. DREZNER AND G. O. WESOLOWSKY, *Single facility l_p-distance minimax location*, SIAM J. Alg. Disc. Meth., 1 (1980), pp. 315-321.

[6] M. E. DYER, *Linear time algorithms for two- and three-variable linear programs*, this Journal, 13 (1984), pp. 31-45.

[7] M. GRÖTSCHEL, L. LOVÁSZ AND A. SCHRIJVER, *The ellipsoid method and its consequences in combinatorial optimisation*, Combinatorica, 1 (1981), pp. 169-197.

[8] D. HEARN AND J. VIJAY, *Efficient algorithms for the (weighted) minimum circle problem*, Oper. Res., 30 (1982), pp. 777-795.

[9] R. KANNAN, A. LENSTRA AND L. LOVÁSZ, *Polynomial factorization and non-randomness of bits of algebraic and transcendental numbers*, Proc. 16th Annual ACM Symposium on Theory of Computing, 1984, pp. 191-200.

[10] N. MEGIDDO, *Linear-time algorithms for linear proramming in R^3 and related problems*, this Journal, 12 (1983), pp. 759-776.

[11] ———, *Linear programming in linear time when the dimension is fixed*, J. Assoc. Comput. Mach., 31 (1984), 114-127.

[12] ———, *The weighted Euclidean 1-centre problem*, Math. Oper. Res., 8 (1983), pp. 498-504.

[13] E. ZEMEL, *A linear time randomized algorithm for local roots and optima of ranked functions*, J. L. Kellogg Graduate School of Management, Northwestern Univ. Evanston, IL, 1983.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
008

SPARSE SETS, LOWNESS AND HIGHNESS*

JOSÉ L. BALCÁZAR†, RONALD V. BOOK‡ AND UWE SCHÖNING§

Abstract. We develop the notions of "generalized lowness" for sets in PH (the union of the polynomial-time hierarchy) and of "generalized highness" for arbitrary sets. Also, we develop the notions of "extended lowness" and "extended highness" for arbitrary sets. These notions extend the decomposition of NP into low sets and high sets developed by Schöning [15] and studied by Ko and Schöning [9].

We show that either every sparse set in PH is generalized high or no sparse set in PH is generalized high. Further, either every sparse set is extended high or no sparse set is extended high. In both situations, the former case corresponds to the polynomial-time hierarchy having only finitely many levels while the latter case corresponds to the polynomial-time hierarchy extending infinitely many levels.

Key words. polynomial-time hierarchy, generalized lowness and highness, extended lowness and highness, sparse sets

AMS (MOS) subject classifications. 68Q15, 03D15

Introduction. Recent studies on the structure of intractable sets have shown that the notion of NP-completeness is not compatible with the notion of being sparse unless the polynomial-time hierarchy has only finitely many levels. Berman [4] showed that tally sets cannot be $\leqq_m^P$-complete for NP unless P = NP. Fortune [6] showed that co-sparse sets cannot be $\leqq_m^P$-complete for NP unless P = NP. Berman and Hartmanis [5] had conjectured that sparse sets cannot be $\leqq_m^P$-complete for NP unless P = NP and Mahaney [13] showed that this conjecture was true. Other results due to Long [10] and to Karp and Lipton [7] showed that if there exist sparse sets with certain properties of being hard or complete with respect to certain reducibilities, then the polynomial-time hierarchy "collapses," that is, $\mathrm{PH} = \Sigma_n^P$ for some n.

In [15] Schöning developed a new approach to such questions. He considered a decomposition of the class NP which depends on the number of distinct levels in the polynomial-time hierarchy. Call a set A in NP "low" if for some n, $\Sigma_n^P(A) \subseteq \Sigma_n^P$, and call a set B in NP "high" if for some n, $\Sigma_{n+1}^P \subseteq \Sigma_n^P(B)$. Thus, if A is low, then with respect to the operator Σ_n^P, A does not encode the power of a quantifier, but if A is high, then A does encode the power of a quantifier. It is easy to see that if there is a set in NP that is both high and low, then the polynomial-time hierarchy collapses. On the other hand, if the polynomial-time hierarchy does not collapse, then the collection of low sets and the collection of high sets are disjoint, and, furthermore, there are sets in NP that are neither high nor low. The structure of the low and high hierarchies in NP reflects the structure of those sets in $\mathrm{NP} - P$ that are not $\leqq_m^P$-complete for NP. Ko and Schöning [9] have shown that the sets in NP with polynomial-size circuits (hence, also sparse sets in NP) are low in NP. Thus, if a set in NP with polynomial-size circuits is high, then the polynomial-time hierarchy collapses.

* Received by the editors August 16, 1984, and in revised form April 1, 1985. This research was supported in part by the U.S.A.-Spanish Joint Committee for Educational and Cultural Affairs, by the Deutsche Forschungsgemeinschaft, and by the National Science Foundation under grant DCR83-12472. Some of the these results were reported at the Symposium on Mathematical Foundations of Computer Science, Praha, September 1984.

† Facultat d'Informàtica, Universitat Politècnica de Barcelona, Jordi Girona Salgado, 31, 08034 Barcelona, Spain.

‡ Department of Mathematics, University of California, Santa Barbara, California 93106.

§ Institut für Informatik, Universität Stuttgart, Azenbergstrasse 12, D-7000 Stuttgart 1, West Germany.

The arguments of Ko and Schöning regarding lowness depend heavily on the fact that the sets under consideration are in NP. Here we consider notions of "low" and "high" that do not depend on the set being in NP. Call a set A "generalized low" if there exist i and j with $i \leqq j$ such that $\Sigma_i^P(A) \subseteq \Sigma_j^P$; clearly, a generalized low set is in some level of the polynomial-time hierarchy. Every set in $\mathrm{PH} = \bigcup_{i \geqq 0} \Sigma_i^P$ is generalized low since $A \in \Sigma_i^P$ implies $\Sigma_i^P(A) \subseteq \Sigma_{2i}^P$. Call a set A "generalized high" if there exist k and l with $k > l$ such that $\Sigma_k^P \subseteq \Sigma_l^P(A)$. Notice that if A is $\leqq_m^P$-complete for Σ_j^P, then $\Sigma_{j+1}^P = \Sigma_1^P(A)$ so that A is generalized high. It is clear that if the polynomial-time hierarchy collapses, then every set in PH is generalized high since $\mathrm{PH} = \Sigma_k^P$ implies that if $A \in \mathrm{PH}$, then $\Sigma_{k+i}^P = \Sigma_k^P(A)$ for all $i \geqq 0$. On the other hand, if the polynomial-time hierarchy does not collapse, then the structure of the class of generalized low sets in PH reflects the structure of $\mathrm{PH} - \{L \mid$ for some i, L is $\leqq_m^P$-complete for $\Sigma_i^P\}$. Clearly, every set in PH is both generalized low and generalized high if the polynomial-time hierarchy collapses. It is shown here that if there is a set that is simultaneously generalized low and generalized high, then the polynomial-time hierarchy collapses. Thus, either every set in PH is both generalized low and generalized high or no set in PH is both generalized low and generalized high.

In the earlier studies [9], [15] subsets of NP with certain quantitative properties were considered (e.g., sparse sets, sets with polynomial-size circuits). Thus we are led to restrict attention to sets in PH with these same quantitative properties. We are interested in finding classes $\mathscr{C}_1$ of sets in PH such that if every set in $\mathscr{C}_1$ is generalized high, then the polynomial-time hierarchy collapses, and classes $\mathscr{C}_2$ of sets in PH such that if one set in $\mathscr{C}_2$ is generalized high, then the polynomial-time hierarchy collapses (so as noted above, every set $\mathscr{C}_2$ is generalized high). Here we focus on the class of sparse sets in PH. One of our main results is that either *every* sparse set in PH is generalized high or *no* sparse set in PH is generalized high. The former case corresponds to the notion of the polynomial-time hierarchy collapsing while the latter corresponds to the notion that there are infinitely many levels in the polynomial-time hierarchy. The same result holds for the class of sets with polynomial-size circuits.

The notion of "generalized low" lifts the idea of "low" sets in NP to sets with similar properties in PH. One can lift this notion to arbitrary sets, not just those in PH, if one has a way of comparing them with respect to NP. This is done in the following way.

Call a set A "extended low" if for some n, $\Sigma_n^P(A) \subseteq \Sigma_{n-1}^P(A \oplus \mathrm{SAT})$. Call a set A "extended high" if for some n, $\Sigma_n^P(A \oplus \mathrm{SAT}) \subseteq \Sigma_n^P(A)$. We show that every sparse set (whether or not it is in PH) is extended low. Further, if any sparse set is extended high, then the polynomial-time hierarchy collapses. Another of our main results is that either *every* sparse set is extended high or *no* sparse set is extended high. Again, the former case corresponds to the collapse of the polynomial-time hierarchy which the latter case corresponds to that hierarchy extended to infinitely many levels.

Our purpose here is to understand the role that structural notions such as sparsity, having polynomial-size circuits, etc., have in determining the underlying structure of complexity classes, with emphasis on the properties of feasibly computable sets. We believe that our results shed new light on the role of sparse sets in determining the structure of the polynomial-time hierarchy.

2. Preliminaries. It is assumed that the reader is familiar with the basic concepts from the theories of automata, computability, and formal languages. Some of the concepts that are most important for this paper are reviewed here, and notation is established.

For a string w, $|w|$ denotes the length of w. The empty string is denoted by e, $|e| = 0$.

For a set S, $\|S\|$ denotes the cardinality of S.

It is assumed that all sets of strings are taken over some fixed alphabet Σ that includes $\{0, 1\}$. If $A \subseteq \Sigma^*$, then $\bar{A} = \Sigma^* - A$.

For sets $A, B \subseteq \Sigma^*$, the *join* of A and B is defined as $A \oplus B = \{0x | x \in A\} \cup \{1y | y \in B\}$.

For a set S and an integer $n \geqq 0$, let $S_{\leqq n}$ denote $\{x \in S | \, |x| \leqq n\}$.

Let $<$ denote any standard polynomial time computable total order defined on Σ^*. We consider $c(\cdot)$ to be an encoding function. For a finite set $S \subseteq \Sigma^*$, let $c(S)$ denote some standard encoding of S into a single string in Σ^*. We use the notation $\langle \cdot, \cdot \rangle$ to denote a polynomial-time pairing function on Σ^* and generalize it to n-tuples in the usual way. Observe that it is possible given $\langle x, c(S) \rangle$ to determine in polynomial time whether $x \in S$.

We assume standard definitions of oracle machines and relativized complexity classes specified by deterministic or nondeterministic oracle machines that are time-bounded or space-bounded. See [8], [11], [15], or [18] for details. In particular, Long [11] has studied the notion of "strong nondeterministic" reducibility. Define $A \leqq_T^{\mathrm{SN}} B$ if $A \in \mathrm{NP}(B)$ and $\bar{A} \in \mathrm{NP}(B)$. This is equivalent to $\mathrm{NP}(A) \subseteq \mathrm{NP}(B)$.

For an integer $k > 1$, a polynomial p, and a set L, we may define a set A as follows: $x \in A$ if and only if $(\exists y_1)_p \cdots (Q_k y_k)_p (\langle x, y_1, \cdots, y_k \rangle \in L)$. The quantifiers are intended to alternate between existential and universal so that if k is even, then Q_k is universal, and if k is odd, then Q_k is existential. For each i, the domain of y is bounded: $|y_i| \leqq p(|x|)$. Similarly, we may define a set B as follows: $x \in B$ if and only if $(\forall y_1)_p \cdots (Q_k y_k)_p (\langle x, y_1, \cdots, y_k \rangle \in L)$. Now Q_k is universal if k is odd and existential if k is even.

Now we review some well-known properties of the polynomial-time hierarchy. We refer the reader to the papers of Stockmeyer [17] and Wrathall [18] where the properties of this hierarchy were first described.

DEFINITION 2.1. (a) Let A be a set. Define $\Sigma_0^P(A) = \Pi_0^P(A) = \Delta_0^P(A) = P(A)$, and for each integer $i \geqq 0$, define $\Delta_{i+1}^P(A) = P(\Sigma_i^P(A))$, $\Sigma_{i+1}^P(A) = \mathrm{NP}(\Pi_i^P(A))$, and $\Pi_{i+1}^P(A) = \mathrm{co}\text{-}\Sigma_{i+1}^P(A)$. The structure $\{(\Delta_i^P(A), \Sigma_i^P(A), \Pi_i^P(A))\}$ is the *polynomial-time hierarchy relative to A*. Define $\mathrm{PH}(A) = \bigcup_{i \geqq 0} \Sigma_i^P(A)$.

(b) For each integer $i \geqq 0$, define $\Delta_i^P = \Delta_i^P(\phi)$, $\Sigma_i^P = \Sigma_i^P(\phi)$, and $\Pi_i^P = \Pi_i^P(\phi)$. The structure $\{(\Delta_i^P, \Sigma_i^P, \Delta_i^P)\}_{i \geqq 0}$ is the *polynomial-time hierarchy*. Define $\mathrm{PH} = \bigcup_{i \geqq 0} \Sigma_i^P$.

Recall that for each $k \geqq 1$, a set A is in Σ_k^P if and only if there is a set $B \in P$ and a polynomial p such that for all x, $x \in A$ if and only if $(\exists y_1)_p \cdots (Q_k y_k)_p (\langle x, y_1, \cdots, y_k \rangle \in B)$. A similar characterization of the classes Π_k^P, $k \geqq 1$, also holds.

It is clear that for every set A and every integer $i \geqq 0$, $\Delta_i^P(A) \subseteq \Sigma_i^P(A) \cap \Pi_i^P(A) \subseteq \Sigma_i^P(A) \cup \Pi_i^P(A) \subseteq \Delta_{i+1}^P(A)$. Also, for every set A and every integer $i \geqq 0$, $\Delta_{i+1}^P(A) = P(\Pi_i^P(A))$ and $\Sigma_{i+1}^P(A) = \mathrm{NP}(\Sigma_i^P(A))$. For any A and $i \geqq 1$, if $\Sigma_i^P(A) = \Pi_i^P(A)$ or $\Delta_i^P(A) = \Sigma_i^P(A)$ or $\Delta_i^P(A) = \Pi_i^P(A)$, then the polynomial-time hierarchy relative to A contains only finitely many different classes.

It is known [3] that there is a set B such that $\Sigma_1^P(B) \subsetneq \Sigma_2^P(B) \subsetneq \Sigma_3^P(B)$ but it is not known whether there is any set C such that the polynomial-time hierarchy relative to C is infinite.

Clearly, for every set A, $\mathrm{PH}(A) = \bigcup_{i \geqq 0} \Pi_i^P(A)$ and $\mathrm{PH}(A) = \bigcup_{i \geqq 0} \Delta_i^P(A)$. For the most part the definition of $\mathrm{PH}(A)$ as $\bigcup_{i \geqq 0} \Sigma_i^P(A)$ is the most useful here.

3. The decomposition of PH. There are a number of results that give conditions on NP under which the polynomial-time hierarchy collapses to Σ_n^P for some n, that

is, $PH = \Sigma_n^P$. For example, Mahaney [13] has shown that $P = NP$ if and only if there is a sparse set S that is $\leqq_m^P$-complete for NP, and Long [10] has shown that if there is a sparse set S such that $NP \subseteq P(S)$ and S is in Δ_2^P, then $PH = \Delta_2^P$. Schöning [15] developed a decomposition theory for NP that generalizes many of these results.

DEFINITION 3.1 [15]. For each integer $n > 0$, let $L_n = \{A \in NP | \Sigma_n^P(A) \subseteq \Sigma_n^P\}$ and let $H_n = \{A \in NP | \Sigma_{n+1}^P \subseteq \Sigma_n^P(A)\}$. Let $LH = \bigcup_{n>0} L_n$ and $HH = \bigcup_{n>0} H_n$.

PROPOSITION 3.2 [15]. (a) *The polynomial-time hierarchy has only finitely many levels if and only if* $LH \cap HH \neq \emptyset$. *Also,* $LH \cap HH \neq \emptyset$ *if and only if for some* $n > 0$, $L_n = H_n = LH = HH = NP$.

(b) *The polynomial-time hierarchy has infinitely many levels if and only if* $NP - (LH \cup HH) \neq \emptyset$.

One of the contributions of the present paper is to extend the techniques of [15] to develop a decomposition of the class PH that is similar to the decomposition of NP. In this section we describe this decomposition. In § 4 we study some of the conditions that cause the polynomial-time hierarchy to collapse, conditions that can be defined on classes in this decomposition.

DEFINITION 3.3. For every $i, j > 0$, define $L(i, j) = \{A | \Sigma_i^P(A) \subseteq \Sigma_j^P\}$. For every k, $l > 0$, define $H(k, l) = \{A | \Sigma_k^P \subseteq \Sigma_l^P(A)\}$.

Notice that for each $n > 0$, $L_n = L(n, n) \cap NP$ and $H_n = H(n+1, n) \cap NP$. We call sets "low" if they lie in $LH = \bigcup_{n>0} L_n$ and "high" if they lie in $HH = \bigcup_{n>0} H_n$, and we use the term "generalized lowness" to refer to properties of sets in $\bigcup_{i,j} L(i, j)$ and the term "generalized highness" to refer to properties of sets in $\bigcup_{k,l} H(k, l)$.

Consider generalized lowness. Clearly, if $A \in L(i, j)$, then $A \in \Sigma_j^P$ so that $\bigcup_{i,j} L(i, j) \subseteq PH$. On the other hand, it makes no sense to take $i > j$ when considering $L(i, j)$ since for $i > j$, $L(i, j) = \emptyset$ if $\Sigma_j^P \neq \Sigma_{j+1}^P$ and $L(i, j) = PH$ if $\Sigma_j^P = \Sigma_{j+1}^P$. Thus, we consider classes $L(i, j)$ only when $i \leqq j$.

Which lowness properties are significant? Let $A \in \Sigma_n^P$. Then for any i, $\Sigma_i^P(A) \subseteq \Sigma_i^P(\Sigma_n^P) = \Sigma_{i+n}^P$ so that if $j \geqq i + n$, we have $A \in L(i, j)$ trivially. Thus, if $A \in \Sigma_n^P$, then the lowness property $A \in L(i, j)$ has significance only if $i \leqq j < i + n$.

There are some obvious inclusion relations.

PROPOSITION 3.4. *For all* i, j, $L(i, j) \subseteq L(i+1, j+1) \cap L(i, j+1) \cap L(i-1, j)$.

Consider generalized highness. Clearly, if $k \leqq l$, then every set A satisfies $\Sigma_k^P \subseteq \Sigma_l^P(A)$ so that $H(k, l) = 2^{\Sigma^*}$. Thus, we consider classes $H(k, l)$ only when $k > l$.

Which highness properties are significant? If A is $\leqq_T^P$-complete for Σ_n^P, then $\Sigma_l^P(A) = \Sigma_{l+n}^P$ so that for $k > l + n$, $A \in H(k, l)$ implies that the polynomial-time hierarchy collapses. More precisely, if $k > l + n$, then $\Sigma_n^P \cap H(k, l) = \phi$ if $\Sigma_{l+n}^P \neq \Sigma_{l+n+1}^P$ and $H(k, l) = H(l+n, l)$ if $\Sigma_{l+n}^P = \Sigma_{l+n+1}^P$. Thus, if $A \in \Sigma_n^P$, then the highness property $A \in H(k, l)$ has significance only if $l < k \leqq l + n$.

There are some obvious inclusion relations.

PROPOSITION 3.5. *For all* i, j, $H(i, j) \subseteq H(i+1, j+1) \cap H(i, j+1) \cap H(i-1, j)$.

There are some characterizations of certain of these classes. The proofs are left to the reader.

PROPOSITION 3.6. (a) *For every* j, $L(1, j) = \Sigma_j^P \cap \Pi_j^P$.

(b) *For every* k, $H(k+1, 1) = \{A | A \text{ is } \leqq_T^{SN}\text{-hard for } \Sigma_k^P\}$.

Schöning [15] showed that the polynomial-time hierarchy collapses if and only if there exist m and n such that $L_m \cap H_n \neq \emptyset$. This is a consequence of the following fact.

THEOREM 3.7. *If* $L(i, j) \cap H(k, l) \neq \emptyset$ *and* $j + l < i + k$, *then the polynomial-time hierarchy collapses to* $\Sigma_{\max\{i, l\}+j-i}^P$.

Proof. Let $A \in L(i,j) \cap H(k,l)$. Since $A \in L(i,j)$, $\Sigma_i^P(A) \subseteq \Sigma_j^P$ so that $\Sigma^P_{\max\{i,l\}}(A) \subseteq \Sigma^P_{\max\{i,l\}+j-i}$. Since $A \in H(k,l)$, $\Sigma_k^P \subseteq \Sigma_l^P(A)$ so that $\Sigma^P_{k+\max\{i,l\}-l} \subseteq \Sigma^P_{\max\{i,l\}}(A)$. Thus, $\Sigma^P_{k+\max\{i,l\}-l} \subseteq \Sigma^P_{\max\{i,l\}+j-i}$ which implies that $\mathrm{PH} = \Sigma^P_{\max\{i,l\}+j-i}$ provided that $k-l > j-i$ or, equivalently, $j+l < i+k$. □

Consider the proof of Theorem 3.7. The crucial parameter in the argument is the difference $j-i$ or the difference $k-l$. One might define a set A to be *n-low* if there exist i and j such that both $n = j-i$ and also $A \in L(i,j)$. Similarly, one might define a set A to be *n-high* if there exist k and l such that both $n = k-l$ and also $A \in H(k,l)$. Then Theorem 3.7 can be restated as follows: For every $n \geqq 0$, no set can be both $(n+1)$-high and also n-low unless the polynomial-time hierarchy collapses.

The notion of "A is n-low" may be interpreted as setting an upper bound on the amount of information that can be encoded in A: the set A has the power of at most n alternating quantifiers or of the composition of at most n applications of the NP( ) operator. Similarly, the notion of "A is n-high" can be interpreted as setting a lower bound on the amount of information that can be encoded in A: the set A has the power of at least n alternating quantifiers, or of the composition of at least n applications of the NP( ) operator. Thus, Theorem 3.7 can be interpreted as saying that no set in PH can have the power of at most n alternating quantifiers and also have the power of at least $n+1$ alternating quantifiers (unless PH collapses to level n, i.e., $\mathrm{PH} = \Sigma_n^P$).

4. Generalized and extended lowness and sparse oracles. The property of being generalized low asserts that the usefulness of a set as an oracle set is quite restricted. If $A \in L(i,j)$ where $i \leqq j$, then A does not have the power of more than $j-i$ alternating quantifiers since $\Sigma_i^P(A) \subseteq \Sigma_j^P$. What type of sets has this property? Ko and Schöning [9] studied sets in NP that have this property. As noted in § 3, any generalized low set is in PH. We want to exhibit sets in PH that have this property.

Consider tally sets in the polynomial-time hierarchy. Notice that for any tally set T there is a polynomial-time oracle machine that on input 0^n will enumerate relative to T the set of strings in T of length at most n. Suppose that T is a tally set in Σ_k^P. By first enumerating the appropriate initial segment of T and then simulating a Σ_{k+1}^P oracle machine, one can show that $\Sigma_{k+1}^P(T) = \Sigma_{k+1}^P$. Hence, for every k, if T is a tally set in Σ_k^P, then $T \in L(k+1, k+1)$.

In a similar way (e.g., see [10]) it can be shown that if S is a sparse set and $S \in \Sigma_k^P$, then $S \in \mathrm{L}(k+1, k+1)$. We can show that if S has polynomial-size circuits and $S \in \Sigma_k^P$, then $S \in \mathrm{L}(k+2, k+2)$. However we will establish a more general theorem.

THEOREM 4.1. *For any set A, if there is a sparse set S and integers n, k with $n > k$ such that $A \in \Sigma_k^P(S)$ and $A \in \Sigma_n^P$, then A is in* $\mathrm{L}(n-k+2, n+2)$.

Proof. Since $A \in \Sigma_k^P(S)$, there are a polynomial p and a deterministic polynomial time-bounded oracle machine M_1 such that for all inputs x, $x \in A$ if and only if $(\exists y_1)_p \cdots (Q_k y_k)_p \langle x, y_1, \cdots, y_k\rangle \in \mathrm{L}(M_1, S)$. Let $B = \{\langle x, c(T)\rangle | T$ is a finite set and $(\exists y_1)_p \cdots (Q_k y_k)_p \langle x, y_1, \cdots, y_k\rangle \in \mathrm{L}(M_1, T)\}$. Clearly, B is in Σ_k^P. Since S is sparse, there is a polynomial q such that for all n, $\|S_{\leqq n}\| \leqq q(n)$.

Let $i > 0$ and consider any $C \in \Sigma_i^P(A)$. There are a polynomial r and a deterministic polynomial time-bounded oracle machine M_2 such that for all u, $u \in C$ if and only if $(\exists z_1)_r \cdots (Q_i z_i)_r \langle u, z_1, \cdots, z_i\rangle \in \mathrm{L}(M_2, A)$. There is a polynomial s (depending on p and the time bound of M_1) such that for all u, $u \in C$ if and only if there is a finite set T whose elements are of size at most $s(|u|)$ and $\|T\| \leqq q(s(u|))$ such that (i) $\langle x, c(T)\rangle \in B \Leftrightarrow x \in A$, and (ii) $(\exists z_1)_r \cdots (Q_i z_i)_r \langle u, z_1, \cdots, z_i\rangle \in \mathrm{L}(M_2, \{w | \langle w, c(T)\rangle \in B\})$. The predicate described by (i) is in Π_{n+1}^P because $A \in \Sigma_n^P$, $B \in \Sigma_k^P$ and $k < n$. The predicate described by (ii) is in $\Sigma_i^P(B) \subseteq \Sigma_i^P(\Sigma_k^P) = \Sigma_{i+k}^P$.

Thus, membership in C is described by a predicate of the form $\exists(\Pi^P_{n+1} \wedge \Sigma^P_{i+k})$ where the existential quantifier is polynomially bounded so that C is in $\Sigma^P_{\max\{n+2,\, i+k\}}$. Hence, C is in Σ^P_{n+2} if we choose $i = n-k+2$. Hence we have shown that $\Sigma^P_{n-k+2}(A) \subseteq \Sigma^P_{n+2}$ so that $A \in L(n-k+2, n+2)$. □

Observe that $A \in \mathrm{L}(n-k+2, n+2)$ for a set $A \in \Sigma^P_n$ is a "significant" lowness condition as described in § 3 since $n-k+2 \leqq n+2 < 2n-k+2$ since $k < n$. Also, notice that in the case $k=0$ and $u=1$, we have $A \in \mathrm{NP}$ and $A \in \mathrm{P}(S)$ so that $A \in \mathrm{L}(3,3)$. Since $\mathrm{NP} \cap \mathrm{L}(3,3) = \mathrm{L}_3$, we see that sets in NP with polynomial-size circuits are in L_3. This was first proved by Ko and Schöning [9].

COROLLARY 4.2. *Let A be a set that is $\leqq^{\mathrm{SN}}_T$-complete for Σ^P_n. Suppose that for some sparse set S and some $k < n$, $A \in \Sigma^P_k(S)$. Then A is in $L(n-k+2, n+2)$ and the polynomial-time hierarchy collapses to Σ^P_{n+2}.*

Proof. Since A is $\leqq^{\mathrm{SN}}_T$-complete for Σ^P_n, $\Sigma^P_{n+1} = \Sigma^P_1(A)$ so that $A \in H(n+1, 1)$. Also, A is in $\Sigma^P_n \cap \Sigma^P_k(S)$ and $k < n$ so that A is in $L(n-k+2, n+2)$ by Theorem 4.1. Thus, $A \in L(n-k+2, n+2) \cap H(n+1, 1)$ so that the polynomial-time hierarchy collapses to Σ^P_{n+2} by Theorem 3.7. □

Notice that in both Theorem 4.1 and Corollary 4.2, the condition that S be sparse could be replaced by the condition that S have polynomial-size circuits and the conclusions will still hold. A set that is in a certain level of the polynomial-time hierarchy relativized to a sparse set might be called "pseudo-sparse." This notion encompasses such properties as a set being in "almost polynomial time" (APT) [14], a set having Σ^P_k-circuits [9], a set having small generators [16], [19], and other properties that can be translated to being recognized relative to a sparse set, that is, properties that can be expressed by membership in a set in $\cup\{\mathrm{PH}(S) | S \text{ is sparse}\}$.

From one viewpoint the notion of generalized lowness should not force sets to be in PH but rather should apply to all sets with some particular property, e.g., to all sparse sets. This requires a change in the definitions. One way to accomplish this is as follows.

DEFINITION 4.3. For each $n > 0$, let $\mathrm{EL}_n = \{A | \Sigma^P_n(A) \subseteq \Sigma^P_{n-1}(A \oplus \mathrm{SAT})\}$ and let $\mathrm{EH}_n = \{A | \Sigma^P_n(A \oplus \mathrm{SAT}) \subseteq \Sigma^P_n(A)\}$. Let $\mathrm{EL} = \cup_n \mathrm{EL}_n$ and $\mathrm{EH} = \cup_n \mathrm{EH}_n$. A set is *extended low* if it is in EL and is *extended high* if it is in EH.

Let K be any set that is $\leq^P_m$-complete for PSPACE. Clearly, for all n, $\Sigma^P_n(K) = \Sigma^P_{n-1}(K \oplus \mathrm{SAT}) = \Sigma^P_n(K \oplus \mathrm{SAT}) = \mathrm{PSPACE}$. Thus, K is in $\mathrm{EL} \cap \mathrm{EH}$, so EL and EH are not disjoint. However, it is easy to see that $\mathrm{EL} \cap \mathrm{EH} \cap \mathrm{PH}$ is empty if and only if the polynomial-time hierarchy is infinite.

THEOREM 4.4. *If S is sparse, then $S \in \mathrm{EL}_3$.*

Proof. Let $B \in \Sigma^P_3(S)$. We must show that $B \in \Sigma^P_2(S \oplus \mathrm{SAT})$. Since $B \in \Sigma^P_3(S)$, there is a polynomial p and a deterministic polynomial time-bounded oracle machine M such that for all strings x, $x \in B$ if and only if $(\exists y_1)_p(\forall y_2)_p(\exists y_3)_p \langle x, y_1, y_2, y_3\rangle \in L(M, S)$. Since S is sparse, there is a polynomial q such that for all n, $\|S_{\leqq n}\| \leqq q(n)$. Thus, there is a polynomial r such that for all strings x, $x \in B$ if and only if there exists a set T of size at most $r(|x|)$ such that (i) for all u with $|u| \leqq r(|x|)$, $u \in T$ if and only if $u \in S$, and (ii) $(\exists y_1)_r(\forall y_2)_r(\exists y_3)_r \langle x, y_1, y_2, y_3\rangle \in \mathrm{L}(M, T)$. The predicate described by (i) is in $\Pi^P_1(S)$ and the predicate described by (ii) is in $\Sigma^P_3 = \Sigma^P_2(\mathrm{SAT})$. Thus, membership in B is described by a predicate of the form $\exists(\Pi^P_1(S) \wedge \Sigma^P_2(\mathrm{SAT}))$ so that B is in $\Sigma^P_2(S \oplus \mathrm{SAT})$. Hence, $\Sigma^P_3(S) \subseteq \Sigma^P_2(S \oplus \mathrm{SAT})$ so $S \in \mathrm{EL}_3$. □

Recall that a set A has polynomial-size circuits if and only if there is a sparse set S such that $A \in \mathrm{P}(S)$. The proof of Theorem 4.4 can be trivially modified to yield the following fact.

COROLLARY 4.5. *If A has polynomial-size circuits, then A is in EL_3.*

The notions of extended lowness and extended highness appear to be natural extensions of the basic idea of Schöning that was developed in [15]. It is easy to see that for every n, $\mathrm{EL}_n \cap \mathrm{NP} = \mathrm{L}_n$ and $\mathrm{EH}_n \cap \mathrm{NP} = \mathrm{H}_n$.

5. Generalized highness and sparse sets. The property of being generalized high asserts that using such a set as an oracle set is an advantage. If $A \in \mathrm{H}(k, l)$ where $k > l$, then A has the power of at least $k - l$ alternating quantifiers since $\Sigma_k^P \subseteq \Sigma_l^P(A)$. What type of sets have this property? Schöning [15] showed that certain types of complete sets for NP have this property and it is easy to see that similar sets in each of the classes Σ_i^P, $i > 0$, have this property.

It is known [10], [13] that if a sparse set is hard for NP with respect to $\leqq_T^P$ or even weaker reducibilities such as $\leqq_T^{\mathrm{SN}}$, then the polynomial-time hierarchy collapses. Being hard for any of the classes Σ_i^P, $i > 0$, represents a certain "highness." Thus, we consider the situation in which a sparse set is generalized high. We need the following notion.

DEFINITION 5.1. A set A is *self-reducible* if there exists a deterministic polynomial time-bounded oracle machine M such that (i) on input of size n, M queries the oracle only about strings of length at most $n-1$ and (ii) $\mathrm{L}(M, A) = A$.

This definition captures the essential idea of the seemingly more general notions due to Ko [8] and Meyer and Paterson [14]. Notice that there are $\geqq_m^P$-complete sets for NP that are self-reducible, and that at each stage Σ_k^P of the polynomial-time hierarchy, the $\leqq_m^P$-complete set for Σ_k^P described by Wrathall [18] is self-reducible.

Condition (ii) of the definition of self-reducible set may be interpreted as saying that the set is the unique fixed point specified by the oracle machine. We say this formally as follows.

LEMMA 5.2. *Let A be a self-reducible set and let M witness A's self-reducibility. For any set B and any n, if $\mathrm{L}(M, B)_{\leqq} = B_{\leqq n}$, then $A_{\leqq n} = B_{\leqq n}$.*

Now we have our results.

LEMMA 5.3. *Let A be a self-reducible set. Suppose that for some $k \geqq 0$ and some sparse set S, $A \in \Sigma_k^P(S)$. Then $A \in \mathrm{L}(2, k+2)$, that is, $\Sigma_2^P(A) \subseteq \Sigma_{k+2}^P$.*

Proof. Let M_1 witness A's reducibility so that M_1 is a deterministic polynomial time-bounded oracle machine. Since $A \in \Sigma_k^P(S)$ there are a polynomial p and a deterministic polynomial time-bounded oracle machine M_2 such that for all x, $x \in A$ if and only if $(\exists y_1)_p \cdots (Qy_k)_p \langle x, y_1, \cdots, y_k\rangle \in \mathrm{L}(M_2, S)$. For each finite set T, define the set A_T as follows: $A_T = \{x \mid (\exists y_1)_p \cdots (Q_k y_k)\langle x, y_1, \cdots, y_k\rangle \in \mathrm{L}(M_2, T)\}$. Then for each T, $A_T \in \Sigma_k^P(T) = \Sigma_k^P$.

Let L be an arbitrary set in $\Sigma_2^P(A)$. Then there are a polynomial q and a deterministic polynomial time-bounded machine M_3 such that for all x, $x \in \mathrm{L}$ if and only if $(\exists v)_q(\forall w)_q\langle x, v, w\rangle \in \mathrm{L}(M_3, A)$.

Now consider membership in L. For some polynomial r and all x, $x \in \mathrm{L}$ if and only if there is a set T of size at most $r(|x|)$ such that

(i) for all u, $u \in \mathrm{L}(M_1, A_T)$ if and only if $u \in A_T$, and

(ii) $(\exists v)_q(\forall w)_q\langle x, v, w\rangle \in \mathrm{L}(M_3, A_T)$.

The predicate described by (i) implies that M_3 accepts a suitably small input string relative to A_T if and only if it accepts that input string relative to A; this follows from Lemma 5.2. Now the predicate described by (i) is in Π_{k+1}^P and the predicate described by (ii) is in Σ_{k+2}^P since $A_T \in \Sigma_k^P$, so that membership in L is described by a predicate of the form $\exists(\Pi_{k+1}^P \wedge \Sigma_{k+2}^P)$. Thus, L is in Σ_{k+2}^P.

Since L was chosen arbitrarily in $\Sigma_2^P(A)$, this yields $\Sigma_2^P(A) \subseteq \Sigma_{k+2}^P$ as desired. □

THEOREM 5.4. *If there exists a sparse set that is generalized high, then the polynomial-time hierarchy collapses. That is, if there is a sparse set S such that for some k, l with $k > l$, $S \in H(k, l)$, then $\Sigma_{k+2}^P = \Pi_{k+2}^P = \mathrm{PH}$.*

Proof. Suppose that $\Sigma_k^P \subseteq \Sigma_l^P(S)$ where S is sparse and $k > l$. Let A be a set that is both self-reducible and also $\leqq_m^P$-complete for Σ_k^P. Since A is self-reducible and $A \in \Sigma_k^P \subseteq \Sigma_l^P(S)$, we have $\Sigma_2^P(A) \subseteq \Sigma_{l+2}^P$ by Lemma 5.3. Since A is $\leqq_m^P$-complete for Σ_k^P, $\Sigma_2^P(A) \subseteq \Sigma_{l+2}^P$ implies $\Sigma_{k+2}^P \subseteq \Sigma_{l+2}^P$. Since $k+2 > l+2$, $\Pi_{l+2}^P \subseteq \Sigma_{k+2}^P$ so that $\Sigma_{k+2}^P \subseteq \Sigma_{l+2}^P$ implies $\Pi_{l+2}^P \subseteq \Sigma_{l+2}^P$. Hence, $\mathrm{PH} = \Sigma_{l+2}^P = \Pi_{l+2}^P$. □

THEOREM 5.5. *If there exists a sparse set S such that S is extended high, then the polynomial-time hierarchy collapses.*

Proof. Let S be sparse. If S is in EH_n, then $\Sigma_n^P(S \oplus \mathrm{SAT}) \subseteq \Sigma_n(S)$. Let A be any set that is $\leqq_p^m$-complete for Σ_{n+1}^P and is self-reducible. Thus, $\Sigma_2^P(A) = \Sigma_{n+3}^P$. Now $A \in \Sigma_{n+1}^P \subseteq \Sigma_n^P(S \oplus \mathrm{SAT}) \subseteq \Sigma_n^P(S)$ so by Lemma 5.3, $\Sigma_2^P(A) \subseteq \Sigma_{n+2}^P$. Hence, $\Sigma_{n+3}^P \subseteq \Sigma_{n+2}^P$ so $\Sigma_{n+3}^P = \Sigma_{n+2}^P = \mathrm{PH}$. □

Clearly, the notion of "sparse set" can be replaced by the notion of "set with polynomial-size circuit" in both Theorem 5.4 and Theorem 5.5 and the conclusions will not change.

Theorem 5.4 generalizes a number of results that assert conditions that force the polynomial-time hierarchy to collapse. Some of these conditions are as follows:

(a) there is a sparse set S such that $\mathrm{SAT} \leqq_m^P S$[13];
(b) there is a sparse or co-sparse set S that is $\leqq_T^P$-complete for NP [10], [13];
(c) NP has polynomial-size circuits [7];
(d) $\mathrm{NP} \subseteq \mathrm{APT}$ [14];
(e) $\mathrm{NP} = \mathrm{R}$ [1];
(f) there are a sparse set S and an integer k such that $\mathrm{PH} \subseteq \Sigma_k^P(S)$;
(g) NP is p-selective [8];
(h) $\mathrm{NP} \subseteq \mathrm{BPP}$ [1].

Theorem 5.4 generalizes results of Karp and Lipton [7] and of Yap [19]. The interested reader might compare the proofs of Lemma 5.3 and Theorem 5.4 with those in [7] and [19].

6. Main result. In Theorem 5.4 we showed that the existence of a sparse set that is generalized high causes the polynomial-time hierarchy to collapse. Clearly, if the polynomial-time hierarchy does collapse, then every sparse set in the polynomial-time hierarchy is "n-high" for sufficiently large n and so is generalized high. Thus, we have the first of our two main theorems.

THEOREM 6.1. *Either every sparse set in* PH *is generalized high or no sparse set in* PH *is generalized high.*

In Theorem 5.5 we showed that the existence of a sparse set that is extended high causes the polynomial-time hierarchy to collapse. But it is conceivable that one sparse set S_1 might be extended high while another sparse set S_2 is not extended high. However, this cannot be the case.

PROPOSITION 6.2 [12]. *If the polynomial-time hierarchy collapses, then for every sparse set S, the polynomial-time hierarchy relative to S collapses. That is, if for some $k \geqq 2$, $\Sigma_k^P = \Pi_k^P$, then for every sparse set S, $\Sigma_k^P(S) = \Pi_k^P(S)$.*

Proof. Let S be sparse. Suppose that for some $k \geqq 2$, $\Sigma_k^P = \Pi_k^P$. It suffices to show that if $A \in \Pi_k^P(S)$, then $A \in \Sigma_k^P(S)$. Since $A \in \Pi_k^P(S)$, there is a polynomial p and a deterministic polynomial time-bounded oracle machine M such that for every x, $x \in A$ if and only if $(\forall y_1)_p(\exists y_2)_p \cdots (Q_k y_k)_p \langle x, y_1, \cdots, y_k \rangle \in \mathrm{L}(M, S)$. Hence, $x \in A$ if and only if $(\forall y_1)_p(\exists y_2)_p \cdots (Q_k y_k)_p \langle x, y_1, \cdots, y_k \rangle \in \mathrm{L}(M, S_{\leqq q(|x|)})$ for some appropriate

polynomial q. Let

$$B = \{\langle x, c(T)\rangle \mid T \text{ is a finite set and}$$

$$(\forall y_1)_p(\exists y_2)_p \cdots (Q_k y_k)_p \langle x, y_1, y_2, \cdots, y_k\rangle \in \mathrm{L}(M, T)\}.$$

From the quantifier characterization of PH, it is clear that $B \in \Pi_k^P = \Sigma_k^P = \Sigma_k^P$. Using B, membership in A can be expressed as follows: $x \in A$ if and only if there exists a finite set T such that (i) $(\forall u)_q[u \in T$ if and only if $u \in S]$ and (ii) $\langle x, c(T)\rangle \in B$. This predicate has the form $\exists(C \wedge B)$ where $C \in \Pi_1^P(S)$ and $B \in \Pi_k^P = \Sigma_k^P$. Since $k \geqq 2$, this yields $A \in \Sigma_k^P(S)$. □

Thus, if the polynomial-time hierarchy collapses, then for every sparse set S there is an integer $k > 0$ such that $\mathrm{PH}(S) = \Sigma_k^P(S)$. Since $\mathrm{NP} \subseteq \mathrm{PH} \subseteq \mathrm{PH}(S)$, this means that $\Sigma_k^P(S \oplus \mathrm{SAT}) = \Sigma_k^P(S)$ so that S is extended high. Thus, we have our other result.

THEOREM 6.3. *Either every sparse set is extended high or no sparse set is extended high.*

REFERENCES

[1] L. ADLEMAN, *Two theorems on random polynomial time*, Proc. 19th IEEE Symposium on Foundations of Computer Science, 1978, pp. 75-83.

[2] T. BAKER, J. GILL AND R. SOLOVAY, *Relativizations of the P = ? NP question*, this Journal, 4 (1975), pp. 161-173.

[3] T. BAKER AND A. SELMAN, *A second step towards the polynomial-time hierarchy*, Theoret. Comput. Sci., 8 (1979), pp. 177-187.

[4] P. BERMAN, *Relationships between density and deterministic complexity of* NP*-complete languages*, Proc. 5th ICALP, Lecture Notes in Computer Science 67, Springer-Verlag, Berlin, pp. 63-71.

[5] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of* NP *and other complete sets*, this Journal, 6 (1977), pp. 305-322.

[6] S. FORTUNE, *A note on sparse complete sets*, this Journal, 8 (1979), pp. 431-433.

[7] R. KARP AND R. LIPTON, *Some connections between nonuniform and uniform complexity classes*, Proc. 12th ACM Symposium on Theory of Computing, 1980, pp. 302-309.

[8] K. KO, *On Self-reducibility and weak* P*-selectivity*, J. Comput. Syst. Sci., 26 (1982), pp. 209-221.

[9] K. KO AND U. SCHÖNING, *On circuit-size complexity and the low hierarchy in* NP, this Journal, 13 (1984), pp. 41-51.

[10] T. LONG, *A note on sparse oracles for* NP, J. Comput. Syst. Sci., 24 (1982), pp. 224-232.

[11] ———, *Strong nondeterministic polynomial-time reducibilities*, Theoret. Comput. Sci., 21 (1982), pp. 1-25.

[12] T. LONG AND A. SELMAN, *Relativizing complexity classes with sparse oracles*, J. Assoc. Comput. Mach., to appear.

[13] S. MAHANEY, *Sparse complete sets for* NP*: solution to a conjecture of Berman and Hartmanis*, J. Comput. Syst. Sci., 25 (1982), pp. 130-143.

[14] A. MEYER AND M. PATERSON, *With what frequency are apparently intractable problems difficult?*, MIT Technical Report, Massachusetts Inst. of Technology, Cambridge, MA, February, 1979.

[15] U. SCHÖNING, *A low- and a high-hierarchy in* NP, J. Comput. Syst. Sci., 27 (1983), pp. 14-28.

[16] ———, *A note on small generators*, Theoret. Comput. Sci., 34 (1984), pp. 337-342.

[17] L. STOCKMEYER, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 1-22.

[18] C. WRATHALL, *Complete sets and the polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1976), pp. 23-33.

[19] C. YAP, *Some consequences of nonuniform conditions on uniform classes*, Theoret. Comput. Sci., 26 (1983), pp. 287-300.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
009

DIGITAL SEARCH TREES REVISITED*

PHILIPPE FLAJOLET† AND ROBERT SEDGEWICK‡

Abstract. Several algorithms have been proposed which build search trees using digital properties of the search keys. A general approach to the study of the average case performance of such algorithms is discussed, with particular attention to the analysis of the *digital search tree* structures of Coffman and Eve. Specifically, the method leads to the solution of a problem left open by Knuth, finding the average number of nodes in digital search trees with both sons null.

The paper may be of interest as a survey and tutorial treatment of the analysis of the three primary digital tree search methods: digital search trees, radix search tries, and Patricia tries.

Key words. analysis of algorithms, search trees, path length, asymptotic analysis, partitions

1. Introduction. A fundamental problem in computer science is the so-called *dictionary problem*, where various operations, chiefly *search* and *insert*, are to be performed on a set of records possessing key values. To *insert* a record is to store it away for later retrieval; to *search* is to find a previously stored record with a given key value. The *binary search tree* is an elementary data structure for solving this problem: records are stored in nodes which contain two distinguished values (*left* and *right*) which point to other nodes or could be null. One node, called the *root*, is pointed to by no other nodes, otherwise each node is referenced by exactly one other node. To search for a record with value v, we set x to point to the root and perform the following operations until termination:

- If x is null then terminate (v not found).
- If $key(x) = v$ then terminate (v found).
- Otherwise, if $v < key(x)$ then set x to $left(x)$;
  if $v > key(x)$ then set x to $right(x)$.

To insert a new record with a new value v, we search, then replace the null pointer that caused termination with a pointer to the new record. The analysis of the performance of this method is well-known: if records with keys from a random permutation of N elements are successively inserted into an initially empty tree, then the expected number of nodes examined in a successful search in the resulting tree is

$$2\left(1+\frac{1}{N}\right)H_N - 3 = (2\ln 2)\lg N + 2\gamma - 3 + O\left(\frac{\log N}{N}\right).$$

See [9] for details. Throughout this paper we use the notations $H_N = \sum_{1\le k\le N} 1/k = \ln N + \gamma + 1/2N + O(1/N^2)$, where $\gamma = .57721\cdots$ is Euler's constant; $\lg N \equiv \log_2 N$; and $\ln N \equiv \log_e N$. The approximate value of the coefficient of $\lg N$ in the leading term is $1.38630\cdots$. For a perfectly balanced tree, the coefficient would be 1, but an $O(N)$ worst case is possible (for example if the keys are inserted in ascending order). Several methods are available to make the worst case search time close to $\lg N$. One technique is to periodically perform structural modifications on the trees to keep them

* Received by the editors April 3, 1984, and in revised form April 24, 1985.

† INRIA, Rocquencourt, France.

‡ Department of Computer Science, Princeton University, Princeton, New Jersey 08544. The research of this author was done primarily while visiting at INRIA, and was also supported in part by the National Science Foundation under grant MCS-83-08806 and by DARPA under contract N00014-83-K-0146 while the author was at Brown University, Providence, Rhode Island.

"well-balanced" (for example, see [16]). In this paper we consider in detail an alternative class of methods.

The *digital search tree* [1] is a data structure which leads to much improved worst case performance (and asymptotically optimal average case performance as well) by making use of the digital properties of the keys, if that is appropriate. We simply assume that the keys can be represented as binary numbers so that it makes sense to refer to the bth bit of a key, where the bits are numbered, say, from left to right. Then, to search for a record with value v, we set x to point to the root and b to 1, then perform the following operations until termination:

- If x is null then terminate (v not found).
- If $key(x) = v$ then terminate (v found).
- Otherwise, if the bth bit of v is 0 then set x to $left(x)$;
  if the bth bit of v is 1 then set x to $right(x)$
- Set b to $b+1$.

Insertion is done exactly as with binary search trees (i.e., the null pointer which caused termination is replaced by a pointer to the new record).

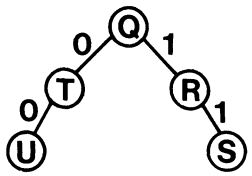


FIG. 1

Figure 1 shows a digital search tree built by inserting the keys 010 (Q), 110 (R), 111 (S), 001 (T), and 000 (U) in that order. Note that the order in which the keys are inserted is relevant. For example, the tree shown in Fig. 2 results from inserting the same keys in reverse order: 000 (U), 001 (T), 111 (S), 110 (R), and 010 (Q).

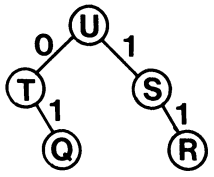


FIG. 2

In some implementations, it may be convenient to assume that the keys are all of the same length, as in the examples above. The method also is appropriate for varying length keys, provided that no key is a prefix of another. The number of nodes examined in a digital search tree of N keys is limited by the number of bits in the keys, which is larger than lg N but is likely to be within a constant factor for many natural situations. The average case performance of this method is also known (in the analysis we assume that the keys are infinitely long): if N records with keys composed of random bit streams are inserted into an initially empty tree, then the average number of nodes examined during a successful search in the resulting digital search tree is

$$\lg N + \frac{\gamma - 1}{\ln 2} + \frac{3}{2} - \alpha + \delta(N) + O\left(\frac{\log N}{N}\right)$$

where α is a constant between 1 and 2, and $\delta(N)$ is a small ($|\delta(N)| < 10^{-6}$) oscillatory term ($\delta(2N) = \delta(N)$) which are defined in detail in the next section. This is about 38% less than for binary tree search, but note that the results are not necessarily directly comparable because the input models differ. The above result is due to Konheim and Newman [11]; the refinement including the periodic term is due to Knuth [9, Ex. 6.3-27]. In this paper we give an alternate derivation that generalizes to yield other statistics about the trees, in particular solving a problem left open by Knuth [9, Ex. 6.3-29].

Digital search trees are easily confused with *radix search tries*, a different application of essentially the same idea. A *binary trie* has two types of nodes: *internal nodes*, which consist of left and right pointers only; and *external nodes*, which consist of keys only. To search for a record with value v in a binary trie, we set x to the root and b to 1, then perform the following operations until termination:

- If x is external, then terminate
  (if $key(x) = v$ then v found, otherwise v not found).
- Otherwise, if the bth bit of v is 0 then set x to $left(x)$;
  if the bth bit of v is 1 then set x to $right(x)$.
- Set b to $b+1$.

Insertion is more complicated in tries than in binary or digital search trees. On termination of an unsuccessful search for a key v to be inserted, we have two keys which belong in the same external node. If the bth bits of those keys differ, we replace that external node by an internal node which points to two external nodes containing the keys; otherwise we have to also include an internal node corresponding to each bit beyond the bth for which the two keys match. Figure 3 shows the trie for our example set of keys {010 (Q), 110 (R), 111 (S), 001 (T), and 000 (U)}.

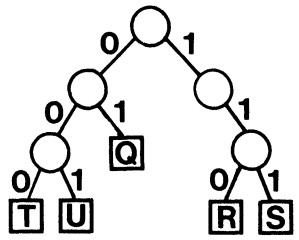


FIG. 3

In contrast to digital search trees, the same trie is constructed no matter in what order the keys are inserted. Tries can have more than N internal nodes to store N keys; also handling multiple node types is inconvenient in many programming environments. It is possible to eliminate both of these problems (see below). Most interesting statistics for tries have been fully analyzed; for example, if N records with keys from random bit streams are inserted into an initially empty trie, then the average number of nodes examined during a successful search in the resulting trie is

$$\lg N + \frac{\gamma}{\ln 2} + \frac{1}{2} + \delta(N) + O\left(\frac{1}{N}\right)$$

even though the average number of internal nodes in the trie is about $N/\ln 2 \approx 1.44269 \cdots N$.

It is possible to ensure that a trie constructed with N keys has just $(N-1)$ internal nodes by collapsing one-way branches on internal nodes. Figure 4 shows the result of this on our example.

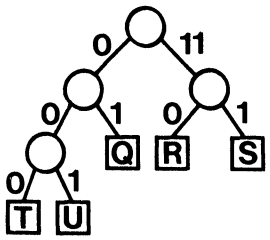


FIG. 4

Equal bits in keys do not affect the structure of such tries. The programming details of how to accomplish this are not relevant to this paper. We refer to these structures as *Patricia tries* [9] because they are the basis of an alogrithm called Patricia which also manages to store the keys in internal nodes and thus avoid the multiple-node-type problem referred to above. Patricia is somewhat more complicated than digital tree searching, but it has applications beyond searching which make it of independent interest. From an analytic standpoint, direct comparisons between Patricia and digital searching are suggested because they both build search keys into (the same class of) binary tree structures, using digital properties of the keys. Knuth has probed many aspects of Patricia in depth: for example, the number of nodes examined in an average successful search is one less than for standard tries.

Many more details on the use and application of these methods may be found in [9] and [16]. In this paper we present new results on the analysis of digital search trees. The above introductory description is intended to motivate this analysis and to provide a context within which we can discuss the relationship of the methods we use to previous analyses of the various algorithms. In the next section, we give an analysis of the average internal path length of a digital search tree which illustrates our basic method and provides an alternate derivation to the one provided by Knuth for this problem. Following that, we use the same general method to find the average number of nodes in a digital search tree with both links null, a somewhat more complicated problem left open by Knuth. In § 4 we consider M-way branching. Section 5 is a discussion of various generalizations.

2. Path length. The *internal path length* of a tree is the sum of the number of nodes on the path from the root to each node in the tree. The average number of nodes examined during a successful search in a search tree with N nodes is one plus the internal path length divided by N.

Let A_N be the average internal path length of a digital search tree built from N (sufficiently long) keys comprised of random bits. Then we have the fundamental recurrence relation

$$A_N = N-1+\sum_k \frac{1}{2^{N-1}}\binom{N-1}{k}(A_k+A_{N-1-k}), \qquad N \geqq 1 \tag{1}$$

with A_0 defined to be 0. This follows from three easily established facts. First, the internal path length of any tree of N nodes is $N-1$ plus the internal path length of the two subtrees of the root. Second, the probability that the left subtree of the root has k nodes (and the right subtree has $N-1-k$ nodes) is $\binom{N-1}{k}/2^{N-1}$, the probability

that exactly k of the $N-1$ nodes that are inserted after the first node start with a 0 bit. Third, the subtrees themselves are randomly built according to the same model. Recurrence relations of this type are used to describe the performance of many tree-searching methods. As discussed further below, slight differences in the equations can make the analysis somewhat more intricate.

By symmetry (change k to $N-1-k$ in the second part of the sum), the recurrence (1) is equivalent to

$$A_N = N-1+2\sum_k \frac{1}{2^{N-1}}\binom{N-1}{k}A_k, \qquad N\geq 1$$

with A_0 defined to be 0. This recurrence is transformed into a functional equation on the exponential generating function $A(z)=\sum_{N>0} A_N z^N/N!$ by multiplying both sides by $z^{N-1}/(N-1)!$ and summing for $N\geq 1$:

$$\begin{aligned}\sum_{N\geq 1}\frac{A_N z^{N-1}}{(N-1)!} &= \sum_{N\geq 2}\frac{z^{N-1}}{(n-2)!}+2\sum_{N\geq 1}\sum_{0\leq k\leq N-1}\frac{1}{2^{N-1}}\binom{N-1}{k}A_k\frac{z^{N-1}}{(N-1)!}\\ &= z\,e^z+2\sum_{k\geq 0}\frac{A_k}{k!}\sum_{N\geq k+1}\left(\frac{z}{2}\right)^{N-1}\frac{1}{(N-k-1)!}\\ &= z\,e^z+2\sum_{k\geq 0}\frac{A_k(z/2)^k}{k!}\sum_{N\geq 0}\frac{(z/2)^N}{N!},\end{aligned}$$

$$A'(z)=z\,e^z+2A(z/2)\,e^{z/2}.$$

This difference-differential equation can be transformed into a somewhat more manageable form by introducing the generating function $B(z)=\sum_{N\geq 0} B_N z^N/N!$ with

$$B(z)\equiv e^{-z}A(z).$$

That is, $A(z)=e^z B(z)$ and $A'(z)=e^z B'(z)+e^z B(z)$. In terms of $B(z)$, the above difference-differential equation becomes

$$B'(z)+B(z)=z+2B(z/2).$$

This corresponds to a simple recurrence on the coefficients

$$B_N+B_{N-1}=\frac{1}{2^{N-2}}B_{N-1},$$

or

$$B_N=-\left(1-\frac{1}{2^{N-2}}\right)B_{N-1}, \qquad N\geq 3,$$

with $B_2=1$, which telescopes to give an explicit formula for B_N:

$$B_N=(-1)^N\prod_{1\leq j\leq N-2}\left(1-\frac{1}{2^j}\right).$$

Similar quantities arise in the theory of partitions, and we shall have occasion later to use classical identities from that theory. We have $B_N=(-1)^N Q_{N-2}$, where

$$Q_N\equiv\prod_{1\leq j\leq N}\left(1-\frac{1}{2^j}\right).$$

As $N\to\infty$, this approaches the limit $Q_\infty=.288788\cdots$. Now, expanding the formula

$A(z)=e^z B(z)$ shows that $A_N=\sum_k \binom{N}{k} B_k$, so (after handling initial conditions) we obtain an explicit formula for A_N:

$$A_N=\sum_{k\geq 2}\binom{N}{k}(-1)^k Q_{k-2}. \tag{2}$$

It remains to evaluate this sum.

At this point, it might be worth noting the relationship between this derivative and the corresponding derivation for binary tries. The fundamental recurrence for tries is

$$A_N^{[T]}=N+\sum_k \frac{1}{2^N}\binom{N}{k}(A_k^{[T]}+A_{N-k}^{[T]}),\qquad N\geq 2, \tag{3}$$

with $A_0^{[T]}$ and $A_1^{[T]}$ both defined to be 0. This is the number of nodes examined during all successful searches, but it is the average *external* path length of the trie. Note that since no key is stored at the root, the subtrees have a total of N keys. The resulting functional equation on the exponential generating function is not a difference-differential equation but simply a difference equation:

$$A^{[T]}(z)=z(e^z-1)+2A^{[T]}(z/2)\,e^{z-2}.$$

It is still convenient to transform the equation with $A(z)=e^z B(z)$ to get the equation

$$B^{[T]}(z)=z(1-e^{-z})+2B^{[T]}(z/2).$$

This yields directly

$$B_N^{[T]}=\frac{N(-1)^N}{1-1/2^{N-1}}$$

and

$$A_N^{[T]}=\sum_{k\geq 2}\binom{N}{k}\frac{k(-1)^k}{1-1/2^{k-1}}$$

which is somewhat simpler than (2) and can be handled directly the Mellin transform techniques [2], as described in full detail in [6] and [9, § 5.2.2].

The fundamental recurrence for the average external path length of Patricia tries is trickier:

$$A_N^{[P]}=N\left(1-\frac{1}{2^{N-1}}\right)+\sum_k \frac{1}{2^N}\binom{N}{k}(A_k^{[P]}+A_{N-k}^{[P]}),\qquad N\geq 1, \tag{4}$$

with $A_0^{[P]}$ defined to be 0. The external path length of a Patricia trie is the sum of the external path lengths of the subtries of the root, plus the number of nodes in the subtries (N) *unless* one of the subtries is empty (probability $1/2^{N-1}$). The functional equation on the exponential generating function is similar to that for tries:

$$A^{[P]}(z)=z(e^z-e^{z/2})+2A^{[P]}(z/2)\,e^{z/2}$$

with transformed version

$$B^{[P]}(z)=z(1-e^{-z/2})+2B^{[P]}(z/2)$$

which yields directly

$$B_N^{[P]}=\frac{N(-1)^N}{2^{N-1}-1}$$

so that

$$A_N^{[P]} = \sum_{k\geq 2} \binom{N}{k} \frac{k(-1)^k}{2^{k-1}-1} = A_N^{[T]} - N$$

as mentioned above.

The method used above is equivalent to the "binomial transform" method described by Knuth, but it is perhaps more transparent.

For the average internal path length of digital search trees, Knuth uses an approach suggested by Konheim and Newman [11] to transform (2) into a form which has essentially the same asymptotics as the above trie sum. This derivation is somewhat indirect, and does not provide a way to analyze other properties of digital search trees. But Knuth gives an alternate method for evaluating the trie sum, which he attributes to S. O. Rice [9, Ex. 5.2.2-53]. We next show how this method applies directly to (2).

Rice's method is based on a classical formula from the calculus of finite differences [12]:

LEMMA 1. *Let C be a closed curve encircling the points $0, 1, \cdots, N$, and let $f(z)$ be a function which is analytic within C. Then*

$$\sum_k \binom{N}{k} (-1)^k f(k) = -\frac{1}{2\pi i} \int_C B(N+1, -z) f(z)\, dz$$

where $B(x, y)$ is the classical Beta function defined by $B(x, y) = \Gamma(x)\Gamma(y)/\Gamma(x+y)$.

Proof. Noting that

$$-B(N-1, -z) = (-1)^N \frac{z(z-1)\cdots(z-n)}{N!} = z \prod_{1\leq k\leq N} \left(\frac{z}{k} - 1\right),$$

we can write an equivalent version (which has amusing similarities between the left- and right-hand sides) of the equation in the statement of the lemma:

$$\sum_k \frac{N(N-1)\cdots(N-k+1)}{k!} (-1)^k f(k) = \frac{1}{2\pi i} \int_C \frac{N!}{z(z-1)\cdots(z-N)} (-1)^N f(z)\, dz.$$

To verify this is a straightforward application of Cauchy's theorem: the integral is the sum of the residues inside C, and the residue at $z = k$ is $\binom{N}{k}(-1)^k f(k)$ for each k in the range $0 \leq k \leq N$. □

This general identity arises in the study of finite differences, since the sum $\sum_k \binom{N}{k}(-1)^k f(k)$ is precisely $\nabla^N f(0)$, were $\nabla f(k) = f(k-1) - f(k)$ (see, for example, [12]).

To use Lemma 1 for asymptotic analysis, we change C to a large curve around which the integral is small, and take into account residues at poles in the larger enclosed area. This method actually plays a rather fundamental role in the analysis of the class of problems considered here.

Note that the function $B(N+1, -z)$ has poles at the integers $0, 1, \cdots, N$. Thus, using Rice's method with Lemma 1 as stated would involve examining only the singularities of $f(z)$. However, the lemma also clearly holds if the sum is taken over any subset of the integers $0, 1, \cdots, N$ (and C is taken to enclose just those points): then application of Rice's method might have to take into account the singularities of $B(N+1, -z)$ outside C. In particular, in the cases of interest in this paper, the points 0 and 1 are not included in C. In fact, we have to cope with double poles at these points (as well as many singularities for the function $f(z)$).

To apply Rice's method to the evaluation of (2), we need to define an appropriate meromorphic function to extend Q_k, which is defined only on the integers. To this end, we introduce the function

$$Q(x)=\left(1-\frac{x}{2}\right)\left(1-\frac{x}{4}\right)\left(1-\frac{x}{8}\right)\cdots.$$

Note that $Q(1)=Q_\infty$ and that $Q_N=Q(1)/Q(2^{-N})$, so that the analytic expression $f(z)=Q(1)/Q(2^{-z})$, which is defined when z is a positive integer, gives the appropriate extension. Actually, this extension can be derived in a rather mechanical fashion, because Q_N is defined by the recurrence relationship

$$Q_{N+1}=\left(1-\frac{1}{2^{N+1}}\right)Q_N, \qquad N\geqq 1,$$

with $Q_0=1$, which simplifies to the expression

$$Q_N=\frac{1}{1-(1/2)^{N+1}}Q_{N+1}.$$

We simply extend this recurrence relation and telescope it:

$$\begin{aligned} f(z)&=\frac{1}{1-(1/2)^{z+1}}f(z+1)\\ &=\frac{1}{1-(1/2)^{z+1}}\,\frac{1}{1-(1/2)^{z+2}}f(z+2)\\ &\vdots\\ &=\frac{1}{Q(2^{-z})}\lim_{z\to\infty}f(z) \end{aligned}$$

provided the limit exists. But $f(0)=Q_0=1$ implies that $\lim_{z\to\infty}f(z)=Q(1)$, so $f(z)=Q(1)/Q(2^{-z})$ is a proper analytic extension. We point out this "mechanical" derivation because we use essentially the same method for a more difficult problem in the next section.

Thus, by Lemma 1 (see also the comments following the lemma), we know that

$$A_N=-\frac{1}{2\pi i}\int_C B(N+1,-z)\frac{Q(1)}{Q(2^{-z+2})}\,dz \tag{5}$$

where C encloses the points $2,3,\cdots,N$. To complete the analysis, we expand C to a larger curve and study the behavior of the integrand at newly enclosed singularities. These residue computations are simplified somewhat because the functions involved have a simple product form; the following lemma, which is elementary, will prove useful for most of the series expansions done in this paper.

LEMMA 2. *If $F(z)=\prod_{j\in R}(1-f_j(z))^{-1}$ for some index set R, then the Taylor series expansion of F at a, if it exists, is given by*

$$F(z)=F(a)\left(1+\sum_{j\in R}\frac{f_j'(a)}{1-f_j(a)}(z-a)+O(z-a)^2\right).$$

Proof. Elementary from the use of the "logarithmic derivative": if $G(z)=\prod_{k\in R}g_k(z)$ then $G'(z)/G(z)=\sum_{k\in R}g_k'(z)/g_k(z)$. □

For example, at $a=1$, we have the following expansion for the Beta function

$$-B(N-1,-z)=\frac{1}{1-z}z^{-1}\prod_{2\leq j\leq N}\left(1-\frac{z}{j}\right)^{-1}$$

$$=\frac{1}{1-z}N(1+(H_{N-1}-1)(z-1)+O((z-1)^2)$$

$$=-\frac{N}{z-1}-N(H_{N-1}-1)+O(z-1).$$

Similarly, for the Q function, we have

$$Q(1)/Q(2^{-z+1})=Q(1)\prod_{j<1}(1-2^{-z+j})^{-1}$$

$$=1-\ln 2\sum_{j<1}\frac{2^{j-1}}{1-2^{j-1}}(z-1)+O(z-1)^2$$

$$=1-\alpha\ln 2(z-1)+O(z-1)^2$$

where $\alpha=1+\frac{1}{3}+\frac{1}{7}+\frac{1}{15}+\cdots$. This is a fundamental constant which arises in the analysis of several algorithms, for example Heapsort [9, p. 156] and approximate counting [3].

We are now ready to complete the analysis of the average intenal path length of digital search trees using Rice's method for the asymptotic analysis:

THEOREM 1. (Konheim–Newman, Knuth). *The average internal path length of a digital search tree built from N records with keys from random bit streams is*

$$(N+1)\lg N+\left(\frac{\gamma-1}{\ln 2}+\frac{1}{2}-\alpha+\delta(N)\right)N+O(N^{1/2})$$

where $\gamma=.577216\cdots$ *is Euler's constant,* $\alpha=1+\frac{1}{3}+\frac{1}{7}+\frac{1}{15}+\cdots=1.606695\cdots$, *and* $\delta(N)$ *is a periodic function in* $\lg N$, *with* $|\delta(N)|<10^{-6}$. *The approximate value of the coefficient of the linear term is* $-1.7155\cdots$.

Proof. Following the discussion above, the value sought is given by the integral (5). If we change C to a large rectangle R_{XY} with corners at the four points $(\frac{1}{2}\pm iY, X\pm iY)$, then we know by Cauchy's theorem that the integral around R_{XY} (which we shall show to be small) is equal to A_N minus the sum of the residues of

$$B(N+1,-z)\frac{Q(1)}{Q(2^{-z+2})}$$

at poles within R_{XY} but not within C.

Rewriting $B(N+1,-z)$ as $\Gamma(N+1)\Gamma(-z)/\Gamma(N+1-z)$, we can make use of standard asymptotic expansions of the Γ function to bound the value of the integral around R_{XY}. We have the approximations

$$\frac{\Gamma(N)}{\Gamma(N+a)}=N^{-a}+O(N^{-a-1})$$

and

$$\Gamma(x+iY)=O(|Y|^{x-1/2}e^{-\pi|Y|/2})$$

(see, for example, [17, Chap. XII]). Thus, a bound for our integral along the top and bottom lines of R_{XY} is given by

$$O\left(\int(N+1)^{x+iY}|Y|^{x-1/2}e^{-\pi Y/2}\,dx\right).$$

This bound is valid only if $Q(2^{-z+2})$ does not get too close to zero; we can insure this by taking Y to be of the form $(\pi/\ln 2)(2Y'+1)$, with Y' an integer. Thus, the integral is exponentially small in Y and vanishes on the top and bottom of R_{XY} as $Y\to\infty$. A similar argument shows that the integral vanishes on the right of R_{XY} as $X\to\infty$. On the left, we have the bound

$$O\left(\int_{-Y}^{Y}\frac{\Gamma(N+1)}{\Gamma(N+1/2-iy)}\,dy\right)=O(N^{1/2}).$$

This proves that A_N plus the sum of the residues in the halfplane to the right of the line $x=\frac{1}{2}$ (but not within C) is $O(N^{1/2})$. We now proceed to calculate these residues.

The integrand has poles at $z=j\pm(2\pi ik)/\ln 2$ for $j=1, 0, -1, -2, \ldots$, and all $k\geqq 0$, since at these points $2^{-z+j}=1$ which causes one of the factors of $Q(2^{-z+2})$ to vanish. The poles at 0 and 1 are double poles because $B(N+1,-z)$ is also singular at 0 and 1. Of these, only the poles for $j=1$ are within the region of interest; thus we have to compute residues at the double pole 1 and at the points $1\pm(2\pi ik)/\ln 2$ for $k\neq 0$.

At $z=1$, we use the series expansions derived from Lemma 2 above:

$$\begin{aligned}-B(N+1,-z)\frac{Q(1)}{Q(2^{-z+2})}&=-B(N+1,-z)\frac{1}{1-2^{-z+1}}\frac{Q(1)}{Q(2^{-z+1})}\\&=\left(-\frac{N}{z-1}-N(H_{N-1}-1)+O(z-1)\right)\\&\quad\times\left(\frac{1}{(z-1)\ln 2}+\frac{1}{2}+O(z-1)\right)\\&\quad\times(1-\alpha\ln 2(z-1)+O(z-1)^2).\end{aligned}$$

The residue at $z=1$ (the coefficient of $1/(z-1)$ in this product) is

$$-\frac{N}{\ln 2}(H_{N-1}-1)+N\left(\alpha-\frac{1}{2}\right)=-N\lg N-N\left(\frac{\gamma-1}{\ln 2}-\alpha+\frac{1}{2}\right)+O(1).$$

The poles at $1\pm 2\pi ik/\ln 2$ add a small contribution to the linear term: the total residue at these points is

$$-\frac{1}{\ln 2}\sum_{k\neq 0}B\left(N+1,-1-\frac{2\pi ik}{\ln 2}\right).$$

As above, we can write $B(N+1,-z)$ as $\Gamma(N+1)\Gamma(-z)/\Gamma(N+1-z)$ and use Lemma 2 to develop an asymptotic expansion. We have

$$\begin{aligned}B\left(N+1,-1-\frac{2\pi ik}{\ln 2}\right)&=\frac{\Gamma(N+1)\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)}{\Gamma\left(N-\frac{2\pi ik}{\ln 2}\right)}\\&=N\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)\frac{\Gamma(N)}{\Gamma\left(N-\frac{2\pi ik}{\ln 2}\right)}\\&=N\Gamma\left(-1-\frac{2\pi ik}{\ln 2}\right)N^{2\pi ik/\ln 2}\left(1+O\left(\frac{1}{N}\right)\right).\end{aligned}$$

This estimate is valid just for fixed k as N grows: the calculation of the uniform bound necessary for the calculation of the error term can be derived from more detailed asymptotics on the Beta function (see [17, pp. 277-8]). The sum of the residues at the points $1 \pm 2\pi ik/\ln 2$ is found to be

$$-N\delta(N) + O(1)$$

where

$$\delta(N) = \frac{1}{\ln 2} \sum_{k \neq 0} \Gamma\left(-1 - \frac{2\pi ik}{\ln 2}\right) e^{2\pi ik \lg N}.$$

This and related functions arise in the study of many algorithms, for example: evaluating arithmetic expressions [5], parallel addition [10], extendible hashing [13], approximate counting [3], and Batcher's merging networks [15]. The properties of $\delta(N)$ cited in Theorem 1 are discussed in Knuth [9, p. 134].

Thus, subtracting the sum of the residues at $z = 1$ and at $z = 1 \pm 2\pi ik/\ln 2$ for $k \neq 0$ from the estimate of the value of the contour integral around R_{XY} in the limit, we have shown that

$$A_N = N \lg N + N\left(\frac{\gamma - 1}{\ln 2} - \alpha + \frac{1}{2} + \delta(N)\right) + O(N^{1/2})$$

as desired. The same method of analysis can be used to expand A_N to any desired asymptotic accuracy, by using a contour R_{XY} which includes more poles. The double pole at $z = 0$ and the poles at $z = \pm 2\pi ik/\ln 2$ for $k \neq 0$ contribute a constant term like the coefficient of the linear term, and the poles at $z = j \pm 2\pi ik/\ln 2$ for $j = -1, -2, \cdots$ contribute more complicated (but very small) oscillatory terms. □

Our interest in this derivation is that it illustrates a general method of evaluating sums of the form $\sum_k \binom{N}{k}(-1)^k f(k)$ even when $f(k)$ is a relatively complicated function. (As mentioned above, the proof of Theorem 1 is quite specific to Q_{k-2}.) Essentially, the asymptotic analysis is reduced to a singularity analysis of a meromorphic function satisfying the same recurrence as $f(k)$. Next, we show how this method applies for a function satisfying an inhomogeneous recurrence. This problem arises naturally in the study of other properties of digital search trees.

3. External internal nodes. A property of trees of some interest is the number of internal nodes which have both links null. An alternate storage representation could be used for such nodes. The question left open by Knuth (see [9, Ex. 6.3-29]) is to determine exactly how much storage can be saved. This is of more practical importance when M-ary trees (not binary) are considered (see the next section). In this paper we are interested in the problem chiefly because it illustrates the power of Rice's method, as contrasted with standard Mellin techniques, which seem difficult to apply directly to this problem. (Another application of Rice's method may be found in [7].)

In a fully balanced binary tree of N nodes the number of nodes with both links null is $\lceil N/2 \rceil$; in a completely unbalanced tree the number is 1. It is a simple exercise to show that the average number of such nodes for a random binary search tree is $(N+1)/3$. We expect digital search trees to be somewhat more balanced than binary search trees; thus the result should be that somewhere between one-third and one-half of the nodes have both links null. It is mildly surpising that the answer is somewhat closer to the former than the latter.

THEOREM 2. *The average number of nodes with both links null in a digital search tree built from N records with keys from random bit streams is*

$$N\left(\beta+1-\frac{1}{Q_\infty}\left(\frac{1}{\ln 2}+\alpha^2-\alpha\right)+\delta^*(N)\right)+O(N^{1/2})$$

where the constants involved have the values

$$\alpha=1+\frac{1}{3}+\frac{1}{7}+\frac{1}{15}+\cdots=1.606695\cdots,$$

$$Q_\infty=\frac{1}{2}\cdot\frac{3}{4}\cdot\frac{7}{8}\cdots=.288788\cdots,$$

and

$$\beta=\frac{1\cdot 2^2}{1}\left[\frac{1}{1}\right]+\frac{2\cdot 2^3}{1\cdot 3}\left[\frac{1}{1}+\frac{1}{3}\right]+\frac{3\cdot 2^4}{1\cdot 3\cdot 7}\left[\frac{1}{1}+\frac{1}{3}+\frac{1}{7}\right]+\cdots=7.74313\cdots.$$

The function $\delta^*(N)$ is a periodic function in lg N, *with* $|\delta^*(N)|<10^{-6}$. *The approximate value of the coefficient of the leading term is* $0.372046812\cdots$.

Proof. As before, we use a simple transform with generating functions to derive an explicit sum, then use Rice's method to evaluate the sum.

If C_N is the quantity sought, then we have the recurrence

$$C_N=\sum_k \frac{1}{2^{N-1}}\binom{N-1}{k}(C_k+C_{N-1-k}),\qquad N\geqq 2, \tag{6}$$

with $C_1=1$ and $C_0=0$. This follows from the same argument as for (1), with the additional observation that the number of nodes with both links null in a tree is exactly the sum of the numbers of such nodes in the two subtrees of the root, unless the tree has just one node.

In terms of the exponential generating function $C(z)=\sum_{N\geqq 0} C_N z^N/N!$, this leads to the equation

$$C'(z)=1+2C(z/2)\,e^{z/2}$$

which becomes, in terms of the transform generating function $D(z)=\sum_{N\geqq 0} D_N z^N/N!$, with $D(z)$ defined to be $e^{-z}C(z)$:

$$D'(z)+D(z)=e^{-z}+2D(z/2).$$

This gives a recurrence on the coefficients as before:

$$D_N+D_{N-1}=(-1)^{N-1}+\frac{1}{2^{N-2}}D_{N-1},$$

$$D_N=(-1)^{N-1}-\left(1-\frac{1}{2^{N-2}}\right)D_{N-1},\qquad N\geqq 2,$$

with $D_0=0$ and $D_1=1$. But this recurrence is inhomogeneous, so the telescoped solution is somewhat more complicated than before:

$$D_N=(-1)^{N-1}\sum_{1\leqq i\leqq N-1}\ \prod_{i\leqq j\leqq N-2}\left(1-\frac{1}{2^j}\right).$$

Rewriting this in terms of

$$R_N \equiv Q_N\left(1+\frac{1}{Q_1}+\cdots+\frac{1}{Q_N}\right)$$

and transforming by $C(z)=e^z D(z)$, we have the following explicit sum for the desired quantity:

$$C_N = N - \sum_{k\geq 2}\binom{N}{k}(-1)^k R_{k-2}. \tag{7}$$

This sum is more difficult to evaluate than (2) because R_N is more complicated than Q_N.

We might begin by mimicking the "mechanical" derivation of $Q(z)$, turning the recurrence defining R_k around to define a meromorphic function satisfying the same recurrence. We have

$$R_{N+1} = 1+\left(1-\frac{1}{2^{N+1}}\right)R_N$$

or, solving for R_N and substituting $q=1/2$:

$$R_N = -\frac{1}{1-q^{N+1}}+\frac{1}{1-q^{N+1}}R_{N+1}. \tag{8}$$

(From this point on, we will use q for $1/2$. Not only is this a notational convenience, but we will see in the next section that this is the only change necessary to solve the same problem for M-ary digital search trees.) Using this recurrence to extent to a function on the complex plane would give

$$\begin{aligned} R(z) &= \frac{1}{1-q^{z+1}}+\frac{1}{1-q^{z+1}}R(z+1) \\ &= \sum_{j\geq 0}\frac{1}{(1-q^{z+1})(1-q^{z+2})\cdots(1-q^{z+1+j})}. \end{aligned}$$

Unfortunately, this sum does not converge when z is a positive integer, so it certainly does not extend R_N. The reason is clear: R_N itself is not bounded as N increases, so extending a recurrence to increasing N is doomed to failure. Fortunately, it is not difficult to avoid this problem: by studying the asymptotic performance of R_N we can find a closely related function which can be extended by the above technique.

This asymptotic development is elementary from Lemma 2 because the generating function for R_N/Q_N is closely related to a classical identity in the theory of partitions:

LEMMA 3 (Euler).

$$\sum_{n\geq 0}\frac{u^n}{(1-q)(1-q^2)\cdots(1-q^n)} = \frac{1}{(1-u)(1-qu)(1-q^2u)\cdots}.$$

Proof. The coefficient of $u^n q^m$ on both sides is the number of ways to write n as the sum of m nonnegative integers. (See Hardy and Wright [8] for related identities and many more details.) □

In the notation that we have been using, this identity says that $\sum_{N\geq 0} u^N/Q_N = 1/(1-u)Q(u)$. This gives a convenient way to write the generating function $T(u)=\sum_{N\geq 0}(R_N/Q_N)u^N$ in product form:

$$T(u) = \frac{1}{1-u}\sum_{N\geq 0}\frac{u^N}{Q_N} = \frac{1}{(1-u)^2 Q(u)}.$$

Now we can expand $Q(u)^{-1}$ by Lemma 2:

$$\frac{1}{Q(u)}=\frac{1}{Q_\infty}+\frac{\alpha}{Q_\infty}(u-1)+O(u-1)^2.$$

Thus $T(u)=1/(Q_\infty(1-u)^2)-\alpha/(Q_\infty(1-u))$ plus a function which is analytic for $|u|\leqq 2$ except for a simple pole at $u=2$, which implies that

$$R_N=Q_N\left(\frac{N+1}{Q_\infty}-\frac{\alpha}{Q_\infty}\right)+O(2^{-N}).$$

Since $Q_N/Q_\infty=1+O(2^{-N})$ this simplifies to

$$R_N=N+1-\alpha+O(N2^{-N}).$$

Now, the function $R_N^*=R_N-(N+1-\alpha)$ not only satisfies a simple recurrence but also converges very quickly, so we can apply the recurrence for increasing N to extend the function. From (8) we have

$$R_N^*=\frac{(N+1-\alpha)q^{N+1}}{1-q^{N+1}}+\frac{1}{1-q^{N+1}}R_{N+1}^* \tag{9}$$

which is extended by the meromorphic function

$$\begin{aligned}R(z)&=\frac{(z+1-\alpha)q^{z+1}}{1-q^{z+1}}+\frac{1}{1-q^{z+1}}R(z+1)\\&=\frac{(z+1-\alpha)q^{z+1}}{1-q^{z+1}}+\frac{(z+2-\alpha)q^{z+2}}{(1-q^{z+1})(1-q^{z+2})}+\frac{1}{1-q^{z+2}}R(z+2)\\&=\sum_{j\geqq 0}\frac{(z+1+j-\alpha)q^{z+1+j}}{(1-q^{z+1})(1-q^{z+2})\cdots(1-q^{z+1+j})}.\end{aligned} \tag{10}$$

This function is meromorphic except for simple poles at the points $z=j\pm 2\pi ik/\ln 2$ for $j\leqq -2$ and all $k\geqq 0$, and for $j=-1$ and $k>0$. Note carefully that $R(-1)$ exists (why?).

Substituting in (7), we have

$$C_N=N-\sum_{k\geqq 2}\binom{N}{k}(-1)^k(R_{k-2}^*+k-1-\alpha).$$

After applying the elementary identities $\sum_k\binom{N}{k}(-1)^k=\sum_k\binom{N}{k}k(-1)^k=0$ we have the simplified result

$$C_N=(N-1)(\alpha+1)-\sum_{k\geqq 2}\binom{N}{k}(-1)^kR_{k-2}^*. \tag{11}$$

Now, by Lemma 1 we know that

$$C_N-(N-1)(\alpha+1)=\frac{1}{2\pi i}\int_C B(N+1,-z)R(z-2)\,dz. \tag{12}$$

The same argument as for Theorem 1 shows that the right-hand side of this equation is equal to the sum of the residues of the integrand at singularities to the right of the line $x=\frac{1}{2}$, to within $O(N^{1/2})$. In this case, the poles at $1\pm 2\pi ik/\ln 2$ are all single poles. The main term is given by $N\lim_{z\to 1}R(z-2)$; the poles for $k\neq 0$ add a small oscillatory term.

The method of calculating $R(-1)$ is to express $R(z)$ in terms of a generating function which generalizes the function of Lemma 3, then to expand that function and exploit certain properties of its derivatives. Specifically, we define

$$F(u, v) = \sum_{j \geq 1} \frac{q^j u^j}{(1-qv)\cdots(1-q^j v)}.$$

This is the generating function for restricted partitions (the coefficient of $u^n v^m q^k$ is the number of ways to partition k into m parts not exceeding n). Note that, by Lemma 3,

$$F(u, 1) + 1 = \frac{1}{(1-qu)(1-q^2u)(1-q^3u)\cdots}$$

so that $F(1, 1) = Q_\infty^{-1} - 1$. Also, from Lemma 2 we have

$$F_1'(u, 1) = \frac{1}{(1-qu)(1-q^2u)(1-q^3u)\cdots}\left(\frac{q}{1-qu} + \frac{q^2}{1-q^2u} + \frac{q^3}{1-q^3u} + \cdots\right)$$

so that $F_1'(1, 1) = \alpha / Q_\infty$. Furthermore, we have

$$F(1, q^{z+1}) = \sum_{j \geq 1} \frac{q^j}{(1-q^{z+2})\cdots(1-q^{z+1+j})},$$

$$F_1'(1, q^{z+1}) = \sum_{j \geq 1} \frac{jq^j}{(1-q^{z+2})\cdots(1-q^{z+1+j})}$$

which gives, from (10), the following expansion:

$$R(z) = \frac{q^{z+1}}{1-q^{z+1}}((z+1-\alpha)(F(1, q^{z+1}) + 1) + F_1'(1, q^{z+1})).$$

From this formulation, a Taylor expansion around $z = -1$ is straightforward:

$$\frac{q^{z+1}}{1-q^{z+1}} = -\frac{1}{(z+1)\ln q} - \frac{1}{2} + O(z+1),$$

$$F(1, q^{z+1}) = F(1, 1) + (z+1)\ln q F_2'(1, 1) + O(z+1)^2$$

and

$$F_1'(1, q^{z+1}) = F_1'(1, 1) + (z+1)\ln q F_{12}''(1, 1) + O(z+1)^2$$

so that

$$R(z) = -\frac{F(1, 1) + 1}{\ln q} + \alpha F_2'(1, 1) - F_{12}''(1, 1) + O(z+1).$$

Note carefully the cancellation of $-\alpha(F(1, 1) + 1) + F_1'(1, 1)$; this is also implied by the fact that $R(z)$ exists at $z = -1$. Thus, to complete the calculation of the main term, we need only calculate $F_2'(1, 1)$ and $F_{12}''(1, 1)$. These are constants which can easily be calculated from the series representations

$$F_2'(1, 1) = \sum_{j \geq 1} \frac{q^j}{(1-q)(1-q^2)\cdots(1-q^j)}\left(\frac{q}{1-q} + \frac{q^2}{1-q^2} + \cdots + \frac{q^j}{1-q^j}\right), \tag{13}$$

$$F_{12}''(1, 1) = \sum_{j \geq 1} \frac{jq^j}{(1-q)(1-q^2)\cdots(1-q^j)}\left(\frac{q}{1-q} + \frac{q^2}{1-q^2} + \cdots + \frac{q^j}{1-q^j}\right). \tag{14}$$

Actually, we can relate $F_2'(1,1)$ to α and Q_∞, for the function $F(u,v)$ has a symmetry property which seems remarkable from an analytic standpoint (though it is more intuitive from the combinatorial interpretation). We have

$$F(u,v)-vF(qu,v)=\sum_{k\geq 1}\frac{q^k u^k(1-q^k v)}{(1-qv)\cdots(1-q^k v)}=qu(1+F(u,v)).$$

This recurrence can be telescoped as follows:

$$\begin{aligned}F(u,v)&=\frac{qu}{1-qu}+\frac{v}{1-qu}F(qu,v)\\&=\frac{qu}{1-qu}+\frac{vq^2u}{(1-qu)(1-q^2u)}+\frac{v^2}{(1-qu)(1-q^2u)}F(q^2u,v)\\&=uF(v,u)/v\end{aligned}$$

or $vF(u,v)=uF(v,u)$. (See [14] for some related identities and techniques.)

Differentiating both sides of this symmetric identity with respect to u, we find that $F_1'(1,1)=F_2'(1,1)+F(1,1)$, so $F_2'(1,1)=(\alpha-1)Q_\infty^{-1}+1$. Note that differentiating again with respect to v produces a trivial identity: there does not seem to be an easy way to express $F_{12}''(1,1)$ in terms of α and Q_∞, so we denote that constant (defined in (14)) simply by β. Collecting terms, we have shown that the residue of the integrand in (12) at $z=1$ is N times

$$\beta+1-\frac{1}{Q_\infty}\left(\frac{-1}{\ln q}+\alpha^2-\alpha\right).$$

It remains to calculate the residues of the integrand in (12) at the other singularities.

This calculation is straightforward: the residue of $(1-q^{z+1})^{-1}$ at $z=-1\pm(2\pi ik)/\ln q$ is $-1/\ln q$, and the other terms in $R(z)$ contribute a factor of $2\pi ik/Q_\infty \ln q$. The factor from $B(N+1,-z)$ is expanded exactly as for Theorem 1; thus we have the oscillatory term

$$\delta^*(N)=\frac{1}{Q_\infty \ln q}\sum_{k\neq 0}\frac{2\pi ik}{\ln q}\Gamma\left(-1-\frac{2\pi ik}{\ln q}\right)e^{2\pi ik \lg N}.$$

This completes the calculation of the coefficient of the linear term. □

For purposes of comparison, it is of some interest to compute the average number of internal nodes with both sons external in Patricia tries. This is a relatively straightforward derivation similar to the path length calculations given above, so we only sketch it here. We start with the recurrence

$$C_N^{[P]}=\sum_k \frac{1}{2^N}\binom{N}{k}(C_k^{[P]}+C_{N-k}^{[P]}),\qquad N\geq 3 \tag{15}$$

with $C_0^{[P]}=C_1^{[P]}=0$ and $C_2^{[P]}=1$. This corresponds to the functional equation

$$C^{[P]}(z)=(z/2)^2+2C^{[P]}(z/2)\,e^{z/2}$$

which transforms to

$$D^{[P]}(z)=(z/2)^2\,e^{-z}+2D^{[P]}(z/2)$$

and eventually gives the sum

$$C_N^{[P]}=\frac{1}{4}\sum_{2\leq k\leq N}\binom{N}{k}\frac{k(k-1)(-1)^k}{1-1/2^{k-1}}.$$

Knuth [9, Ex. 6.3-19] gives specific evaluations of such sums. The eventual result is that the proportion of nodes in Patricia tries with both sons external is $1/(4 \ln 2) = .3606\cdots$ plus a small oscillating term. Thus, according to this measure, digital search trees are (slightly) more balanced than Patricia tries.

4. Multiway branching. A natural generalization of the digital tree search (and trie search) algorithm studied above is to allow M-way branching (not just left and right), each node containing M links to other nodes. If $M = 2^m$ and the keys are in binary, this is conveniently implemented as follows. To search for a record with value v, set x to point to the root and b to 1, and perform the following operations until termination:

- If x is null then terminate (v not found).
- If $key(x) = v$ then terminate (v found).
- Otherwise, if the $m(b-1)+1$ through mbth bits of v represent k then set x to the kth link of x.
- Set b to $b+1$.

Here each node is assumed to have M links, numbered from 0 to $M-1$. A similar implementation of tries (with 0 keys and M links per node) is straightforward. (The generalization of binary search trees is quite different: M-ary search trees have $M-1$ keys and M links per node.)

It turns out that the analysis given above survives largely intact for the M-ary case. For example, to find the average number of nodes in a M-ary digital search tree with all links null, we begin with the fundamental recurrence:

$$C_N^{[M]} = \sum_{k_1+k_2+\cdots+k_M=N} \frac{1}{M^{N-1}} \binom{N-1}{k_1, k_2, \cdots, k_M} (C_{k_1}^{[M]} + C_{k_2}^{[M]} + \cdots + C_{k_M}^{[M]}), \quad N \geq 2, \tag{16}$$

with $C_1^{[M]} = 1$ and $C_0^{[M]} = 0$. This is proven in the same manner as (1) and (6). First, the number of nodes with all links null in a tree is exactly the number of such nodes in all the subtrees of the root, unless the tree has just one node. Second, the probability that the first subtree has k_1 nodes, the second has k_2 nodes, etc. with $k_1 + k_2 + \cdots + k_M = N-1$ is exactly

$$\frac{1}{M^{N-1}} \binom{N-1}{k_1, k_2, \cdots . k_M}.$$

Third, all the subtrees are randomly built according to the same model.

By symmetry, (16) is equivalent to

$$C_N^{[M]} = M \sum_{k_1+k_2+\cdots+k_m=N} \frac{1}{M^{N-1}} \binom{N-1}{k_1, k_2, \cdots, k_M} C_{k_1}^{[M]}, \qquad N \geq 2,$$

with $C_1^{[M]} = 1$ and $C_0^{[M]} = 0$. Now, by manipulations generalizing those leading to (2), we define the exponential generating function $C^{[M]}(z) = \sum_{N \geq 0} C_N^{[M]} z^N / N!$ and derive the following difference-differential equation:

$$\begin{aligned} \frac{d}{dz} C^{[M]}(z) &= 1 + MC^{[M]}(z/M)(e^{z/M})^{M-1} \\ &= 1 + MC^{[M]}(z/M)\, e^{(1-1/M)z}. \end{aligned} \tag{17}$$

For $M = 2$, this is exactly the equation derived from (6); moreover, none of the manipulations used for solving (6) depend in an essential way on the value of that

constant. In fact, we defined $q=1/2$ for notational convenience in that derivation: if we take $q=1/M$ in the solution, we get the solution for M-ary digital trees.

COROLLARY. *The average number of nodes with all links null in an M-ary search tree (for $M \geqq 2$) built from N records with keys from random bit streams is*

$$N\left(\beta^{[M]}+1-\frac{1}{Q_\infty^{[M]}}\left(\frac{1}{\ln M}+\alpha^{[M]}(\alpha^{[M]}-1)\right)+\delta^{[M]}(N)\right)+O(N^{1/2})$$

where the constants involved are given by

$$Q_\infty^{[M]}=\prod_{k\geqq 1}\left(1-\frac{1}{M^k}\right),$$

$$\alpha^{[M]}=\sum_{k\geqq 1}\frac{1}{M^k-1},$$

$$\beta^{[M]}=\sum_{k\geqq 1}\frac{kM^{k+1}}{(M-1)(M^2-1)\cdots(M^k-1)}\sum_{1\leqq j\leqq k}\frac{1}{M^j-1}$$

and the oscillatory term is

$$\delta^{[M]}(N)=\frac{1}{Q_\infty \ln M}\sum_{k\neq 0}\frac{2\pi ik}{\ln M}\Gamma\left(-1+\frac{2\pi ik}{\ln M}\right)e^{2\pi ik\lg N}.$$

Proof. Immediate from the discussion above. □

Table 1 below gives the approximate value of $C_N^{[M]}$ and the various constants for small values of M.

TABLE 1

M	$Q_\infty^{[M]}$	$\alpha^{[M]}$	$\beta^{[M]}$	$C_N^{[M]}/N$
2	.28879	1.60670	7.74313	.37205
3	.56013	.68215	.71399	.47602
4	.68854	.42110	.22414	.53054
8	.85941	.16097	.02748	.62506
16	.93359	.07085	.00510	.68928

Note carefully that, in a perfectly balanced M-ary tree, about $(M-1)/M$ of the nodes have all links null. Measured against this standard, the constants in the last column of the table show that digital search trees are about 70%-75% balanced for small M. That is, the ratio between the constants given and $(M-1)/M$ is between .70 and .75. Of course, as $M\to\infty$, this ratio approaches 1.

5. General framework. The methods that we have used in the previous sections can be applied to study many other properties of digital search trees. If $X(T)$ and $x(T)$ are parameters of trees satisfying

$$X(T)=\sum_{\substack{\text{subtrees } T_j\\ \text{of the root of } T}} X(T_j)+x(T) \tag{18}$$

then the exponential generating functions for the expectations X_N and x_N for an M-ary digital search tree built from N records with keys from random bit streams satisfy

$$X'(z)=MX(z/M)\,e^{(1-1/M)z}+x(z).$$

This is derived in exactly the same manner as (17). Now, in terms of the transform generating functions $Y(z)=e^{-z}X(z)$ and $y(z)=e^{-z}x(z)$ this becomes

$$Y'(z)+Y(z)=MY(z/M)+y'(z)+y(z). \tag{19}$$

This leads to a nonlinear recurrence like (8) satisfied by Y_N, with the solution sought given by $X_N=\sum_k \binom{N}{k} Y_k$. If the quantity $(-1)^k Y_k$ is sufficiently well behaved, we can study its asymptotics and find a function Y_k^* which:

(i) is simply related to Y_k so that $\sum_k \binom{N}{k}(Y_k-(-1)^k(Y_k^*))$ is easily evaluated,

(ii) satisfies a recurrence of the form $Y_{N+1}^*=(1-g(M, N))Y_N^*+f(M, N)$,

(iii) goes to zero quickly as $N\to\infty$.

Depending on the nature of $g(M, N)$, $f(M, N)$ and the speed of convergence, conditions (ii) and (iii) may allow the recurrence to be turned around to extend Y_N^* to the complex plane and so allow the desired expectation to be computed by evaluating the sum $\sum_k \binom{N}{k}(Y_k-(-1)^k(Y_k^*))$ as detailed in the previous sections.

For example, this method could be used to find the distribution of occupancy of nodes in M-ary digital search trees, and many other problems.

The same type of generalization applies to the study of tries (and Patricia tries), and the simpler nature of the recurrences follows through the generalization. For example, the exponential generating functions for the expectations X_N and x_N of parameters of trees satisfying (18) for a random trie built from N records from random bit streams is

$$X(z)=MX(z/M)\,e^{z/M}+x(z) \tag{20}$$

which is considerably easier to deal with.

These methods allow quite full analysis of the types of trees considered, and they clearly expose the fundamental differences and similarities among the analyses.

A final note: the reader who is still awake may have noticed that the "transforms" that are essential to these computations are not at all arbitrary functions. Indeed, if $x(z)=\sum_{N\geq 0} x_N z^N/N!$ is the exponential generating function for the expectation of a parameter X, then $Y(z)=e^{-z}x(z)$ is the expectation of X if the number of keys is Poisson with parameter z.

For digital search trees, tries, and Patricia tries, this function satisfies a simple functional equation like (20) which makes it amenable to direct solution by Mellin transform techniques. (See [6] for details of the application of this method to the analysis of tries: essentially the Mellin transform is trivially computed by taking the transform of both sides of the functional equation, and then a singularity analysis is done for the reverse transform. Another example of this technique may be found in [4].) The relationships among the Bernoulli and Poisson models and Mellin transform and Rice's method of asymptotic analysis are a fruitful area for further study. More details will be reported in a future paper.

Acknowledgments. The authors would like to express their gratitude to Janet Incerpi, who is still awake. Also, thanks are due to our Viennese friends, especially Helmut Prodinger, for helping us find several bugs in the manuscript.

REFERENCES

[1] E. G. COFFMAN, JR. AND J. EVE, *File structures using hashing functions*, Comm. ACM, 13, 7 (1970), pp. 427-436.

[2] E. DAVIES, *Integral Transforms and Their Applications*, Springer-Verlag, 1978.

[3] P. FLAJOLET, *Approximate counting: a detailed analysis*, BIT, to appear.

[4] P. FLAJOLET AND C. PUECH, *Partial match retrieval of multidimensional data*, INRIA Research Report, 1983.
[5] P. FLAJOLET, J. C. RAOULT, AND J. VUILLEMIN, *On the average number of registers required for evaluating arithmetic expression*, Theoret. Comput. Sci., 9 (1979), pp. 99–125.
[6] P. FLAJOLET, M. REGNIER, AND R. SEDGEWICK, *Mellin transform techniques for the analysis of algorithms*, in preparation.
[7] P. FLAJOLET AND R. SEDGEWICK, *The asymptotic evaluation of some alternating sums involving binomial coefficients*, INRIA Research Report, 1983.
[8] G. HARDY AND E. WRIGHT, *An Introduction to the Theory of Numbers*, Clarendon Press, Oxford, 1960.
[9] D. E. KNUTH, *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
[10] ———, *The average time for carry propagation*, P. Kon Ned A, 81 (1978), pp. 238–242.
[11] A. G. KONHEIM AND D. J. NEWMAN, *A note on growing binary trees*, Discrete Math., 4 (1973), pp. 57–63.
[12] N. E. NÖRLUND, *Vorlesungen über Differenzenrechnung*, Chelsea, New York, 1954.
[13] M. REGNIER, *Evaluation des performances du hachage dynamique*, Thèse de 3ème cycle, Université de Paris-Sud, 1983.
[14] G. PÓLYA AND G. SZEGÖ, *Problems and Theorems in Analysis* I, Springer-Verlag, Berlin, 1976.
[15] R. SEDGEWICK, *Data movement in odd-even merging*, this Journal, 7 (1978), pp. 239–273.
[16] ———, *Algorithms*, Addison-Wesley, Reading, MA, 1983.
[17] E. WHITTAKER AND G. WATSON, *A Course of Modern Analysis*, Cambridge Univ. Press, Cambridge, 1927.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
010

REDUCING MULTIPLE OBJECT MOTION PLANNING TO GRAPH SEARCHING*

J. E. HOPCROFT† AND G. T. WILFONG‡

Abstract. The motion planning problem for multiple objects is studied where an object is a 2-dimensional region whose sides are line segments parallel to the axes of $\mathbf{R}^2$ and translations are the only motions allowed. Towards this end we analyze the structure of configuration space, the space of points that correspond to positions of the objects. In particular, we consider CONNECTED, the set of all points in configuration space that correspond to configurations of the objects where the objects form one connected component. We show that CONNECTED consists of faces of various dimensions such that if there is a path in CONNECTED between two 0-dimensional faces (vertices) of CONNECTED then there is a path between them along 1-dimensional faces (edges) of CONNECTED. It is known that if there is a motion between two configurations of CONNECTED then there is a path in CONNECTED between the configurations. Thus the existence of a motion between two vertices of CONNECTED implies a motion corresponding to a path along edges of CONNECTED. Hence the motion planning problem is reduced from a search of a high dimensional space to a graph searching problem.

From this result it is shown that motion planning for rectangles in a rectangular boundary is in PSPACE. Since it is known that the problem is PSPACE-hard, we conclude it is a PSPACE-complete problem.

Key words. motion planning, robotics, polynomial space, computational complexity, algorithm

AMS(MOS) subject classifications. 05C40, 68C25, 55N99

Introduction. Finding collision-free motions of objects is an important problem in robotics and computer-aided manufacture. There are two general ways of posing motion problems. One problem is to actually plan a motion if one exists. That is, given initial and final configurations of the objects compute a collision-free motion if one exists and otherwise report that no such motion exists. This is what is called the *motion planning problem.* Another motion problem is the decision problem "Given initial and final configurations, is there a collision-free motion of the object(s) from the initial to the final configuration?". This problem is referred to as the *mover's problem.* In this paper we will show that the motion planning problem for rectangles enclosed in another rectangle can be solved in polynomial space.

Probably the most influential work in motion planning is due to Lozano-Perez and Wesley [6] whose paper brought to the academic community the realization that motion planning was a rich mathematical area with practical applications. They considered the motion of a single object in the presence of obstacles by shrinking the object to a point and growing the obstacles in such a manner that the original object could be moved from A to B if and only if the point could be moved from A to B in the presence of the enlarged obstacles. The enlarged obstacles were called configuration space obstacles thereby introducing the notion of representing a configuration by a point in a space. The points inside a configuration space obstacle represented illegal configurations where the moving object overlapped an obstacle.

The next major contribution was a series of papers by Schwartz and Sharir [10]-[12]. They provided a precise framework for general motion planning sufficient

* Received by the editors January 9, 1985, and in revised form May 20, 1985. This research was supported in part by the National Science Foundation under grant ECS-8312096 and a National Science and Engineering Research Council graduate scholarship.

† Department of Computer Science, Cornell University, Ithaca, New York 14853.

‡ Department of Computer Science, Cornell University, Ithaca, New York 14853. Present address, AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

to encompass not only coordinated motion of multiple rigid objects but also of objects whose shape could change such as a robot arm. By applying techniques from the theory of reals they proved that even in this general setting motion planning was decidable. In fact, for problems with a fixed bounded number of degrees of freedom they gave a polynomial time algorithm. The consideration of multiple objects introduced a new aspect to motion planning; namely, the coordination of motion. Simply calculating trajectories was no longer sufficient.

The complexity of motion problems has been considered by a number of researchers. For instance, in [9] Reif showed that the mover's problem for multiple polyhedra freely linked together was PSPACE-complete. Hopcroft, Joseph, and Whitesides [3] showed that the mover's problem for linkages, even in two dimensions, was PSPACE-hard. Hopcroft, Schwartz, and Sharir [4] considered the mover's problem for rectangles inside a 2-dimensional rectangular box and they showed that even if translation were the only allowable motion then the problem was PSPACE-hard. Thus the problem of actually planning a motion of rectangles in a box is PSPACE-hard. However, there was a fundamental difficulty in establishing that these problems were in PSPACE. The difficulty is that there could conceivably be a motion but not a motion that could be described in polynomial space. For example, what if the only possible motions involved moving objects to positions that were not even algebraically related. Although the work of Schwartz and Sharir [11] showed this not to be the case, it did not rule out moving objects to algebraic positions of high degree, too high to represent in an obvious manner in a polynomial amount of space.

Hopcroft and Wilfong [5] considered motion as a special case of a transformation. In their setting an object was a parameterized mapping from a canonical object to a region of 3-space. A motion was a path in the space of parameters (i.e., configuration space) and thus included not only translations and rotations but growth and parameterized continuous deformations. They considered configurations of the objects (called connected configurations) in which the objects formed one connected piece. That is, for connected configurations, if G is the graph with a vertex for each object and an edge between vertices whenever the corresponding objects touch in the configuration then G is a connected graph. It was shown that any motion of n objects between two connected configurations could be transformed to one in which the objects formed connected configurations throughout the motion. This reduced motion planning from searching an entire space to searching the surface of some generalized configuration space obstacle (i.e. the set of all connected configurations). An appealing method of searching the surface of the obstacle would be to show that the space of connected configurations C consists of various dimensional surfaces and then prove that if a motion exists on n-dimensional surfaces of C between points on a lower dimensional surface then a motion exists on $(n-1)$-dimensional surfaces of C between these points. If the points happen to be on 0-dimensional surfaces, then the above method would be repeated until finally it is shown that there is a motion along 1-dimensional surfaces (edges) of C between the 0-dimensional surfaces (vertices) of C. If the vertices of C were connected by paths consisting solely of vertices and edges, then the search of the geometrical surface could be reduced to the search of a purely combinatorial structure, i.e., the graph consisting of the vertices and edges of C. Although the number of vertices of the graph may be astronomical, the entire graph need not be constructed. By means of some heuristic, a search could proceed by generating an edge only when it was to be traversed in the search. If in practical cases, where objects are being moved in a relatively open work space, only a small number of vertices need be generated, then a practical and effective algorithm might be possible although the worst-case complexity

would be prohibitive. The search process envisioned is somewhat analogous to linear programming where only a small number of vertices of a polytope are examined.

In this paper we study the motion of two-dimensional objects with straight line surfaces. We call these surfaces *faces*. The objects will be allowed to translate but not to rotate. The position of each object is then determined by the location of some fixed point on the object. Thus if there are n objects moving about then the arrangements of the objects are in an obvious one-to-one relationship with a point in a $2n$-dimensional Euclidean space. That is, in this case configuration space is $\mathbf{R}^{2n}$. We call the points in configuration space (as well as the corresponding arrangements of the objects) *configurations*. Configuration space is partitioned as follows. We define classes of configurations by specifying the faces of the objects that touch one another. For example, consider the case where there are just two rectangles A and B. In this case there are eight classes. Four classes correspond to where one face of A touches one face of B (e.g. the class of all configurations where the top face of A touches the bottom face of B) and four classes correspond to configurations where two pairs of faces touch (e.g. the class of all configurations where the top, left corner of A touches the bottom, right corner of B). The classes corresponding to configurations in which the objects form a connected arrangement are actually various dimensional hyperplanes in configuration space. The classes that contain only one configuration are called vertex configurations and the classes that correspond to 1-dimensional hyperplanes are called edges. We show that if there is any motion of the objects between two vertex configurations then there is a motion that follows edges from vertex to vertex. This edge connectedness of the vertex configurations is not true for more general situations in which rotation is allowed.

Consider the situation in Fig. 1 where rotations are allowed. This is an example of a vertex configuration but the only other vertex configuration that can be reached from this one along edges is the one resulting from moving the objects as a rigid piece to the left. Notice that when corners a and b of objects A and B come apart, object A is free to rotate by sliding c along L and d along M, and object B is free to rotate and move corner e along N. Such configurations lie on a higher dimensional surface and not on an edge. A similar phenomenon occurs if any other constraint is moved.

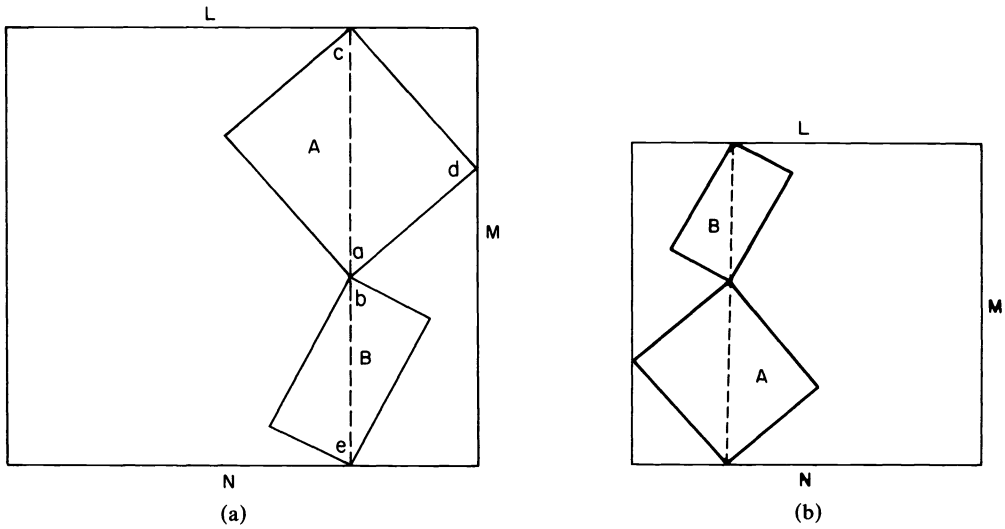


FIG. 1. *Vertices not connected by edges when rotations allowed.*

Thus the vertex configuration shown in Fig. 1b cannot be reached by edges from the vertex configuration shown in Fig. 1a.

Using the result about edge connectedness of vertex configurations, we show how to calculate a motion between two configurations in polynomial space. This is done by showing that a vertex can be stored in polynomial space since it is the solution of a linear system of equations with $O(n)$ unknowns where n is the number of objects. Thus, a nondeterministic algorithm using polynomial space can guess the path vertex by vertex and the result follows from the fact that PSPACE = NSPACE. Thus, deciding if a motion of rectangles within a rectangular boundary exists is in PSPACE. Combining this result with the result of [4], we conclude that the problem of deciding if a motion of rectangles in a rectangular boundary exists is PSPACE-complete.

1. Definitions. This section defines the various concepts that will be used throughout the paper. We will describe how configuration space will be partitioned into various dimensional surfaces (which we will call faces). The main goal of the next section will be to show that the existence of a path in the face between two points in the boundary of the face implies a path in the boundary between the points. It will then be shown in § 2 that the boundary consists of surfaces that are of lower dimension than the dimension of the face and so the search for a path will be reduced to searching lower dimensional surfaces.

We will say that a function p is a *path* in a set S from x to y if p is a homeomorphism $p\colon [0, 1] \to S$ such that $p(0) = x$ and $p(1) = y$. A set is said to be *path connected* if between any two points in the set there is a path completely contained in the set.

In this paper we will consider the motion of *objects* that are 2-dimensional, path connected, compact regions with a finite number of faces that are closed line segments parallel to one of the two axes of $\mathbf{R}^2$. The faces of an object that are parallel to the x-axis of $\mathbf{R}^2$ are called *horizontal faces* of the object and the ones that are parallel to the y-axis are called *vertical faces* of the object. Each object is the closure of its interior but is not necessarily convex. Figure 2 illustrates some typical objects. Unless otherwise stated we will assume there are $n+1$ objects. The objects will be denoted by $A_0, A_1, \cdots, A_n$. Object A_0 has its position fixed and the others are allowed to translate but not to rotate. Fix some point on each object and call it the *origin* of the object. Thus the location of object A_i, $1 \leqq i \leqq n$, is completely determined by the position (x_i, y_i) of its origin in $\mathbf{R}^2$. Therefore a point $(x_1, y_1, \cdots, x_n, y_n) \in \mathbf{R}^{2n}$ completely defines an arrangement of the objects. We will call such a point a *configuration.* When we speak about a configuration, we may be talking about a point in $\mathbf{R}^{2n}$ or an arrangement of the objects but the two notions are interchangeable. The set of all configurations, $\mathbf{R}^{2n}$,

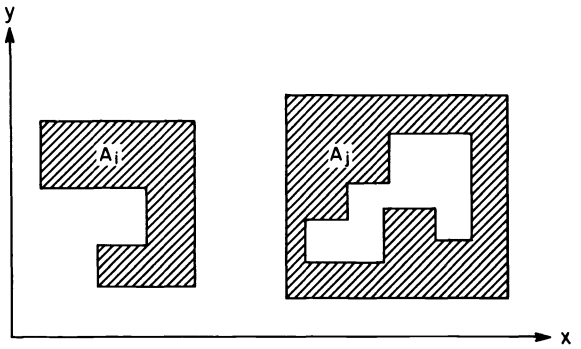


FIG. 2. *Examples of objects.*

is called *configuration space.* For notational purposes we use lower case v's to denote points in configuration space.

Notice that a path in configuration space corresponds to a motion of the objects. Clearly the relationship "there is a path from x to y" is a symmetric, reflexive and transitive relation (i.e. it is an equivalence relation). Thus a space can be partitioned into components called *path connected components* where two points are in the same path connected component if and only if there is a path between them. Since a path in configuration space corresponds to a motion of the objects, deciding whether two points are in the same path connected component of some subspace of configuration space is the same as determining whether there is a motion of the objects between the two configurations such that all intermediate configurations during the motion are configurations in the specified subspace. Also because of this relationship between the notion of a path in configuration space and a motion of the objects, the words "path" and "motion" can be used interchangeably without causing confusion. Thus when we speak of a motion within a subspace of configuration space we mean that every point on the path in configuration space that represents that motion is in the specified subspace.

We are interested in finding a motion of the objects between configurations such that during the motion no two objects "collide". To make this notion more rigorous we first define some subspaces of configuration space. Two objects are said to *overlap* if the intersection of their interiors is nonempty. Let NONOVERLAP denote the set of all configurations in which no two objects overlap. Thus the motions that are of interest are those motions within NONOVERLAP. Throughout this paper when we say that a motion exists we will usually mean that a motion in NONOVERLAP exists unless otherwise stated. Two objects *touch* if the intersection of the objects is nonempty but the objects do not overlap.

For every configuration v we define the *graph of* v, denoted by G_v, to be the graph with a node corresponding to each object of v and an edge between nodes if the corresponding objects intersect in v. We say that a configuration is a *connected configuration* if the graph of the configuration is connected. Define CONNECTED to be the set of all connected configurations in NONOVERLAP. Notice that if two objects in a configuration $v \in$ CONNECTED intersect then they must touch because $v \in$ NONOVERLAP. By the result in [5] we know that if there is a motion between two configurations both of which are in CONNECTED then there is a motion completely contained in CONNECTED between them. Hence it is sufficient to deal with motions in CONNECTED.

Next we will describe how the subspace CONNECTED will be partitioned. Towards this end we classify the various faces of the objects as follows. Remember that a face of an object is a closed straight line segment parallel to one of the axis of $\mathbf{R}^2$. Let a_i be a face of object A_i and let b be some point of a_i such that b is not an endpoint of a_i. Notice that we can draw a disc with center b and with small enough radius so that the half-disc on one side of a_i does not intersect the interior of A_i and the half-disc on the other side of a_i is contained in the object A_i. See Fig. 3. If a_i is a horizontal face and the half-disc above a_i is the one that does not intersect the interior of A_i, then we say that a_i is an *upper face* of A_i. If however, the half-disc below a_i is the one that does not intersect the interior of A_i, then a_i is a *lower face* of A_i. Similarly if a_i is a vertical face and the half-disc to the left of a_i does not intersect the interior of A_i then we call a_i a *left face* of A_i. If the half-disc to the right of a_i is the one that does not intersect the interior of A_i then a_i is called a *right face* of A_i.

We say that the face a_i of object A_i *touches* face a_j of object A_j if the intersection of a_i and a_j is nonempty and a_i and a_j are an upper and lower face respectively, or

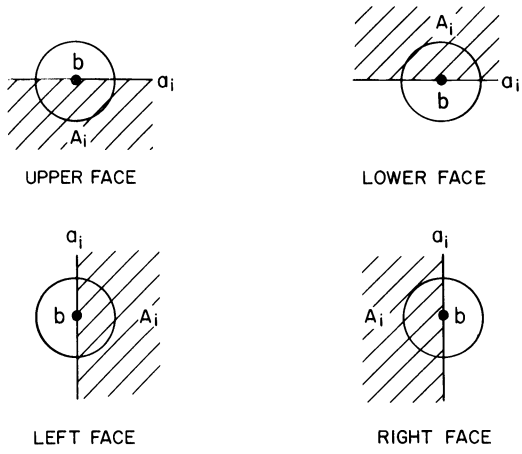


FIG. 3. *Classification of faces.*

a left and right face respectively. Thus objects A_i and A_j touch if and only if some face of A_i touches some face of A_j. For every pair of faces that touch, we can write a corresponding linear equation relating the horizontal or vertical distance between the origins of the objects as follows. Suppose face a_1 of object A_1 touches face a_2 of object A_2 where one of a_1 and a_2 is a left face and the other is a right face. Let t be the horizontal distance between the origins of A_1 and A_2 whenever a_1 touches a_2. Assume that the origin of A_2 is no further to the left than the origin of A_1 whenever a_1 and a_2 touch. Thus $t = x_2 - x_1$ where the position of the origin of A_i is given by (x_i, y_i) for $i = 1, 2$. Then the corresponding linear equation relating the horizontal distance between the origins is $x_2 - x_1 = t$. If the faces that touch one another are upper and lower faces, then the corresponding linear equation would be $y_2 - y_1 = t$.

A *description* is a predicate that is a conjunction of clauses of the form "face a_i of object A_i touches face a_j of object A_j." If "face a_i of object A_i touches face a_j of object A_j" is a clause of description D then we say that face a_i and face a_j touch according to D. A configuration v is said to *satisfy* the description D if each pair of faces that touch according to D touch in v.

With each description D we associate a linear system $M_D v = c_D$ where the equations in the system are the linear equations corresponding to the clauses in D as described above. Notice that a configuration can satisfy the system of linear equations without satisfying the predicate D. In Fig. 4 the objects have faces that are aligned and so satisfy the linear equation but the faces do not touch.

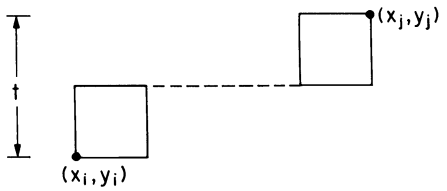


FIG. 4. *Objects satisfy the linear system but do not touch.*

We say that a configuration v *satisfies a description* D *exactly* if v satisfies D and the only faces that touch in v are those implied by D. Notice that for each configuration v there is a description D such that v satisfies D exactly. Thus when we speak of the matrix M_v for configuration v we mean the matrix M_D where D is the description

that v satisfies exactly. Throughout this paper we will consider only descriptions D such that any configuration that satisfies D is a connected configuration.

Define H_D to be the set of all configurations that satisfy the linear system corresponding to D. That is $H_D = \{v | M_D v = c_D\}$. Thus if $k = \text{rank}\,(M_D)$ then H_D is a $(2n-k)$-dimensional hyperplane.

Let E_D be the set of all configurations in NONOVERLAP that satisfy description D exactly. Notice that E_D is not necessarily path connected. In Fig. 5 the description D states that the lower faces of A and C touch the upper face of B. However there is one path connected component of E_D where A is to the left of C and another path connected component of E_D where A is to the right of C. We will denote the path connected components of E_D by $P_D(1), \cdots, P_D(t_D)$. Let $K_D(i)$ be the closure of $P_D(i)$ in H_D. The notation we will use to denote the closure of a set S in the space B is $cl_B(S)$. Each $K_D(i)$ is called a *face* of CONNECTED of dimension d if rank $(M_D) = 2n-d$. We will refer to a 0-dimensional face of CONNECTED as a *vertex* of CONNECTED. Also, a 1-dimensional face of CONNECTED will be called an *edge* of CONNECTED.

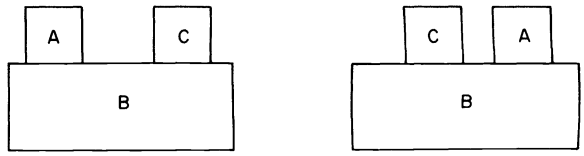


FIG. 5. *E_D with two path connected components.*

Notice that a face as defined here may not conform to what one would usually call a face. For example, in Fig. 6 consider faces K_1 and K_2. Although they are collinear surfaces with nonempty intersection, they are two distinct faces by the definitions used in this paper.

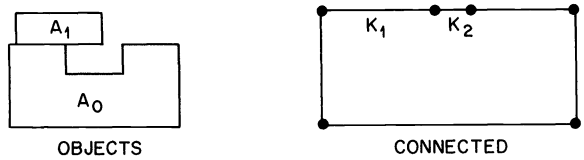


FIG. 6. *Surface of* CONNECTED *divided into more than one face.*

Since a configuration can not satisfy more than one description exactly, the sets of configurations that satisfy descriptions exactly partition CONNECTED. As noted earlier, not all E_D's are path connected, and since distinct path connected components must be disjoint, we can further partition CONNECTED into the path connected components of the sets of configurations that satisfy descriptions exactly. That is, we partition CONNECTED into the various $P_D(i)$'s whose closures are the faces of CONNECTED.

We have now divided up CONNECTED into faces of dimension $2n$ or less. In this sense CONNECTED is very much like a polytope in that it is made up of various dimensional surfaces. Also, suppose F_1 and F_2 are two faces of CONNECTED that intersect one another and the dimension of F_i is n_i. Then, as will be shown in § 3, the intersection of the two faces consists of faces whose dimensions are lower than the maximum of n_1 and n_2. This is a property shared by polytopes. In § 2 we will discuss

a property of the faces of CONNECTED that is not a general property of polytopes. In particular we will show that a face $K_D(i)$ of CONNECTED and its boundary $K_D(i) - P_D(i)$ have the same number of path connected components. We will then conclude that if there is a path in a face between two points in the boundary of the face then there is a path in the boundary of the face between the points. In Fig. 7 we see that the face a of the general polytope shown consists of one path connected component whereas its boundary contains two path connected components b and c. Clearly there is no path from x to y in the boundary of a even though there is a path p from x to y in a.

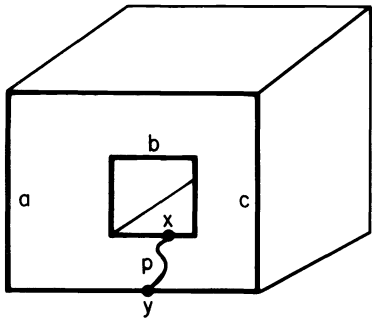


FIG. 7. *Boundary of face of polytope not path connected.*

For illustration, think of CONNECTED as the surface of a cube in $\mathbf{R}^3$. Then CONNECTED consists 26 faces, 8 of which are vertices, 12 of which are edges, and 6 of which are the 2-dimensional "sides" of the cube. The boundary of any of the 2-dimensional faces is path connected and consists of 8 faces of CONNECTED of lower dimension (4 vertices and 4 edges).

Using the result of § 2 which shows that a path in a face between two points on the boundary of the face implies the existence of a path in the boundary of the face between the points and another result of § 2 which says that the boundary of a face consists of faces of lower dimension, we will show in § 3 that a path in CONNECTED between two vertices of CONNECTED implies a path along edges of CONNECTED between the vertices.

2. Existence of paths in the boundary of a face. We will now show that if there is a path in $K_D(i)$, a face of CONNECTED, between two configurations in $K_D(i) - P_D(i)$, the boundary of the face, then there is a path between them in the boundary. This is accomplished by first showing that $K_D(i) - P_D(i)$ has the same number of path connected components as $K_D(i)$. To do this, we will need the following facts:

(1) H_D is contractible to a point and is path connected.

(2) $K_D(i)$ is closed in H_D.

(3) $H_D - P_D(i)$ is closed in H_D and is path connected.

Note that a set S is contractible to a point $y \in S$ if there is a continuous function $f\colon S \times [0, 1] \to S$ such that

$$\left.\begin{aligned} f(x, 0) &= x \\ f(x, 1) &= y \end{aligned}\right\} \quad x \in S,$$

$$f(y, t) = y, \qquad t \in [0, 1].$$

Since H_D is a hyperplane (1) is immediate and (2) is trivial because of the definition of $K_D(i)$. Lemmas 2.1 to 2.4 will be used to establish (3). The above facts along with Lemma 2.5 allow us to conclude in Lemma 2.6 that a face and its boundary have the same number of path connected components. Starting with Lemma 2.7 we will show that each path connected component of a face of CONNECTED contains exactly one path connected component of its boundary. Thus if x and y are two points in the boundary of a face $K_D(i)$ and there is a path in $K_D(i)$ between them (i.e., they are in the same path connected component C of $K_D(i)$), then there must be a path in the boundary of $K_D(i)$ between them (i.e. they are in the same path connected component of the boundary of $K_D(i)$ because C contains only one such path connected component).

First it will be shown that E_D, the set of configurations that satisfy the description D exactly is open in H_D, the set of configurations that satisfy the linear system of the description. We will then conclude that those configurations that satisfy the linear system of the description but are not in some fixed path connected component of the configurations that satisfy the description exactly, form a closed set.

LEMMA 2.1. *E_D is open in H_D.*

Proof. Let $v \in E_D$. We wish to show that there is an open ball B about v such that $B \subseteq E_D$. By definition v satisfies D exactly and $v \in$ NONOVERLAP. Let $\varepsilon > 0$ be the minimum distance between any two right and left faces or upper and lower faces of objects such that the faces do not touch in v. Then any configuration v' in H_D where the objects are closer than $\varepsilon/2$ from their positions in v is such that no faces touch in v' that do not in v. That is, there is an open ball B in H_D about v such that any faces that touch in $v' \in B$ also touch in v. Also we can conclude that $B \subseteq$ NONOVERLAP because $v \in$ NONOVERLAP.

Suppose $v' \in B$ and there is a face a_1 of A_1 and a face a_2 of A_2 that touch according to D, and hence touch in v, but do not touch in v'. Since B is a ball, between v' and v there is a path contained in $B \subseteq H_D$. Let m be the motion corresponding to the path. Throughout m, A_1 and A_2 must be aligned along a_1 and a_2. Since a_1 and a_2 touch in v but not in v' and m is continuous, we conclude that there is a point in the motion when A_1 and A_2 touch at corners. That is, at this point there are two faces that touch but they do not touch in v. This contradicts the property of B that was shown above.

Therefore, for any $v' \in B$, faces touch in v' if and only if they touch according to D and $v' \in$ NONOVERLAP. That is $B \subseteq E_D$ and so E_D is open in H_D. □

We can now prove one of the facts that we have stated will be needed in order to establish that a path exists in the boundary of a face of CONNECTED between two configurations in the boundary whenever there is a path in the face between the configurations.

LEMMA 2.2. *For any $P_D(i)$, $H_D - P_D(i)$ is closed in H_D.*

Proof. H_D is locally pathwise connected and E_D is open in H_D. Thus any path connected component of E_D is open. See Willard [14]. In other words, $P_D(i)$ is open in H_D and so $H_D - P_D(i)$ is closed in H_D. □

In order to show that $H_D - P_D(i)$ is path connected, we introduce the following notion. Let D be some description and S be a subset of the objects. Choose one of the objects of S(take it to be A_0 if $A_0 \in S$) to be considered the object of S whose position is fixed. We construct a matrix $M(S)$ from the matrix M_D for the objects in S as follows. There is a column in $M(S)$ for x_i and y_i such that A_i is an object in S and A_i is not the chosen fixed object. Let A_{i_0} be the fixed object of S. Let r be the sum of some of the rows of M_D. If r has a 1 in a column corresponding to an object in S and a -1 in a column corresponding to an object in S and all other entries in r are

0, then there is a row in $M(S)$ which is r with the columns not corresponding to the moveable objects in S removed. If A_0 is in S then if r has one 1 in a column corresponding to an object in S and all other entries 0 then also add a row to $M(S)$ with a 1 in the same column and all other entries 0. The matrix $M(S)$ has the same dependencies between the variables corresponding to objects in S as the matrix M_D. If the rank of $M(S)$ is full and S is maximal with respect to this property, then we say that S is a *vertex object* of D. Intuitively, a vertex object of a description D is a maximal collection of the objects that have the same positions relative to one another in any configuration that lies in the hyperplane H_D. Thus if one is considering only configurations in a particular H_D then one could think of each vertex object as one rigid object. As we mentioned before, for every configuration there is a description such that the vertex satisfies that description exactly. Thus when we speak of the vertex objects of a configuration we mean the vertex objects of the description that the configuration satisfies exactly.

In Fig. 8 there are two vertex objects $S_1 = \{A_0, A_1\}$ and $S_2 = \{A_2, A_3, A_4\}$. Notice that although taking $S_3 = \{A_2, A_3\}$ results in a coefficient matrix of full rank, S_3 is not maximal and so $\{A_2, A_3\}$ is not a vertex object.

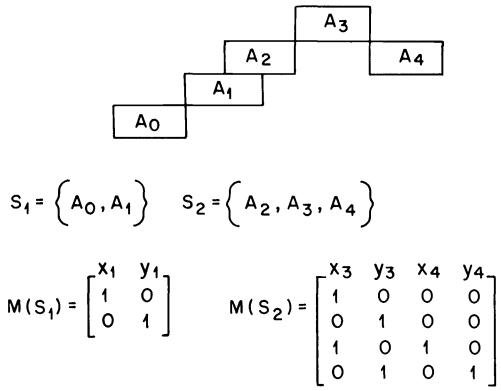


FIG. 8. *Two vertex objects.*

LEMMA 2.3. *Let D be such that* rank $(M_D) \leqq 2n-2$. *Then D has at least three vertex objects.*

Proof. If D has only one vertex object then by definition rank $(M_D) = 2n$. Suppose D has exactly two vertex objects and one has m_1 objects and the other has m_2 objects where $m_1 + m_2 = n+1$ the total number of objects. Then the number of independent rows of M_D that correspond to two objects of the first vertex object touching is $2(m_1 - 1)$. Similarly we have $2(m_2 - 1)$ independent rows for the second vertex object. Since the two vertex objects must touch, there must be an additional independent row. Thus rank $(M_D) \geqq 2(m_1 + m_2 - 2) + 1 = 2(n-1) + 1 = 2n - 1$. Hence if rank $(M_D) \leqq 2n-2$ then D has at least three vertex objects. □

We are now in the position to prove the last result that we stated would be needed in order to establish that a path exists in the boundary of a face between two points in a face if a path in the face exists between the points. In particular, we show that the complement of any of the path connected components $P_D(i)$, with respect to the hyperplane H_D, is path connected in the case where the dimension of H_D is greater than 1.

LEMMA 2.4. *If* rank $(M_D) \leqq 2n-2$ *then* $H_D - P_D(i)$ *is path connected.*

Proof. It will be shown that there is a fixed configuration v_0 in $H_D - P_D(i)$ such that there is a path in $H_D - P_D(i)$ from any v in $H_D - P_D(i)$ to v_0. The existence of such a v_0 establishes that $H_D - P_D(i)$ is path connected.

By Lemma 2.3 we know that D has at least three vertex objects. Notice that if B is a vertex object of a configuration in H_D, then B is some union of vertex objects of D. Let $v_0 \in H_D - P_D(i)$ be a configuration such that if any two vertex objects of D are held fixed to their positions in v_0, then any motion of the remaining vertex objects of D can not result in a connected configuration and thus can not result in a configuration in E_D. Such a configuration can be visualized as one where the three or more vertex objects of D are moved so far apart that no two can be joined by a "bridge" consisting of the remaining ones.

We will show that there is a path in $H_D - P_D(i)$ from any configuration in $H_D - P_D(i)$ to v_0. We proceed in two steps. First, we show how to move within $H_D - P_D(i)$ from any configuration in $H_D - P_D(i)$ to one in $H_D - E_D$. Then we show how to move within $H_D - P_D(i)$ from any configuration in $H_D - E_D$ to v_0.

Let $v \in H_D - P_D(i)$. If $v \notin E_D$ then our first step is completed. Suppose $v \in E_D$ but $v \notin P_D(i)$. Move a vertex object along an edge of another vertex object that it is touching until a new pair of faces touch. The resulting configuration is in $H_D - E_D$ (because there are faces that touch in the configuration for which there is not a corresponding clause in D). The entire motion except the final configuration is in $E_D - P_D(i)$ and so the motion is in $H_D - P_D(i)$.

Next we show that any $v \in H_D - E_D$ can be moved in $H_D - E_D$ (and hence in $H_D - P_D(i)$) to v_0. Let B_1 and B_2 be two vertex objects of D that touch according to D but in v they either touch along faces not specified by D, overlap one another, or do not touch along some face specified by D. Without loss of generality we can assume that B_1 is the vertex object that contains A_0, the fixed object. Thus B_1 is in the same position as it is in v_0. Since B_1 and B_2 touch according to D their relative positions have only one degree of freedom in H_D. Fix the relative positions of B_1 and B_2 as they are in v. This adds one constraint to the system. Since rank $(M_D) \leqq 2n-2$ there is still a collection C of vertex objects that is unconstrained in one of the x or y directions. Notice that any configuration in which B_1 and B_2 have the same positions that they have in v must be a configuration in $H_D - E_D$.

Without loss of generality assume that C is unconstrained in the x direction. Move C until the distance from C to B_1 in the x direction is so great that no motion of B_2 can create a configuration in CONNECTED and hence no motion of B_2 will create a configuration in E_D. Then move B_2 until it is in its position in v_0. By construction this motion must be in $H_D - E_D$. The motion of B_2 may require moving objects in C but only in the y direction so this does not cause a problem. Then since B_1 and B_2 are in the same positions as they are in v_0 and by the definition of v_0 we can move the remaining vertex objects to their positions in v_0 and the motion will not be in CONNECTED and hence will not be in E_D. □

The following lemma shows that a certain sequence of groups is an exact sequence where an exact sequence

$$G_1 \xrightarrow{h_1} G_2 \xrightarrow{h_2} G_3 \xrightarrow{h_3} \cdots \xrightarrow{h_n} \{0\}$$

is such that the image of h_i, Im (h_i), is the kernel of h_{i+1}, ker (h_{i+1}) where G_i is a group and h_i is a homomorphism. Let $\mathbf{Z}$ be the group of integers under addition. The notation H_1 and H_0 in the lemma is as follows:

(i) $H_0(S)$ is the zeroth homology group of S, where $H_0(S) \simeq \mathbf{Z} \oplus \cdots \oplus \mathbf{Z}$ (m copies of $\mathbf{Z}$) if S has m path connected components.

(ii) $H_1(S)$ is the first homology group of S where $H_1(S) \simeq \{0\}$ if $H_1(S)$ is contractible to a point.

LEMMA 2.5. (Mayer-Vietoris). *Let A and B be two closed sets. Then the sequence*

$$H_1(A \cup B) \xrightarrow{h_1} H_0(A \cap B) \xrightarrow{h_2} H_0(A) \oplus H_0(B) \xrightarrow{h_3} H_0(A \cup B) \xrightarrow{h_4} \{0\}$$

is an exact sequence.

Proof. See Massey [8]. □

We will not be interested in the particular h_i's of Lemma 2.5 except for the fact that they form an exact sequence. Next we will use Lemma 2.5 with A of the lemma replaced with $H_D - P_D(i)$ and B of the lemma replaced with $K_D(i)$. Notice that $H_D - P_D(i) \cap K_D(i) = K_D(i) - P_D(i)$ which is the boundary of the face $K_D(i)$. The following result shows that the number of path connected components of the boundary of a face of CONNECTED is equal to the number of path connected components of the face of CONNECTED.

LEMMA 2.6. $H_0(K_D(i) - P_D(i)) \simeq H_0(K_D(i))$ *when* rank $(M_D) \leqq 2n-2$.

Proof. $H_D - P_D(i)$ and $K_D(i)$ are closed sets in H_D by Lemma 2.2 and the definition of $K_D(i)$. $H_D = (H_D - P_D(i)) \cup K_D(i)$ is clearly contractible to a point and path connected and so $H_0(H_D) \simeq \{0\}$. By Lemma 2.4, $H_D - P_D(i)$ is path connected and so $H_0(H_D - P_D(i)) \simeq \mathbf{Z}$. Thus taking $A = H_D - P_D(i)$ and $B = K_D(i)$ in Lemma 2.5, we get the situation shown in Fig. 9. Then since we have an exact sequence, we conclude that h_2 is $1-1$ and h_3 is onto. Thus $\mathbf{Z} \oplus H_0(K_D(i)) \simeq \mathbf{Z} \oplus \text{Im}\,(h_2) \simeq \mathbf{Z} \oplus H_0((H_D - P_D(i)) \cap K_D(i))$. In other words, we have that $H_0((H_D - P_D(i)) \cap K_D(i)) = H_0(K_D(i) - P_D(i)) \simeq H_0(K_D(i))$. By the definition of the zeroth homology group we conclude that $K_D(i)$ has the same number of path connected components as $K_D(i) - P_D(i)$. □

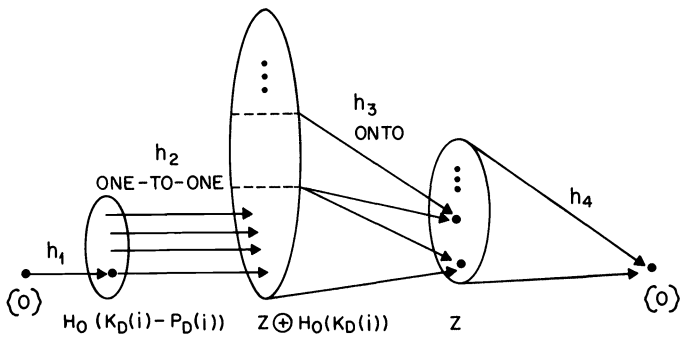


FIG. 9. *Mayer-Vietoris sequence.*

We have now shown that a face of CONNECTED and its boundary have the same number of path connected components. However, we further need the result that each path connected component of a face of CONNECTED contains exactly one path connected component of its boundary. In order to prove this, the following two lemmas are needed.

LEMMA 2.7. *Each path connected component of $K_D(i) - P_D(i)$ intersects exactly one path connected component of $K_D(i)$.*

Proof. This follows from the fact that $K_D(i) - P_D(i) \subseteq K_D(i)$. □

Next it is shown that each path connected component of a face of CONNECTED contains a path connected component of its boundary provided that the objects do not form a vertex object.

LEMMA 2.8. *Each path connected component of* $K_D(i)$ *contains at least one path connected component of* $K_D(i) - P_D(i)$ if $\text{rank}(M_D) \leqq 2n-1$.

Proof. It is sufficient to show that for any configuration v_1 in $K_D(i)$ there is a configuration v_2 in $K_D(i) - P_D(i)$ and a path in $K_D(i)$ from v_1 to v_2.

Thus the result is trivial if $v_1 \in K_D(i) - P_D(i)$. Otherwise $v_1 \in P_D(i)$. Since $\text{rank}(M_{v_1}) = \text{rank}(M_D) \leqq 2n-1$ there is a vertex object B_1 of v_1 with a face b_1 that touches a face b_2 of another vertex object B_2 of v_1. Moving B_1 in $P_D(i)$ so that b_1 moves across b_2 until some face of B_1 touches some face that was not touching in v_1 results in a configuration v_2 where $v_2 \notin E_D$ and so $v_2 \notin P_D(i)$. Clearly $v_2 \in K_D(i)$ by definition of v_2. Thus the lemma follows. □

The preceding results can be combined to conclude that there is a one-to-one correspondence between the path connected components of a face of CONNECTED and those of its boundary. This is stated in the following lemma.

LEMMA 2.9. *Each path connected component of* $K_D(i)$ *contains exactly one path connected component of* $K_D(i) - P_D(i)$.

Proof. The lemma follows from Lemma 2.6 and Lemmas 2.7 and 2.8 by a simple counting argument. □

As a consequence of Lemma 2.9 we can now conclude that a path in the boundary of a face exists between configurations in the boundary if there is a path in the face between the configurations.

LEMMA 2.10. *If* $\text{rank}(M_D) \leqq 2n-2$ *and there is a path in* $K_D(i)$ *between two configurations in* $K_D(i) - P_D(i)$, *then there is a path in* $K_D(i) - P_D(i)$ *between these configurations.*

Proof. Let $v_1, v_2 \in K_D(i) - P_D(i)$ and p be a path in $K_D(i)$ between v_1 and v_2. Thus, v_1 and v_2 are in the same path connected component of $K_D(i)$ and hence by Lemma 2.9, v_1 and v_2 are in the same path connected component of $K_D(i) - P_D(i)$. That is, there is a path in $K_D(i) - P_D(i)$ between v_1 and v_2. □

We will show that the boundary of a face $K_D(i)$ of CONNECTED consists of faces of CONNECTED of dimension less than that of the face $K_D(i)$. Thus using Lemma 2.10, we will conclude that if there is a path in a face of CONNECTED between two configurations in the boundary of the face, then there is a path between the two configurations contained in faces of CONNECTED of dimension less than that of the face $K_D(i)$. We begin by showing that a configuration in a face satisfies the description that defines the face.

LEMMA 2.11. *Let* v *be a configuration in* $K_D(i)$. *Then* v *satisfies* D *and so* $\text{rank}(M_v) \geqq \text{rank}(M_D)$.

Proof. Suppose v does not satisfy D. Then there must be two faces that touch according to D but do not touch in v. Let $\varepsilon > 0$ be the distance between the two closest such faces in v. Any configuration in H_D where each object has been moved less than $\varepsilon/2$ from its position in v also does not satisfy D. Thus there is a ball B in H_D about v such that $B \cap P_D(i) = \varnothing$. Since $K_D(i)$ is the closure of $P_D(i)$ in H_D, v is not in $K_D(i)$, a contradiction. Hence each v in $K_D(i)$ satisfies D and thus M_v has at least the rows of M_D. Therefore $\text{rank}(M_v) \geqq \text{rank}(M_D)$. □

We now show that a configuration v in the boundary of a face, $K_D(i)$ of CONNECTED lies in a face of CONNECTED with dimension less than that of the face $K_D(i)$.

LEMMA 2.12. *If* $v \in K_D(i) - P_D(i)$ *then* $\text{rank}(M_v) > \text{rank}(M_D)$.

Proof. Suppose $v \in P_D(j)$ where $j \neq i$. By Lemma 2.2, $P_D(j)$ is open in H_D and so there is an open ball B about v such that $B \subseteq P_D(j)$. However, v is in $K_D(i)$, the closure of $P_D(i)$ in H_D. This implies that $B \cap P_D(i) \neq \emptyset$. Thus $P_D(j) \cap P_D(i) \neq \emptyset$. This contradicts the assumption that $P_D(i)$ and $P_D(j)$ are distinct path connected components of E_D. Thus $v \notin E_D$. That is, v does not satisfy D exactly.

By Lemma 2.11, rank $(M_v) \geqq$ rank (M_D). Suppose rank $(M_v) =$ rank (M_D). Let D_1 be the description that v satisfies exactly (i.e. $M_{D_1} = M_v$). By Lemma 2.11 we have that v satisfies D and so all of the clauses of D are clauses of D_1. Therefore, $H_{D_1} \subseteq H_D$. By the above, we know that $D_1 \neq D$ and hence $H_{D_1} \neq H_D$. Thus $H_{D_1} \subset H_D$. But this implies that the dimension of H_{D_1} must be less than the dimension of H_D and so rank $(M_{D_1}) =$ rank $(M_v) >$ rank (M_D). □

It is now possible to prove that the boundary of a face of CONNECTED, $K_D(i) - P_D(i)$, is exactly the set of configurations in $K_D(i)$ that lie in faces of CONNECTED of dimension less than that of the face $K_D(i)$.

LEMMA 2.13. $K_D(i) - P_D(i) = K_D(i) \cap \{v | \text{rank } (M_v) > \text{rank } (M_D)\}$.

Proof. Let $v \in K_D(i) - P_D(i)$. By Lemma 2.12, rank $(M_v) >$ rank (M_D) and so $K_D(i) - P_D(i) \subseteq K_D(i) \cap \{v | \text{rank } (M_v) > \text{rank } (M_D)\}$.

Let $v \in K_D(i) \cap \{v | \text{rank } (M_v) > \text{rank } (M_D)\}$. Then $v \in K_D(i)$. Since rank $(M_v) >$ rank (M_D) we know that v does not satisfy D exactly (i.e. $v \notin E_D$) and hence v can not be in any path connected component of E_D. Thus $v \in K_D(i) - P_D(i)$ and so $K_D(i) \cap \{v | \text{rank } (M_v) > \text{rank } (M_D)\} \subseteq K_D(i) - P_D(i)$. Therefore, we can conclude that $K_D(i) - P_D(i) = K_D(i) \cap \{v | \text{rank } (M_v) > \text{rank } (M_D)\}$. □

Lemma 2.10 showed that if a path in the face of CONNECTED exists between two points in the boundary of the face then there is a path in the boundary between the points. Combining this result and Lemma 2.13 allows us to conclude that if there is a path in a face of CONNECTED of dimension d between two configurations in the boundary of the face then there is a path between them contained in faces of CONNECTED of dimension less than d.

3. Edge connectedness. In this section we will show that there is a path consisting of edges of CONNECTED between two vertices of CONNECTED whenever there is any path in NONOVERLAP between them. In order to further examine the structure of CONNECTED, we introduce the notion of a complex. A *d-complex*, C_d, is the union of all faces of CONNECTED of dimension d or less. The faces of CONNECTED in C_0 and C_1 are thus the vertices and edges of CONNECTED, respectively.

We can interpret the results of the previous section as follows. Let K_D be a face of CONNECTED such that $K_D \subseteq C_{d+1}$. Then we conclude that if there is a path in K_D between two points in the boundary of K_D (hence the points are in C_d) then there is a path in the boundary (hence a path in C_d) between them. Now we wish to extend this to show that a path in C_d exists between two configurations in C_d if there is a path in C_{d+1} between them, even if the path goes through more than one face of C_{d+1}. The proof proceeds by showing in Lemma 3.2 that configurations in the intersection of two faces of dimension d lie in faces of dimension $d-1$ or less. This result is used to show in Lemma 3.3 that any two configurations in C_d, $d > 0$, that are connected by a path in C_{d+1} are connected by a path in C_d. An inductive argument is then used in Theorem 3.6 to show that there exists a path consisting of vertices and edges. To begin the induction, we use a result from [5] to argue that a path in NONOVERLAP implies a path in CONNECTED and Lemma 3.5 that establishes that CONNECTED equals C_n.

We now proceed to establish these results. Lemma 3.1 is a technical lemma concerning the intersections of closed sets. The reader may wish to skip immediately to Lemma 3.2. In the following when we say p: $[0, 1] \to P$ is a path in set S we mean that $P \subseteq S$ and P is the range of p.

LEMMA 3.1. *Let A and B be closed sets and let p: $[0, 1] \to P$ be a path in $A \cup B$ from $x \in A$ to $y \in B$. Then $P \cap A \cap B \neq \emptyset$.*

Proof. Suppose $P \cap A \cap B = \emptyset$. Then $p^{-1}(P \cap A \cap B) = \emptyset$ and so $p^{-1}(P \cap A) \cap p^{-1}(P \cap B) = \emptyset$. Since P is in $A \cup B$, $p^{-1}(P \cap A) \cup p^{-1}(P \cap B) = [0, 1]$. Since $p^{-1}(P \cap A)$ and $p^{-1}(P \cap B)$ are closed, we conclude that $[0, 1]$ is not connected, a contradiction. Therefore, $P \cap A \cap B \neq \emptyset$. □

Suppose there is a path p in C_{2n-k} between two configurations v and w in C_{2n-k-1}. Let $K_{D_1}(i_1), K_{D_2}(i_2), \cdots, K_{D_t}(i_t)$ be the sequence of faces that the path p intersects between v and w. Let K_j denote $K_{D_j}(i_j)$. Thus $K_j \neq K_{j+1}$. By Lemma 3.1 for each j, $1 \leqq j \leqq t-1$ there must be a $t_j \in [0, 1]$ such that $p(t_j) \in K_j \cap K_{j+1}$. Let $p(t_j) = v_j$ and p_j be the section of p on K_j. Also let $v_0 = v$ and $v_t = w$. To be able to apply Lemma 2.10 to p_j, we must show that v_{j-1} and v_j are in $K_j - P_j$ where P_j denotes $P_{D_j}(i_j)$. By Lemma 2.13 it is sufficient to show that rank $(M_{v_j}) > k$ and rank $(M_{v_{j-1}}) > k$. Lemma 3.2 will establish this fact.

LEMMA 3.2. *Let the dimensions of K_j and K_{j+1} be at most $2n-k$ (i.e.* rank $(M_{D_i}) \geqq k$ *for $i = j, j+1$). That is K_j and K_{j+1} are contained in C_{2n-k}. Let v be in their intersection. Then* rank $(M_v) > k$. *In other words $v \in C_{2n-k-1}$.*

Proof. Suppose $D_j = D_{j+1}$. Since $K_j \neq K_{j+1}$ it must be that $P_j \neq P_{j+1}$ where K_i is the closure of P_i in H_{D_i} for $i = j, j+1$. Thus P_j and P_{j+1} are two distinct path connected components of E_{D_j} and so $P_j \cap P_{j+1} = \emptyset$. In particular, $v \notin P_j \cap P_{j+1}$ and so $v \in K_j - P_j$ or $v \in K_{j+1} - P_{j+1}$. In either case, by Lemma 2.12, rank $(M_v) > k$.

Suppose $D_j \neq D_{j+1}$. Then v cannot satisfy both D_j and D_{j+1} exactly and so $v \notin P_j \cap P_{j+1}$ and hence $v \in K_j - P_j$ or $v \in K_{j+1} - P_{j+1}$. Then, again by Lemma 2.12, rank $(M_v) > k$. □

We will show that there is a path in C_{2n-k-1} between v and w. If the dimension of K_j is less than $2n-k$ then p_j is contained in C_{2n-k-1}. Suppose the dimension of K_j is $2n-k$ (i.e., rank $(M_{D_j}) = k$). By Lemma 3.2, rank $(M_{v_{j-1}}) > k$ and rank $(M_{v_j}) > k$ and so $v_j, v_{j+1} \in K_j - P_j$ by Lemma 2.13. Thus p_j is a path in K_j between two configurations in $K_j - P_j$ and so by Lemma 2.10 there is a path in $K_j - P_j$ between them. By Lemma 2.13 we conclude that there is a path in C_{2n-k-1} between v_{j-1} and v_j in this case. Thus there is a path in C_{2n-k-1} between each v_{j-1} and v_j and so there is a path from v to w in C_{2n-k-1}.

LEMMA 3.3. *If there is a path in C_{2n-k} from v to w where $v, w \in C_{2n-k-1}$ then there is a path in C_{2n-k-1} from v to w where $k \leqq 2n-2$.*

Proof. See preceding discussion. □

We are going to want to apply Lemma 3.3 inductively and so we will first show two results about CONNECTED to provide a starting point for the induction.

LEMMA 3.4. *If v is a connected configuration of $n+1$ objects, then* rank $(M_v) \geqq n$.

Proof. The proof is by induction on the number of objects. Suppose v is a connected configuration of 2 objects. Then there is at least one face of the moveable object that touches a face of the fixed object. Thus M_v has at least one nonzero row and so rank $(M_v) \geqq 1$.

Assume the result holds for configurations of n objects. Let v be a connected configuration of $n+1$ objects. As before, let G_v be the graph with a node for each object and an edge between nodes if the corresponding objects touch in v. Let T be a depth first spanning tree of G_v with the node corresponding to A_0, the fixed object,

as the root. Then the leaves of T are not articulation nodes of G_v (see [1]). That is, removing an object corresponding to a leaf of T results in a connected configuration w of n objects. By the induction hypothesis, rank $(M_w) \geqq n-1$. Without loss of generality assume that the object removed was A_n. Then

$$M_v = \begin{bmatrix} M_w & 0 \\ * & B \end{bmatrix}$$

where B contains nonzeros because A_n touches at least one of the other objects. Thus rank $(M_v) \geqq$ rank $(M_w)+1 \geqq n$ as required. □

LEMMA 3.5. CONNECTED $= C_n$.

Proof. By Lemma 3.4, $v \in$ CONNECTED implies rank $(M_v) \geqq n$. But rank $(M_v) \geqq n$ means $v \in C_p$ for some $p \leqq n$. Since $p \leqq n$ implies $C_p \subseteq C_n$ we conclude that $v \in C_n$.

Let $v \in C_n$. Then $v \in K_C(i)$ for some D and i with rank $(M_D) \geqq n$. Let $P_D(i)$ be the path connected component of E_D such that $cl_{H_D}(P_d(i)) = K_D(i)$. By definition of E_D, $P_D(i) \subseteq$ CONNECTED $\cap H_D$ which implies that $K_D(i) \subseteq Cl_{H_D}($CONNECTED $\cap H_D) =$ CONNECTED $\cap H_D$. Therefore $K_D(i) \subseteq$ CONNECTED and so $C_n \subseteq$ CONNECTED. Hence CONNECTED $= C_n$. □

We are now in a position to prove our main goal. That is, we will show that if there is a path in NONOVERLAP between two vertices of CONNECTED then there is a path contained in the edges of CONNECTED between them.

THEOREM 3.6. *Let v and w be vertices of* CONNECTED. *If there is a path in* NONOVERLAP *between v and w then there is a path contained in the edges of* CONNECTED *between v and w.*

Proof. By [5] we know that there is a path in CONNECTED from v to w. Thus by Lemma 3.5 there is a path in C_n from v to w. Applying Lemma 3.3 inductively we conclude that there is a path in C_1 from v to w. In other words, there is a path along edges from v to w. □

4. A PSPACE-complete motion problem. In this section we will show that the problem of determining whether a motion exists between two configurations of two-dimensional rectangles within a rectangular enclosure with integer sizes where the fixed object is at an integer location is in PSPACE. To accomplish this, a nondeterministic method will be described and will be shown to require polynomial space. Since NSPACE equals PSPACE, we will conclude that the problem is in PSPACE.

First it must be shown how to get from an arbitrary configuration to a vertex configuration. Move the object which is closest to the fixed object in the x or y direction until it touches the fixed object. Considering these two objects as one fixed object, repeat this until a configuration in CONNECTED results. Now the motion in Lemma 2.8 can be repeated until a vertex configuration is encountered.

It is now sufficient to show how to determine if there is a motion between vertex configurations. From a vertex configuration, nondeterministically guess a subset of objects and face of an object to move this subset along. Check to see if this is a motion along an edge of CONNECTED. If so, move the subset of objects until a new pair of faces touch. By the previous results, this new configuration must be a vertex configuration. Continue this process until the desired vertex configuration is encountered.

Since the position of the objects in a vertex configuration is the solution to an integer linear system of equations, the positions of the objects in a vertex configuration can be stored in polynomial space (see [2]). Clearly, finding the next faces that touch when moving objects in the direction of one of the coordinates axes can be done in

polynomial space. Calculating the rank of the matrix M_D where D is the description resulting from moving the subset of objects to see if the motion is along an edge of CONNECTED also can be done in polynomial space. Thus the problem is in NSPACE and so in PSPACE.

In [4] it was shown that the problem of deciding whether there is a motion between two configurations of two-dimensional rectangles in a two-dimensional box is PSPACE-hard. Hence the problem is PSPACE-complete.

5. Conclusions. Using the concept that determining if a coordinated motion of multiple objects exists is equivalent to searching for a path in a high dimensional space (configuration space), it has been shown that for very general two-, three-dimensional objects, the search in configuration space can be restricted to CONNECTED, the configuration space body whose points correspond to configurations in which all the objects form one connected piece [5]. Making use of the above result, it was shown in this paper that for two-dimensional objects with sides parallel to the coordinate axes, the search for such a path in configuration space can be reduced to searching for a path in a graph when translations are the only motions allowed. The path in the graph corresponds to a path in configuration space that consists of zero and one-dimensional faces of CONNECTED.

In the case of two-dimensional objects with linear sides that are not necessarily parallel to the coordinate axes of the plane and where only translations are permitted, similar results to those developed in this paper could be obtained.

With more general objects, analyzing the structure of CONNECTED is much more difficult because the faces will not necessarily be portions of hyperplanes and so the intersection of faces will be more complicated.

All of the results of this paper up to Lemma 2.3 remain valid for three-dimensional objects with planar faces parallel to the planes of the coordinate axis of $\mathbf{R}^3$. Basically Lemma 2.3 says that the faces of CONNECTED of dimension two or more correspond to cases where there are at least three vertex objects. This property is needed in the construction in the proof of Lemma 2.4. In the case of the three-dimensional objects it can only be shown that the faces of dimension three or more correspond to cases where there are at least three vertex objects. Thus we can conclude only that faces of dimension three or more have the same number of path connected components as the boundary of the face in the case of three-dimensional objects. Therefore, in this case, the search for a motion can be reduced to searching for a path within the faces of CONNECTED of dimension two or less. It appears that the search for a path in configuration space could be reduced even further to searching for a path in the vertices and edges of CONNECTED even for three-dimensional objects. Unfortunately there are faces of CONNECTED in this case, such that the vertices of the face are not connected by edges of the face. Thus, the method of proof used in this paper will not suffice to prove the desired result because the analysis consisted of studying one face of CONNECTED at a time and although vertices on a face may not be connected by edges of that face, there may be a path consisting of edges and vertices of other faces that join the two points. Thus a more global study of the structure of CONNECTED would have to be performed to obtain this result rather than the restricting one's view to one face at a time.

The structure of configuration space has an additional complexity when rotations are allowed. There are two choices of how to define configuration space. One can identify configurations in which an object is rotated in a complete circle or one can think of these two as distinct configurations. In the first case configuration space

is no longer merely a high dimensional Euclidean space but some sort of quotient space.

Understanding the structure of the various sets in configuration space should be a valuable aid in deciding if a motion exists between two configurations. In the cases where searching for a path among the vertices and edges of CONNECTED is sufficient, this special structure of CONNECTED will help in the actual planning of a motion because we know that only motions that correspond to paths along edges have to be considered. Thus in the case of a small number of objects moving about in a relatively open workspace, the graph could be small enough to search for a path. In any case, it at least provides a tool that a heuristic could make use of. Even in the cases indicated above where searching a graph is not sufficient, a more complete knowledge of the structure of configuration space should be useful in developing heuristics for motion planning.

Acknowledgment. The authors would like to thank the referees for their valuable comments.

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] I. BOROSH AND L. B. TREYBIG, *Bounds on positive integral solutions of linear diophantine equations*, Proc. Amer. Math. Soc., 55 (1976), pp. 299-304.

[3] J. E. HOPCROFT, D. JOSEPH AND S. WHITESIDES, *Movement problems for 2-dimensional linkages*, this Journal, 13 (1984), pp. 610-629.

[4] J. E. HOPCROFT, J. T. SCHWARTZ AND M. SHARIR, *On the complexity of motion planning for multiple independent objects*; PSPACE *hardness of the "warehouseman's problem"*, Robotics Research Technical Report No. 14, Computer Science Division, New York Univ., New York, February 1984.

[5] J. E. HOPCROFT AND G. T. WILFONG, *On the motion of objects in contact*, Proc. 2nd International Symposium on Robotics Research, Kyoto, Japan, 1984, pp. 81-90.

[6] T. LOZANO-PEREZ AND M. A. WESLEY, *An algorithm for planning collision-free paths among polyhedral obstacles*, Comm. ACM, 22 (1979), pp. 560-570.

[7] T. LOZANO-PEREZ, *Automatic planning of manipulator transfer movements*, IEEE Trans. Syst., Man, Cybern., SMC-11 (1981), pp. 681-698.

[8] W. S. MASSEY, *Homology and Cohomology Theory*, Marcel Dekker, New York, 1978.

[9] J. REIF, *Complexity of the mover's problem and generalizations*, in Proc. 20th IEEE Foundations of Computer Science Conference, Institute of Electrical and Electronics Engineers, New York, 1979, pp. 421-427.

[10] J. T. SCHWARTZ AND M. SHARIR, *On the piano mover's problem* I. *The case of a two-dimensional rigid polynomial body moving amidst polygonal barriers*, Comm. Pure Appl. Math., 36 (1983), pp. 345-398.

[11] ———, *On the piano mover's problem* II. *General techniques for computing topological properties of real algebraic manifolds*, Adv. Appl. Math., 4 (1983), pp. 298-351.

[12] ———, *On the piano mover's problem* III. *Coordinating the motion of several independent bodies. The special case of circular bodies moving amidst polygonal barriers*, Intern. J. Robotics Research, 2, 3, Fall 1983.

[13] M. SHARIR AND E. ARIEL-SHEFFI, *On the piano mover's problem* IV. *Various decomposable two-dimensional motion planning problems*, Comm. Pure Appl. Math., 37 (1984), pp. 479-493.

[14] S. WILLARD, *General Topology*, Addison-Wesley, Reading, MA, 1970.

SIAM J.COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
011

CHURCH-ROSSER THUE SYSTEMS THAT PRESENT FREE MONOIDS*

FRIEDRICH OTTO†

Abstract. It is undecidable in general whether the monoid presented by a given Thue system is a free monoid. Here it is shown that this question is decidable for Church-Rosser Thue systems.

Key words. Thue congruence, Church-Rosser Thue system, monoid presentation, free monoid, Markov property, Tietze transformation

AMS(MOS) subject classifications. 03D03, 20M05

1. Introduction. Let T be a Thue system on Σ, where Σ is a finite alphabet. Then the ordered pair $(\Sigma; T)$ is called a (*monoid*) *presentation*, and the monoid M it presents is the quotient of the free monoid Σ^* by the smallest congruence including T [4]. How much information about the algebraic structure of M can be deduced from a given presentation of M? In general, it is impossible to derive much information, since even the following problem is undecidable in general [7]:

Instance. A finite (monoid) presentation $(\Sigma; T)$.
Question. Is the monoid M presented by $(\Sigma; T)$ trivial, i.e., does M consist only of the identity?

Many decision problems can be formulated for monoids given by finite presentations, e.g., the word problem, the power problem, and the conjugacy problem. Although being undecidable in general, all these problems are decidable for monoids presented by certain restricted classes of presentations, e.g., the word problem and the conjugacy problem are decidable for monoids presented by finite Church-Rosser Thue systems [1], [10], and the power problem is decidable for monoids presented by finite monadic Church-Rosser Thue systems [2]. In fact, in [2] Book develops a technique for solving a whole class of decision problems for monoids presented by finite monadic Church-Rosser Thue systems.

In this note we are dealing with the following problem:

Instance. A finite (monoid) presentation $(\Sigma; T)$.
Question. Is the monoid M presented by $(\Sigma; T)$ a free monoid?

Since the property of being free is a *Markov property* for finitely presented monoids [7], this problem is also undecidable in general. Does it become decidable when it is restricted to monoids presented by finite Church-Rosser Thue systems?

It seems that the property of being free is not expressible by a "linear sentence" as defined in [2]. Hence, the results of [2] cannot be used here. Furthermore, since they only hold for monadic Church-Rosser Thue systems [12], they would not meet our demand anyway. So, we have to find a different way to solve the above problem for monoids presented by finite Church-Rosser Thue systems.

After giving some basic definitions and notation in § 2, we prove the following as a first step in § 3. Assume that the monoid M presented by $(\Sigma; T)$ is free. Then M is freely generated by a subset Σ_0 of Σ. Hence, M is also presented by $(\Sigma_0; \emptyset)$. This

* Received by the editors September 4, 1984, and in revised form May 1, 1985.
† Fachbereich Informatik, Universität Kaiserslautern, Postfach 3049, 6750 Kaiserslautern, West Germany.

implies that there is a finite sequence of Tietze transformations [6] that transforms the presentation $(\Sigma; T)$ into $(\Sigma_0; \varnothing)$. In fact, this is true for every two finite presentations of the same monoid, the problem being how to find such a finite sequence of Tietze transformations.

If the Thue system T is reduced and Church-Rosser, then we can derive some information about the form of the rules of T. This is done in § 4. Since for every finite Church-Rosser Thue system T, there is a unique reduced Church-Rosser Thue system T' that is equivalent to T, and since T' is computable from T in polynomial time using linear space [9], we lose no generality by restricting our attention to reduced Church-Rosser Thue systems. Now this additional information on the form of the rules of T is sufficient to effectively construct a finite sequence of Tietze transformations from $(\Sigma; T)$ to $(\Sigma_0; \varnothing)$, if such a sequence exists. In § 5 an algorithm is given that, based on this observation, on input a presentation $(\Sigma; T)$ where T is a finite Church-Rosser Thue system, decides whether the monoid defined by this presentation is free or not. The correctness of this algorithm is proved from the results of the previous sections.

So the problem of deciding freeness of monoids gives another example of a decision problem, that, although being undecidable in general, becomes decidable when being restricted to monoids presented by finite Church-Rosser Thue systems. Hence, it again underlines the usefulness of the Church-Rosser property for solving decision problems for monoids.

2. Thue systems. Let Σ be a finite alphabet and Σ^* the set of all words over Σ including the empty word e. For $w \in \Sigma^*$, the *length* of w is denoted by $|w|: |e| = 0$, and $|wa| = |w| + 1$ for all $w \in \Sigma^*$, $a \in \Sigma$.

A *Thue system* T on Σ is a subset of $\Sigma^* \times \Sigma^*$. The elements of T are called *rules*. Given a Thue system T on Σ, $\text{domain}(T) = \{l | \exists r \in \Sigma^*: (l, r) \in T\}$, and $\text{range}(T) = \{r | \exists l \in \Sigma^*: (l, r) \in T\}$.

For a Thue system T on Σ, let $\leftrightarrow_T$ be the following relation: if (l, r) is a rule of T, then for all $x, y \in \Sigma^*$,

$$xly \leftrightarrow_T xry \quad \text{and} \quad xry \leftrightarrow_T xly.$$

The reflexive and transitive closure $\leftrightarrow_T^*$ of $\leftrightarrow_T$ is a congruence on Σ^*, the *Thue congruence* generated by T. If $u \leftrightarrow_T^* v$ one says that u and v are *congruent* (*modulo* T). The *congruence class* $[u]_T$ of u is the set

$$\{v \in \Sigma^* | v \leftrightarrow_T^* u\}.$$

Since the relation $\leftrightarrow_T$ is symmetric, we can assume without loss of generality that no rule of T is length-increasing, i.e., if $(l, r) \in T$ then $|l| \geqq |r|$.

Two Thue systems T_1 and T_2 are *equivalent*, if they define the same Thue congruence, i.e., they are both on the same alphabet Σ, and

$$\leftrightarrow_{T_1}^* = \leftrightarrow_{T_2}^*.$$

PROPOSITION 2.1. [4]. *Let T be a Thue system on Σ. Then the set of congruence classes $\{[u]_T | u \in \Sigma^*\}$ forms a monoid under the operation $[u]_T \circ [v]_T = [uv]_T$ with identity $[e]_T$. This monoid is denoted as $\Sigma^*/\leftrightarrow_T^*$.*

Let M be a monoid. If $M \cong \Sigma^*/\leftrightarrow_T^*$, i.e., if the monoids M and $\Sigma^*/\leftrightarrow_T^*$ are isomorphic, then the ordered pair $(\Sigma; T)$ is called a *presentation* of M with Σ being the set of generators, and T being the set of *defining relations* of this presentation. M is called *finitely presented*, if there exists a finite presentation of M, i.e., if there exists a presentation $(\Sigma; T)$ of M with Σ and T being finite.

A monoid M is *free*, if it has a presentation of the form $(\Sigma; \emptyset)$, i.e., if it has a presentation with an empty set of defining relations. Now, two words $u, v \in \Sigma^*$ are congruent modulo $\emptyset$ if and only if they are equal. Hence,

$$\Sigma^*/\leftrightarrow_\emptyset^* \cong \Sigma^*.$$

Let T be a Thue system on Σ. For $u, v \in \Sigma^*$, if $u \leftrightarrow_T v$ and $|u| > |v|$, then define $u \to_T v$. The reflexive and transitive closure $\to_T^*$ of $\to_T$ is the *reduction* relation defined by T. Since words cannot have negative length, the relation $\to_T$ is Noetherian, i.e., there exists no infinite chain $u_1 \to_T u_2 \to_T u_3 \to_T \cdots$. If $u \to_T^* v$, one says that u *reduces* to v, u is an *ancestor* of v, and v is a *descendant* of u (modulo T). A word u is *irreducible* if it has no descendant except itself, otherwise it is *reducible* (modulo T). Obviously, each word has at least one irreducible descendant (modulo T).

Following the notation of Book [1], we call a Thue system T *Church-Rosser* if every two congruent words have a common descendant. In other words, for every choice of u and v, $u \leftrightarrow_T^* v$ implies that for some z, $u \to_T^* z$ and $v \to_T^* z$. Equivalently, a Thue system is Church-Rosser if every congruence class contains a unique irreducible word, which can then be considered as a representative for that class.

Obviously, a Church-Rosser Thue system T is equivalent to the subsystem T' consisting of all the length-reducing rules of T. Hence, we may assume without loss of generality that a Church-Rosser Thue system T contains length-reducing rules only, i.e., $|l| > |r|$ for each rule $(l, r) \in T$. So the class of Church-Rosser Thue systems is a very restricted one. On the other hand, this restriction has been justified by many interesting results, e.g., the word problem for a finite Church-Rosser Thue system is decidable in linear time [1], it can be checked in polynomial time and linear space whether a given finite Thue system is Church-Rosser [3], and the conjugacy problem for a finite Church-Rosser Thue system is decidable [10], while the conjugacy problem is undecidable in general even for finite Thue systems that define unique representatives without being length-reducing [11].

3. Thue systems presenting free monoids. A property P of monoids is called *invariant* if every monoid that is isomorphic to a monoid possessing property P itself possesses this property. An invariant property of finitely presented monoids is a *Markov property* [7], [8], if it satisfies the following conditions:

(1) There is a finitely presented monoid M_1 which does not have property P, and which is not isomorphic to a submonoid of any finitely presented monoid having property P, and

(2) there exists a finitely presented monoid M_2 having property P.

Obviously, the property of being free is an invariant property of finitely presented monoids, and it satisfies (2). On the other hand, each finitely generated submonoid of a free monoid has a decidable word problem. Thus, a finitely presented monoid with an undecidable word problem is neither free nor isomorphic to a submonoid of a free monoid, i.e., condition (1) is also satisfied. So the property of being free is a Markov property. Hence, by the main result of [7], the following problem is undecidable in general:

Instance. A finite presentation $(\Sigma; T)$.
Question. Is the monoid $\Sigma^*/\leftrightarrow_T^*$ a free monoid?

Although this problem is undecidable, we do at least have the following information on presentations of free monoids.

THEOREM 3.1. *If the monoid $\Sigma^*/\leftrightarrow_T^*$ presented by $(\Sigma; T)$ is free, then there exists a subset Σ_0 of Σ that freely generates this monoid.*

Proof. Let $\Sigma = \{a_1, a_2, \cdots, a_n\}$, and assume that the monoid $M = \Sigma^*/\leftrightarrow_T^*$ is free of rank m. Then $\Sigma^*/\leftrightarrow_T^* \cong \Gamma^*$ for some alphabet $\Gamma = \{b_1, b_2, \cdots, b_m\}$. Hence, for each $a_i \in \Sigma$, there exists a word $u_i \in \Gamma^*$ such that a_i and u_i represent the same element of M. Analogously, for each $b_j \in \Gamma$, there exists a word $v_j \in \Sigma^*$ such that b_j and v_j represent the same element of M. Since $M \cong \Gamma^*$, no $b_j \in \Gamma$ represents the identity of M, and so $v_j \neq e, j = 1, \cdots, m$. Further the words $v_j \in \Sigma^*$ can be chosen in such a way that no v_j contains an occurrence of a letter $a_i \in \Sigma$ with $a_i \leftrightarrow_T^* e$. On the other hand, $u_i = e$ if and only if $a_i \leftrightarrow_T^* e$, $i = 1, \cdots, n$.

Let $b_j \in \Gamma$, and assume that $v_j = a_{i_1} a_{i_2} \cdots a_{i_k}$. For each λ, $1 \leqq \lambda \leqq k$, u_{i_λ} represents the same element of M as a_{i_λ}. Hence, b_j and the word $u_{i_1} u_{i_2} \cdots u_{i_k} \in \Gamma^*$ represent the same element of M. Since M is free on Γ, this implies that $b_j = u_{i_1} u_{i_2} \cdots u_{i_k}$. By the choice of v_j, we have $u_{i_\lambda} \neq e$ for all λ implying that $k = 1$, i.e., $v_j \in \Sigma$.

Take $\Sigma_0 = \{a \in \Sigma | \exists b_j \in \Gamma: a = v_j\}$. Then $\Sigma_0 = \{v_j | j = 1, \cdots, m\}$ is a subset of Σ that freely generates M. □

Let T be a Thue system on Σ such that the monoid $M = \Sigma^*/\leftrightarrow_T^*$ is free, and let Σ_0 be a subset of Σ that freely generates M. Then, for each $a \in \Sigma$, there is a unique word $u_a \in \Sigma_0^*$ with $a \leftrightarrow_T^* u_a$. Define a homomorphism $\varphi: \Sigma^* \to \Sigma_0^*$ by taking $\varphi(a) := u_a$ for all $a \in \Sigma$. Then we have the following.

LEMMA 3.2. *For all $u, v \in \Sigma^*$, $u \leftrightarrow_T^* v$ if and only if $\varphi(u) = \varphi(v)$.*

Proof. Since $\varphi(w) \leftrightarrow_T^* w$ for all $w \in \Sigma^*$, $u \leftrightarrow_T^* v$ if and only if $\varphi(u) \leftrightarrow_T^* \varphi(v)$. But $\varphi(u), \varphi(v) \in \Sigma_0^*$, and Σ_0 freely generates M implying that $\varphi(u) \leftrightarrow_T^* \varphi(v)$ if and only if $\varphi(u) = \varphi(v)$. □

4. Church–Rosser Thue systems presenting free monoids. Many decision problems for monoids, that are undecidable in general, are decidable for monoids presented by finite Church-Rosser Thue systems, e.g., the word problem [1], and the conjugacy problem [10]. Does the problem of deciding whether the monoid defined by a given presentation $(\Sigma; T)$ is free also become decidable, when it is restricted to presentations involving finite Church-Rosser Thue systems? We will show that this is indeed the case. To this end some properties of Church-Rosser Thue systems presenting free monoids are derived in this section. These properties will be used in the next section to prove the correctness of our decision procedure for the problem stated above.

A Thue system $T = \{(l_i, r_i) | i \in I\}$ is called *reduced* if, for each $i \in I$, the word r_i is irreducible modulo T and the word l_i is irreducible modulo $T - \{(l_i, r_i)\}$. For each finite Church-Rosser Thue system T, there exists a reduced Church-Rosser Thue system T' that is equivalent to T. In fact, T' is uniquely determined by T, and T' is computable from T in polynomial time using linear space [9]. Hence, we can restrict our attention in the following to reduced Church-Rosser Thue systems.

So, for the remainder of this section let T be a fixed reduced Church-Rosser Thue system on Σ such that the monoid $M = \Sigma^*/\leftrightarrow_T^*$ is free. By Theorem 3.1 there exists a subset Σ_0 of Σ that freely generates M. Let $\varphi: \Sigma^* \to \Sigma_0^*$ be the corresponding homomorphism, and let $\Sigma_1 = \{a \in \Sigma | \varphi(a) = e\}$. Obviously, we have $\Sigma_0 \cap \Sigma_1 = \emptyset$.

LEMMA 4.1. *$T \cap (\Sigma^* \times \{e\}) = \Sigma_1 \times \{e\}$, i.e., the only rules with right-hand side e that T contains are the rules $\{(a, e) | \varphi(a) = e\}$.*

Proof. Σ_0 freely generates M, and $\varphi(w) \leftrightarrow_T^* w$ for all $w \in \Sigma^*$, implying that $[w]_T \cap \Sigma_0^* = \{\varphi(w)\}$ for all $w \in \Sigma^*$. Thus, $\varphi(w) = e$ if and only if $w \leftrightarrow_T^* e$, and so $\Sigma_1 = \{a \in \Sigma | a \leftrightarrow_T^* e\}$. Now T being Church-Rosser implies that $\Sigma_1 = \{a \in \Sigma | a \to_T e\}$, and therefore, $\{(a, e) | \varphi(a) = e\} = \{(a, e) | a \in \Sigma_1\} \subseteq T$.

It remains to show that T does not contain rules of the form (u, e) with $|u| \geqq 2$. Assume that $(u, e) \in T$ for some u with $|u| \geqq 2$. Then $u \leftrightarrow_T^* e$, and hence $\varphi(u) = \varphi(e) = e$ by Lemma 3.2, implying that $u \in \Sigma_1^*$ since after all $\varphi: \Sigma^* \to \Sigma_0^*$ is a homomorphism.

But this contradicts the fact that T is reduced. Thus, $T \cap (\Sigma^* \times \{e\}) = \Sigma_1 \times \{e\}$, as claimed. □

Σ_0 and Σ_1 are disjoint subsets of Σ. Let Σ_2 denote the remaining letters of Σ, i.e., $\Sigma_2 := \Sigma - (\Sigma_0 \cup \Sigma_1)$.

LEMMA 4.2. $\Sigma_2 = \text{range}(T) \cap \Sigma$, *i.e., for each* $a \in \Sigma_2$, *there exists a word* $u \in \Sigma^*$ *such that* $(u, a) \in T$, *and these are the only rules of* T *with right-hand sides of length* 1.

Proof. Let $a \in \Sigma_2$. Then $a \notin \Sigma_1$ implying that $a \not\to_T^* e$, and hence, a is irreducible modulo T.

Now $\varphi(a) \in \Sigma_0^*$ with $\varphi(a) \leftrightarrow_T^* a$. Since T is Church-Rosser, we have $\varphi(a) \to_T^* a$. In particular, this shows that $|\varphi(a)| \geqq 2$. The Thue system T is reduced. Hence, the Thue system $T_1 := T - \{(a, e) | a \in \Sigma_1\}$ is contained in $(\Sigma_0 \cup \Sigma_2)^* \times (\Sigma_0 \cup \Sigma_2)^*$, i.e., no rule of T_1 contains an occurrence of a letter from Σ_1. The words $\varphi(a)$ and a are in $(\Sigma_0 \cup \Sigma_2)^*$ with $\varphi(a) \to_T^* a$, and so we actually have $\varphi(a) \to_{T_1}^* a$, since during the reduction $\varphi(a) \to_T^* a$ no letter from Σ_1 can be introduced. By Lemma 4.1, the Thue system T_1 contains no rule with right-hand side e. Therefore, it must contain a rule with right-hand side a, implying that $\Sigma_2 \subseteq \text{range}(T)$.

As already mentioned, no rule of T_1 contains an occurrence of a letter from Σ_1. Thus, $\Sigma_1 \cap \text{range}(T) = \emptyset$. Now assume that $(u, a) \in T_1$ for some $a \in \Sigma_0$. Then $u \in (\Sigma_0 \cup \Sigma_2)^*$ with $|u| \geqq 2$. But $|\varphi(b)| \geqq 1$ for all $b \in \Sigma_0 \cup \Sigma_2$, which gives $|\varphi(u)| \geqq |u| \geqq 2$. On the other hand, $\varphi(u) = \varphi(a)$ by Lemma 3.2, and $\varphi(a) = a$ since $a \in \Sigma_0$, implying $1 = |\varphi(a)| = |\varphi(u)| \geqq 2$, a contradiction. This proves that $\Sigma_0 \cap \text{range}(T) = \emptyset$, and so $\text{range}(T) \cap \Sigma = \Sigma_2$. □

Since $\varphi(u) = \varphi(v)$ if $u \leftrightarrow_T^* v$, and since $|\varphi(b)| > 0$ for all $b \in \Sigma_0 \cup \Sigma_2$, we have the following as an easy consequence of Lemma 3.2.

COROLLARY 4.3. *For each* $a \in \Sigma_2$, *if* $u \in (\Sigma_0 \cup \Sigma_2)^*$ *with* $u \leftrightarrow_T^* a$, *then either* $u = a$ *or* $|u|_a = 0$.

Here, $|u|_a$ denotes the *a-length* of u, i.e., the number of occurrences of the letter a in u.

5. The algorithm. We are now going to present an Algorithm (A), that decides the following problem:

Instance: A presentation $(\Sigma; T)$, where T is a finite Church-Rosser Thue system.
Question: Is the monoid M presented by $(\Sigma; T)$ free?

After stating the algorithm we will prove its correctness. For doing so we need the results of the previous two sections.

Algorithm (A) is defined as follows:

(A) **begin input** a finite alphabet Σ, and a finite Church-Rosser Thue system T on Σ;
(1) **compute** the reduced Church-Rosser Thue system that is equivalent to T, and call it T;
(2) $\Sigma_1 := \{a \in \Sigma | (a, e) \in T\}$;
(3) $\Sigma := \Sigma - \Sigma_1$;
(4) $T := T - \{(a, e) | a \in \Sigma_1\}$;
(5) **if** $e \in \text{range}(T)$ **then reject**;
(6) $\Sigma_2 := \text{range}(T) \cap \Sigma$;
(7) **while** $\Sigma_2 \neq \emptyset$ **do**
(8) **begin choose** a letter $a \in \Sigma_2$ together with a rule $(u, a) \in T$;
(9) **if** $|u|_a > 0$ **then reject**;
(10) $\Sigma_2 := \Sigma_2 - \{a\}$;

(11) $\Sigma := \Sigma - \{a\}$;
(12) $T := T - \{(u, a)\}$;
(13) **substitute** each occurrence of a in a rule of T by the word u
end;
(14) **if** $T \subseteq \{(w, w) | w \in \Sigma^*\}$ **then accept else reject**
end.

LEMMA 5.1. *Algorithm* (A) *accepts an input* $(\Sigma; T)$, *where* T *is a finite Church-Rosser Thue system on* Σ, *if and only if the monoid* M *presented by* $(\Sigma; T)$ *is a free monoid.*

Proof. Let Σ be a finite alphabet, and let T be a finite Church-Rosser Thue system on Σ. Then there exists a unique reduced Church-Rosser Thue system on Σ that is equivalent to T, and this reduced system can be computed from T in polynomial time using linear space [9]. Therefore, we may assume without loss of generality that T is a reduced system. Since T is Church-Rosser, we have for all $a \in \Sigma$, $a \leftrightarrow_T^* e$ if and only if $(a, e) \in T$. Hence, $\Sigma_1 = \{a \in \Sigma | (a, e) \in T\}$ contains all the letters from Σ that present the identity of the monoid $M = \Sigma^*/\leftrightarrow_T^*$. Since T is reduced, the letters from Σ_1 do not occur in the rules of T other than $\{(a, e) | a \in \Sigma_1\}$, and so M is also presented by $(\Sigma - \Sigma_1; T - \{(a, e) | a \in \Sigma_1\})$. Hence, after executing lines (1) to (4) of (A) we have a presentation of M by a finite reduced Church-Rosser Thue system containing no rules of the form (a, e) with $|a| = 1$. This presentation is again called $(\Sigma; T)$.

If M is a free monoid, then there is a subset Σ_0 of Σ that freely generates M. Let $\Sigma_2 = \Sigma - \Sigma_0$, then $\Sigma_2 = \text{range}(T) \cap \Sigma$ by Lemma 4.2. Further, by Lemma 4.1 T contains no rules with right-hand side e, i.e., $e \notin \text{range}(T)$. Hence, (A) does not reject in line (5).

Let $a \in \Sigma_2$. Then there is at least one rule of the form (u, a) in T. Choose one such rule (u, a). Since $|u| \geqq 2$ and $u \in (\Sigma_0 \cup \Sigma_2)^*$, we have $|u|_a = 0$ by Corollary 4.3, and so (A) does not reject in line (9). In lines (11) to (13) (A) performs a Tietze transformation [6] that results in deleting the letter a and the rule (u, a) from the presentation $(\Sigma; T)$. Also, a is deleted from the set Σ_2 giving the set Σ_2'.

Let $(\Sigma'; T')$ denote the presentation of M this Tietze transformation yields. Then for each $b \in \Sigma_2'$, T' contains at least one rule of the form (v, b) with $v \in \Sigma'^*$. Hence, $v \in (\Sigma_0 \cup \Sigma_2')^* \subseteq (\Sigma_0 \cup \Sigma_2)^*$, and $v \leftrightarrow_T^* b$ implying that $|v|_b = 0$. Thus, (A) performs the loop (7)-(13) until $\Sigma_2 = \varnothing$ without rejecting in line (9).

When this loop is left, the presentation of M at this point is $(\Sigma_0; T')$ for some $T' \subseteq \Sigma_0^* \times \Sigma_0^*$. But M is freely generated by Σ_0, and so $T' \subseteq \{(w, w) | w \in \Sigma_0^*\}$, i.e., (A) accepts.

Now assume that (A) accepts, i.e., (A) transforms the presentation $(\Sigma; T)$ into a presentation $(\Sigma_0; T')$ for some $\Sigma_0 \subseteq \Sigma$ with $T' \subseteq \{(w, w) | w \in \Sigma_0^*\}$. During this transformation (A) only applies Tietze transformations to presentations of M starting with $(\Sigma; T)$. Hence, $(\Sigma_0; T')$ is a presentation of M showing that M is a free monoid generated by Σ_0. □

Hence, we have shown the following.

THEOREM 5.2. *The following problem is decidable*:

Instance. A presentation $(\Sigma; T)$, *where* T *is a finite Church-Rosser Thue system on* Σ.
Question. Is the monoid presented by $(\Sigma; T)$ *free*?

Notice that it is decidable in polynomial time and linear space whether a given finite Thue system is Church-Rosser [3]. So, what is the complexity of Algorithm (A)? Letters from Σ_2 may represent long words from Σ_0^*. Thus, if all the words created during the execution of (A) are actually written down as words, then exponential time

and space may be used. However, if only pointers to the subwords inserted in line (13) are stored, then (A) can be executed in linear space. This gives the following.

COROLLARY 5.3. *The problem described in Theorem* 5.2 *is decidable in linear space.*

So the problem of deciding whether the monoid defined by a given presentation is free gives another example of a decision problem, that, although being undecidable in general, becomes decidable when it is restricted to presentations involving finite Church-Rosser Thue systems.

We conclude with an example of a nontrivial presentation of a free monoid.

Example 5.4. Let $\Sigma = \{a, b, c, d_1, \cdots, d_{n+1}, f, g\}$ for some $n \geqq 1$, and let $T = \{(a^2ba^2c, d_1), (bd_nc^2, f), (d_{n+1}c, g), (d_nf, g)\} \cup \{(d_ibd_ic, d_{i+1}) | i = 1, \cdots, n\}$. Then T is a finite Thue system on Σ, and it can be checked easily that T is reduced and Church-Rosser.

Take $\Sigma_0 = \{a, b, c\}$, and define $\varphi: \Sigma^* \to \Sigma_0^*$ by $\varphi(a) = a$, $\varphi(b) = b$, $\varphi(c) = c$, $\varphi(d_1) = a^2ba^2c$, $\varphi(d_{i+1}) = \varphi(d_i)b\varphi(d_i)c$ for $i = 1, \cdots, n$, $\varphi(f) = b\varphi(d_n)c^2$, and $\varphi(g) = \varphi(d_n)\varphi(f)$. Applying (A) to the input $(\Sigma; T)$ yields the presentation $(\Sigma_0; \{(\varphi(d_{n+1})c, \varphi(g))\})$. Since $\varphi(g) = \varphi(d_n)\varphi(f) = \varphi(d_n)b\varphi(d_n)c^2 = \varphi(d_{n+1})c$, the monoid M presented by $(\Sigma; T)$ is free on Σ_0, and (A) accepts. In particular, φ is the homomorphism that for each word $w \in \Sigma^*$, gives the corresponding word from Σ_0^*. It is easy to see that, for all $i = 1, \cdots, n+1$, $|\varphi(d_i)| = 2^{i+2} - 2$, $|\varphi(f)| = 2^{n+2} + 1$, and $|\varphi(g)| = 2^{n+3} - 1$. So in fact the letters f and g represent long words from Σ_0^*.

Acknowledgments. The author wishes to thank Prof. K. Madlener and Dr. C. Wrathall for fruitful discussions concerning the results presented here.

REFERENCES

[1] R. V. BOOK, *Confluent and other types of Thue systems*, J. Assoc. Comput. Mach., 29 (1982), pp. 171-182.

[2] ———, *Decidable sentences of Church-Rosser congruences*, Theoret. Comput. Sci., 24 (1983), pp. 301-312.

[3] R. V. BOOK AND C. O'DÚNLAING, *Testing for the Church-Rosser property*, Theoret. Comput. Sci., 16 (1981), pp. 223-229.

[4] G. LALLEMENT, *Semigroups and Combinatorial Applications*, Wiley-Interscience, New York, 1979.

[5] M. LOTHAIRE, *Combinatorics on Words*, Addison-Wesley, Reading, MA, 1983.

[6] W. MAGNUS, A. KARRASS AND D. SOLITAR, *Combinatorial Group Theory*, 2nd rev. ed., Dover, New York, 1976.

[7] A. MARKOV, *Impossibility of algorithms for recognizing some properties of associative systems*, Dokl. Akad. Nauk SSSR, 77 (1951), pp. 953-956.

[8] A. MOSTOWSKI, *Review of* [7], J. Symbolic Logic, 17 (1952), pp. 151-152. (In Russian.)

[9] P. NARENDRAN, *Church-Rosser and related Thue systems*, Doctoral Dissertation, Rensselaer Polytechnic Institute, Troy, NY, 1984.

[10] P. NARENDRAN AND F. OTTO, *Complexity results on the conjugacy problem for monoids*, Theoret. Comput. Sci., 35 (1985), pp. 227-243.

[11] ———, *The problems of cyclic equality and conjugacy for finite complete rewriting systems*, submitted for publication.

[12] F. OTTO, *Some undecidability results for non-monadic Church-Rosser Thue systems*, Theoret. Comput. Sci., 33 (1984), pp. 261-278.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
012

THREE-DIMENSIONAL CIRCUIT LAYOUTS*

FRANK THOMSON LEIGHTON† AND ARNOLD L. ROSENBERG‡

Abstract. Recent advances in fabrication technology have rendered imminent the fabrication of multilayer (i.e., three-dimensional) chips, wafers, and packages. In this paper, we examine the savings in material (as measured by Area in a two-dimensional medium and Volume in a three-dimensional one) and in communication time (as measured by the length of the longest uninterrupted run of wire) afforded by this developing technology. We derive close upper and lower bounds on the efficiency with which circuits can be realized in a multilayer medium, based on the sizes of the smallest *bifurcators* of the circuit. We find that the smallest Volume of any three-dimensional layout of an N-device circuit is no more than (roughly) $(NA)^{1/2}$, where A is the smallest Area of any two-dimensional layout of the circuit. We then refine our layout techniques so that we can deal with multilayer layouts having a fixed number of layers. We find that we can efficiently transform a two-dimensional layout of Area A and Maximum Wire Run R into a three-dimensional layout of Volume (roughly) $V = A/H$ and Maximum Wire Run $R^* = R/H$ for moderate numbers of layers H. Two noteworthy features of the study are: (1) that, within logarithmic factors, the indicated savings can be realized with layouts that use the third dimension only for interconnect; and (2) that the indicated savings can be realized algorithmically: we present polynomial-time algorithms that transform a given two-dimensional layout into a more efficient three-dimensional one.

Key words. graph embeddings, graph layouts, VLSI theory, multilayer circuit realizations, one-active layer layouts, area-volume tradeoffs

AMS(MOS) subject classifications. 94C15, 68C25, 05C99, 68E10

1. Introduction. Recent advances in fabrication technology [4]-[8], [10]-[12], [17]-[19], [22], [28] have allowed circuit and system designers to begin using the third dimension in realizing their designs. Multilayer packages with impressive performance have been fabricated [9], [10], [20], and there has been extensive research toward the goal of three-dimensional chips [8], [12], [17]-[19], [28]. The rapid rate of progress in VLSI technology suggests that multilayer chips and packages will be commonplace in the not-distant future. Indeed, the president of Texas Instruments (quoted in [8]) predicts the production of three-dimensional chips by the end of the decade.

One expects at least two benefits to accrue from the use of the third dimension in circuit realization. First, since one can avoid obstacles by using the third dimension, runs of wire should be shorter, at least in the worst case. Second, since avoiding obstacles in a two-dimensional environment can require area-consuming circuitous routing of wires, one would expect savings in material: the Volume of a three-dimensional realization of a circuit should be less than the Area of any two-dimensional realization of the circuit. In order to realize these expected benefits, we must develop effective techniques for devising and analyzing multilayer circuit layouts. Such is the goal of this paper: we develop and analyze an algorithmic strategy for laying out VLSI circuits (viewed here as undirected graphs) in three-dimensional chips (viewed here as three-dimensional grids).

Our notion of the layout of a circuit follows the two-dimensional framework of [2], [13], [14], [16], [26], [27], as adapted for the third dimension in [23], [24]: circuits are undirected graphs whose vertices correspond to active devices (transistors, gates,

* Received by the editors February 17, 1984, and in revised form May 13, 1985.

† Department of Mathematics and Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The research of this author was supported in part by a Bantrell Fellowship, in part by DARPA contract N00014-80-C-0622, and in part by Air Force contract OSR-82-0326.

‡ Department of Computer Science, Duke University, Durham, North Carolina 27706. The research of this author was supported in part by National Science Foundation grants MCS-81-16522 and MCS-83-01213.

etc.) and whose edges correspond to wires connecting these devices. The media in which the circuits are to be realized are (two- or three-dimensional) rectangular grids. A circuit layout is an edge-disjoint embedding of the circuit-graph in the grid.

Two models have been proposed for studying three-dimensional VLSI [23], [24]. The first, *one-active-layer*, model requires that all active devices be placed on a designated layer of the chip. The second, unrestricted, *many-active-layer*, model allows devices to be placed arbitrarily throughout the chip. It is clear how these two possibilities manifest themselves in our formal setting. Although the many-active-layer model affords one more flexibility when laying out one's circuits, it places significantly more stringent demands on the fabrication technology; cf. [24]. There is thus a tradeoff between the cost of fabricating a chip with multiple layers of devices and the savings (in terms of Volume and Maximum Wire Run) resulting from the increased layout flexibility. One of our more surprising results here is that, at least within our abstract framework, many-active-layer layouts are little or no more efficient than one-active-layer layouts when the number of layers is relatively small: either mode of using the third dimension affords one appreciable but similar savings over any two-dimensional layout. Additionally, we show that multiple layers are effective in reducing Volume and Maximum Wire Run only up to a certain point, after which they are wasteful. Although these results are definitive only for the theoretical model our analysis is based on, they suggest strongly that VLSI chips that have a higher (and costlier) degree of sophistication (in terms of number of layers and placement of devices) may not be more efficient for many applications than significantly more modest chips.

Although there has been a substantial amount of work on the two-dimensional version of the layout problem, related work on the three-dimensional problem has largely been confined to one of:

* the study of routing in the presence of a few extra layers [3], [9], [21];
* the study of optimal multilayer layouts for a few special networks [20], [23], [29];
* the study of optimal multilayer layouts for the class of "hardest-to-realize" networks [23], [24].

Notable among the results in these papers, for our purposes, is the use in [23], [24] of optimal three-dimensional layouts of the N-input Benes permutation network [1] to prove:

> Every small-degree N-vertex graph can be laid out in a three-dimensional grid with Volume $O(N^{3/2})$ and wire-length $O(N^{1/2})$.

There exist graphs that do not admit any more compact layout; for such graphs, these bounds contrast with the lower bounds of Area $\Omega(N^2)$ and wire-length $\Omega(N)$ [13], [26] in the two-dimensional case. In effect, the contribution of the present paper is to generalize the specialized three-dimensional results of Rosenberg and Preparata (among others) to a level of generality comparable to the two-dimensional work of Bhatt, Leighton, Leiserson, Thompson and Valiant (among others). Perhaps the most important contribution of this paper is an algorithm that transforms a two-dimensional circuit layout of Area A and Maximum Wire Run R into a three-dimensional layout of the circuit that is within logarithmic factors of Volume A/H and Maximum Wire Run R/H, for moderate values of H. The layouts produced are close to optimal in the sense that using H layers rather than just one layer (which is how the two-dimensional case is viewed in our formal framework) can never improve Area or Maximum Wire Run by a factor smaller than $1/H$. Certain special situations where the logarithmic factors can be avoided are described in [15], wherein is also a special case of our algorithm.

The remainder of the paper is divided into four sections. In § 2 we review basic definitions and cite work on two-dimensional layouts that is relevant to our study. Sections 3 and 4 are devoted to the development and analysis of our three-dimensional layout strategy, with particular attention paid to issues of Volume and Maximum Wire Run. We conclude in § 5 with some remarks on the implications of our work.

2. Preliminaries. *Underlying assumptions.* The formal framework of our study carries with it certain implicit assumptions:

1. Our associating circuits with graphs limits our study to circuits with two-point nets.

2. Our associating chips with grids limits our circuits to having small vertex-degrees, specifically $\leqq 4$.

3. Our adherence to the models of VLSI layout theory renders the vertices of our circuits as unit-side squares or cubes.

4. Our method of extending the two-dimensional model assumes isometry in all dimensions: a unit of height is equivalent to a unit of width.

It is worthwhile placing these assumptions in perspective.

1. The restriction to two-point nets is a significant one: although extending our results to circuits with three- or four-point nets is not difficult, extending the results to circuits with arbitrary multipoint nets remains an inviting challenge.

2. Techniques that are now standard can be used to generalize our results to include circuits with high vertex-degrees, but the associated analysis is technically somewhat more complicated.

3. Restricting attention to unit-side devices is a purely clerical device; extending the analysis to any uniform-size devices should present no problem [2], [13].

4. Aside from clerical simplification, the isometry assumption acknowledges the potential problem of cross-talk between parallel runs of wire [24]; moreover, our results concerning fixed-height layouts can be applied to a non-isometric model.

The formal framework. An undirected graph comprises a finite set V of *vertices* and a set of two-element subsets of V, called *edges*. We say that the edge $\{u, v\}$ is *incident* to vertices u and v. The *degree* of the vertex v is the number of edges incident to v; the *degree* $D(G)$ of G is the largest degree of any of its vertices. As noted earlier, our desire to embed graphs in grids forces us to look at graphs with small degrees. In particular, our desire to compare three-dimensional layouts with competing two-dimensional layouts restricts us to graphs of degree at most four.

The $W \times L$ *planar grid* is the graph whose vertex-set is the set of pairs $[W] \times [L]$ and whose edges connect vertices $\langle a, b\rangle$ and $\langle c, d\rangle$ just when $|a-c|+|b-d|=1$. (Here and throughout, $[n]$ denotes the set $[n]=\{1, 2, \cdots, n\}$.) The $H \times W \times L$ *solid grid* is the graph whose vertex-set is the set of triples $[H] \times [W] \times [L]$ and whose edges connect vertices $\langle a, b, c\rangle$ and $\langle d, e, f\rangle$ just when $|a-d|+|b-e|+|c-f|=1$.

An *embedding* or *layout* of the graph G in the grid Γ (solid or planar) is a one-to-one association β of the vertices of G with vertices of Γ, together with a one-to-one association α of the edges of G with edge-disjoint paths in Γ, subject to the constraint that the path $\alpha(u, v)$ cannot pass through any vertex-image $\beta(w)$ other than $\beta(u)$ and $\beta(v)$. An embedding in a solid grid Γ of dimensions $H \times W \times L$ is a *one-active-layer* embedding if it associates all vertices of G with vertices of Γ of the form $\langle i_0, j, k\rangle$ for some fixed layer i_0 in $[H]$.

We gauge the cost of an embedding of a graph in a grid in terms of the amount of material consumed by the embedding (Area in the two-dimensional case and Volume

in the three-dimensional case), and in terms of the maximum length of any run of wire that does not encounter a device.

The *Volume* (resp., *Area*) *of an embedding* of the graph G in a solid (resp., planar) grid Γ is the product of the dimensions of Γ. The *Volume* (resp., *Area*) *of the graph* G, $\mathrm{VOL}(G)$ (resp., $\mathrm{AREA}(G)$), is the minimum Volume (resp., Area) of any embedding of G in a solid (resp., planar) grid. The *one-active-layer Volume of the graph* G, $\mathrm{VOL}_{1-AL}(G)$, is the minimum Volume of any one-active-layer embedding of G in a solid grid. When we relativize either $\mathrm{VOL}(G)$ or $\mathrm{VOL}_{1-AL}(G)$ with the integer parameter H, as in $\mathrm{VOL}(G; H)$ or $\mathrm{VOL}_{1-AL}(G; H)$, it is to be understood that the volume minimization is done over all H-layer embeddings (of the appropriate kind).

Say that we are considering an embedding of the graph G in a grid, with the (graph edge)-(grid path) association α. The *wire-length of the embedding* is the maximum length of any path $\alpha(e)$ over all edges e of G. This corresponds informally to the Maximum Wire Run of the layout, i.e., the length of the longest run of wire that does not encounter a device. The *solid* (resp., *planar*) *wire-length of the graph* G, $\mathrm{WL}_3(G)$ (resp., $\mathrm{WL}_2(G)$), is the minimum wire-length of any embedding of G in a solid (resp., planar) grid. The *one-active-layer wire-length of the graph* G, $\mathrm{WL}_{1-AL}(G)$, is the minimum wire-length of any one-active-layer embedding of G in a solid grid. As before, relativization of these measures with the integer parameter H, as in $\mathrm{WL}_3(G; H)$ or $\mathrm{WL}_{1-AL}(G; H)$, restricts the indicated minimization to H-layer embeddings of the appropriate kind.

Leiserson [16] and Valiant [27] showed that the "decomposition structure" of a graph could be exploited in order to find an efficient two-dimensional layout of the graph. Leighton [14] and Thompson [26] proved that the Leiserson-Valiant strategy could not be improved in general, though it often produced layouts that could be dramatically improved. Bhatt and Leighton [2], [13] significantly improved the layout strategy by recasting its framework. Specifically, they reformulated the underlying notion of the "decomposition structure" of a graph to one in which the Leiserson-Valiant strategy yielded layouts that were provably good, in the sense of being within logarithmic factors of optimal, for any graph. One of the central ideas in the Bhatt-Leighton framework is that of a *decomposition tree* for a graph. The graph G has an $(F_0, F_1, \cdots, F_r)$-*decomposition tree* if G can be decomposed into two subgraphs G_0

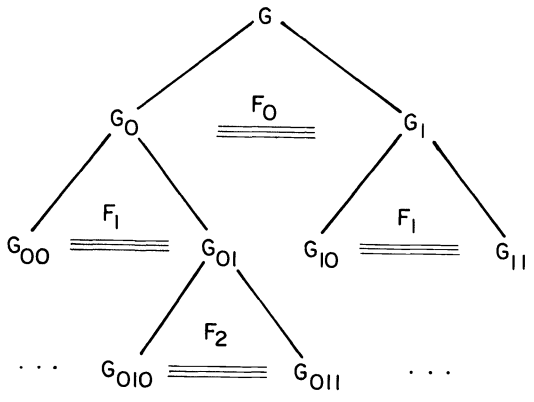


FIG. 1. *An* $(F_0, F_1, \cdots, F_r)$*-decomposition tree.*

and G_1 by removing at most F_0 edges from G; each of G_0 and G_1 can be decomposed into two subgraphs by removing at most F_1 edges from each; and so on, until each subgraph produced by the decomposition is either empty or an isolated vertex. See Fig. 1.

Decomposition trees for which the F_i decrease at a uniform rate are of particular importance to us. A graph that has an $(F, F/\rho, F/\rho^2, \cdots, 1)$-decomposition tree for some real $\rho > 1$ is said to have an (F, ρ)-*bifurcator* or, equivalently, a *ρ-bifurcator of size F*. Since the decomposition tree of an N-vertex graph must have at least $\log N$ levels, it is clear that $F \geqq N^{\log \rho}$. (Unless otherwise indicated, all logarithms are to the base 2.) For convenience, we shall also assume that $F \leqq N/2$ for all graphs.

Returning to the issue of efficient two-dimensional layouts, Bhatt and Leighton proved that finding a small $2^{1/2}$-bifurcator for the graph to be laid out was the entire story, in the sense of the following result.

THEOREM 2.1 [2], [13]. *Let F be the size of the smallest $2^{1/2}$-bifurcator of the N-vertex graph G. Then*

$$F^2 \leqq \text{AREA}(G) \leqq (\text{const}) F^2 \log^2 (N/F)$$

and

$$(\text{const}) F^2/N \leqq \text{WL}_2(G) \leqq (\text{const}) F \log (N/F)/\log \log (N/F).$$

Moreover, these bounds are existentially tight in the sense that each of the four inequalities is sometimes an equality.

A key step in the proof of Theorem 2.1 is the demonstration that an arbitrary decomposition tree can be *fully balanced* at little or no cost, in the sense that

(1) each graph G_i in the tree is split into two equal-size subgraphs, G_{i0} and G_{i1}; and

(2) the number of edges of G having precisely one end in the (arbitrary) tree-vertex/subgraph G_{ia} of G is at most a small fixed multiple of the number of edges leaving G_{ia} to go to its brother subgraph $G_{i\bar{a}}$.

The notion "fully balanced" applies to ρ-bifurcators in the obvious way. Bhatt and Leighton prove the following basic result, via a polynomial-time algorithm for constructing a fully balanced bifurcator from a given arbitrary one.

LEMMA 2.2 [2], [13]. *There is a fixed constant $c > 0$ such that, if the graph G has a ρ-bifurcator of size F, then it has a fully balanced ρ-bifurcator of size cF.*

Lemma 2.2 guarantees that any graph with an (F, ρ)-bifurcator has a decomposition tree in which any subgraph G_w on level i of the tree is incident to at most cF/ρ^i edges of G that are not wholly contained within G_w. Lacking Lemma 2.2, we would know only that at most F/ρ^i edges of G link G_w to its *brother* in the decomposition tree (as opposed to *any* other subgraph at level i of the tree).

A second technical lemma is crucial to our layout strategy. A *multigraph* comprises a set V of *vertices* and a multiset M of doubleton subsets of V, called *edges*. Thus a multigraph can be viewed as a graph in which each pair of vertices can be connected by several edges. The multigraph notions of "incidence", "degree of a vertex", and "degree of a multigraph" derive immediately from the corresponding notions for graphs. An *edge-coloring* of a multigraph is a labelling of the edges of the multigraph with "colors" in such a way that edges incident to the same vertex get labelled with distinct colors. Shannon [25] showed, via an efficient algorithm for edge-coloring multigraphs, that one needs never use a lot of colors to edge-color a small-degree multigraph.

LEMMA 2.3 [25]. *Any multigraph G can be edge-colored using at most $\lceil 3D(G)/2 \rceil$ colors. Moreover, this bound is existentially tight.*

3. Efficient three-dimensional layouts.

3.1. One active-layer-layouts. We consider first the problem of embedding a graph in a three-dimensional grid in accordance with the one-active layer model, i.e., so that all of the graph's vertices reside on a single layer of the layout. We assume that we have in hand a minimal-size $(F, 2^{1/2})$-bifurcator for the graph to be laid out, as well as an associated recursive decomposition of G.

THEOREM 3.1. The One-Active-Layer Layout Theorem. *Let G be an N-vertex graph, and let F be the size of its minimum $2^{1/2}$-bifurcator.*

Height-H layout. There is a constant $h > 0$ such that, for any height H in the range

$$1 \leqq H \leqq h \frac{F}{N^{1/2}} \log (N/F),$$

the height-H one-active-layer layouts of G satisfy

$$\max\left(FN^{1/2}, \frac{F^2}{H}\right) \leqq \mathrm{VOL}_{1-AL}(G; H) \leqq (\text{const}) \frac{F^2}{H} \log^2 (N/F)$$

and

$$(\text{const}) \max\left(\frac{F}{N^{1/2}}, \frac{F}{HN}\right) \leqq \mathrm{WL}_{1-AL}(G; H) \leqq (\text{const}) \frac{F}{H} \log (N/F).$$

Unrestricted-height layout. The minimum-resource one-active-layer layout of G satisfies

$$FN^{1/2} \leqq \mathrm{VOL}_{1-AL}(G) \leqq (\text{const}) FN^{1/2} \log (N/F)$$

and

$$(\text{const}) \frac{F}{N^{1/2}} \leqq \mathrm{WL}_{1-AL}(G) \leqq (\text{const}) N^{1/2};$$

moreover, the number of layers (H) that minimizes VOL_{1-AL} is at most (const)-$(F/N^{1/2}) \log (N/F)$.

Given F and an associated recursive decomposition of G, the embeddings yielding the upper bounds can be found in time polynomial in N.

Proof. Let G and F be as in the statement of Theorem 3.1.

The lower bounds. We present two proofs that expose different aspects of the situation.

Proof 1. Consider an arbitrary one-active-layer layout of G, having Volume V, height H, and base area B. Let us recursively bisect this "box" across the smaller of its base dimensions in such a way that the base area is halved with each bisection. The boxes we bisect at stage i of this recursion (we start at stage 0) have height H and base area $B/2^i$. When we bisect each of these boxes, we are severing no more than $(B/2^i)^{1/2} H$ edges of G, since the area of the cutting plane is no greater than this, and since wires have unit cross-sections. This means that G has a $(B^{1/2}H, 2^{1/2})$-bifurcator. Given that F is the size of G's smallest $2^{1/2}$-bifurcator, it is immediate that $F \leqq B^{1/2}H$, so $F^2 \leqq BH^2 = VH$, so

$$V \geqq \frac{F^2}{H}.$$

Moreover, the fact that all vertices of G lie on one layer implies that $B \geqq N$; hence,

$$F \leqq \frac{V}{B^{1/2}} \leqq \frac{V}{N^{1/2}},$$

whence

$$V \geqq FN^{1/2}.$$

Since we have been looking at an arbitrary height-H one-active-layer layout of G, the lower bounds on Volume follow.

Proof 2. Our second, indirect, proof yields a lower bound on wire-length also.

The key step here is to transform an H-layer layout of the graph G with Volume $V = BH$ (B being the area of the base of the layout) into a two-dimensional layout with Area $9BH^2$. We shall then be able to conclude that

$$\text{AREA}(G) \leqq 9BH^2 = 9VH,$$

so that

$$V \geqq \frac{A}{9H}.$$

Since (as before) $B \geqq N$, we shall also be able to conclude that

$$\text{AREA}(G) \leqq 9BH^2 = \frac{9V^2}{B} \leqq \frac{9V^2}{N},$$

so that

$$V \geqq (NA)^{1/2}/9.$$

By Theorem 2.1 we know that $\text{AREA}(G) \geqq F^2$, and thus

$$V \geqq (\text{const}) \max\left(FN^{1/2}, \frac{F^2}{H}\right).$$

The desired transformation has two steps. First we project the H-layer grid holding the layout of G onto the plane, as illustrated in Fig. 2. Ignoring for the moment that the projection produces diagonal edges (corresponding to the edges that changed layers in the solid grid), it converts an $H \times W \times L$ solid grid into an $HW \times HL$ planar pseudo-grid ("pseudo-" because of the diagonal edges). The second step of our transformation replaces this planar pseudo-grid by a $3HW \times 3HL$ planar grid by: (1) adding two new vertical grid lines to the right of each vertical line in the pseudo-grid; (2) adding two new horizontal grid lines below each horizontal line in the pseudo-grid; (3) deleting the diagonal edges.

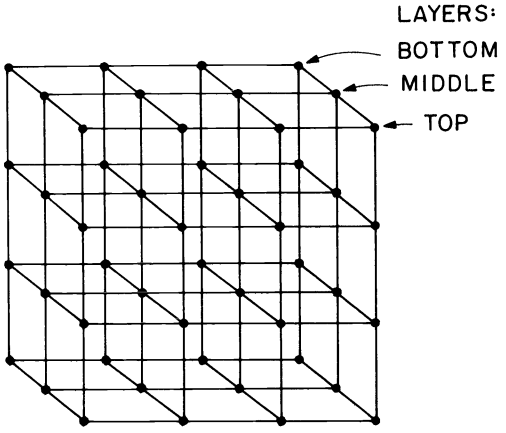
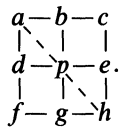


FIG. 2. *The two-dimensional projection of the* 3*-layer* 4×4 *grid.*

We must now show how to reroute those edges of G that used the diagonal (layer-changing) grid-edges in the layout—all edges that used only rectilinear edges are extended in the obvious way through the new gridpoints with no changes in direction. To this end, note that the net effect of the second step of our transformation has been to surround each "old" gridpoint p by a *box* of eight "new" gridpoints, of the generic form:



Our task is to reroute diagonal runs of wire that run within these boxes, as well as those that run between boxes.

Edges between boxes. Diagonal edges that run between boxes are easy to reroute: each can simply be rerouted in the neighboring "right angle". (All segments of that right angle are new, hence devoid of wires.)

Edges within boxes. The rerouting of diagonal runs of wire within the boxes will be described by reference to our generic box above. We consider two cases.

Gridpoint p holds a vertex of graph G. Let the vertex v of G reside at gridpoint p. Since v has degree $\leqq 4$, if k ($=0$, 1 or 2) of the diagonal edges incident to p hold wires that realize edges of G that are incident to v, then a like number of rectilinear edges incident to p do not hold wires that realize edges of G. One can, therefore, reroute the diagonal edge(s) by using the peripheral edges of the box followed by one rectilinear edge. For example, if both diagonal edges a-p and p-h are used, and if rectilinear eges b-p and d-p are unused, then we can reroute

$$a\text{-}p \quad \text{by} \quad a\text{-}b\text{-}p$$

and

$$p\text{-}h \quad \text{by} \quad p\text{-}d\text{-}f\text{-}g\text{-}h.$$

Gridpoint p does not hold a vertex of graph G. In this case, all six of the edges incident to p could hold wires. We shall never reroute wires passing through p that use only rectilinear edges, so we can restrict attention to the wires passing through the diagonal edges. We consider three subcases.

(1) Say first that an edge of G runs through both diagonal edges (i.e., in the original layout it changed at least two layers at point p). Then we can reroute

$$a\text{-}p\text{-}h \quad \text{by} \quad a\text{-}b\text{-}c\text{-}e\text{-}h.$$

(2) Say next that an edge of G uses a diagonal edge and makes an acute angle, such as a-p-d. We replace the acute angle by the peripheral edge connecting its endpoints, as when replacing

$$a\text{-}p\text{-}d \quad \text{by} \quad a\text{-}d.$$

(3) Say finally that an edge of G uses a diagonal edge and makes an obtuse angle, such as a-p-e. If there is only one such obtuse angle, then we replace that path by a peripheral path connecting its endpoints, as when replacing

$$a\text{-}p\text{-}e \quad \text{by} \quad a\text{-}b\text{-}c\text{-}e;$$

one of the peripheral paths must be free if there is only one obtuse angle. If there are two obtuse angles, such as a-p-e and b-p-h, then we replace the first by a peripheral

path connecting its endpoints, and the second by a Z-shaped path using the inner rectilinear edge freed up by the rerouting of the first, as when replacing

$$a\text{-}p\text{-}e \quad \text{by} \quad a\text{-}b\text{-}c\text{-}e$$

and

$$b\text{-}p\text{-}h \quad \text{by} \quad b\text{-}p\text{-}e\text{-}h.$$

The area of the two-dimensional layout resulting from our transformation and rerouting is

$$9WLH^2 = 9BH^2,$$

as was claimed.

The same transformation yields the lower bound on wire-length: the projection maps unit-length vertical and horizontal segments onto length-H segments. Unit-length segments that run between layers are transformed into segments of length $\leqq 4$ when rerouted. Thus a wire of length L_3 in the H-layer layout is transformed into a wire of length $L_2 \leqq 4HL_3$ in the two-dimensional layout. We can now apply Theorem 2.1 to conclude that

$$L_3 \geqq (\text{const}) \frac{F^2}{NH}.$$

Since all of the vertices lie on a single layer, we know also that the longest wires have length $L_3 \geqq H$ if all H layers are used. This combines with the previous inequality to show that

$$L_3 \geqq (\text{const}) \frac{F}{N^{1/2}},$$

as was claimed.

Although we did not do so here, we could also show that the *average* H-layer wire-length for any N-vertex graph is at least

$$(\text{const}) \max\left(\frac{F}{N^{1/2}}, \frac{F^2}{HN}\right).$$

The upper bounds. The upper bounds are substantially more intricate to establish. Our task is lightened, though, by the fact that we can establish both the restricted-height and unrestricted-height upper bounds via a single construction, the latter bound following from the former by assigning to H its maximum allowed value. We shall, therefore, prove only the restricted-height upper bound, assuming that a legitimate target height H has been specified; this H is fixed henceforth. We establish the bound by means of a construction that recursively produces an embedding with the desired Volume for a subgraph G_i of G on level i of G's decomposition tree, given the appropriate embeddings of the four subgraphs comprising G_i, which occur on level $i+2$ of the tree. Our main task will be to route wires for those edges that have precisely one endpoint in a subgraph. Our strategy will be to route wires for such edges in a bottom-up manner, and to connect up these edges only when we process the level of the decomposition tree where these edges were removed. To aid the reader in following this procedure, we include Figs. 3 and 4.

Let us concentrate on one graph G_i at level i of G's decomposition tree, and on the four subgraphs comprising G_i at level $i+2$ of the tree. By Lemma 2.2, we know that G has a fully balanced $2^{1/2}$-bifurcator of size bF for some constant b and thus

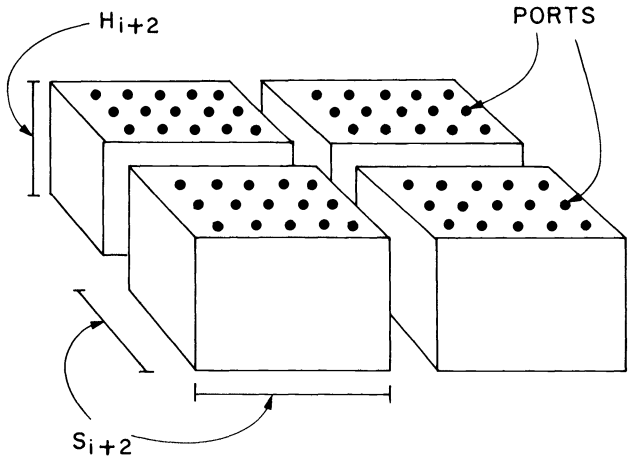


FIG. 3. *The given one-active-layer layouts of the four subgraphs of* G_i.

that each of the four subgraphs comprising G_i has a fully balanced $2^{1/2}$-bifurcator of size $bF/2^{(i+2)/2}$. Hence we assume inductively that we have in hand one-active-layer layouts for these four subgraphs, each layout having height H_{i+2} and a square base of side

$$S_{i+2} =_{\text{def}} h\frac{F\log(N/F)}{H}2^{-(i+2)/2}$$

(h being the constant in the statement of the theorem). (Since $S_{i+2} \geqq (N/2^{i+2})^{1/2}$ when H is in the indicated range, the base of the layout is indeed big enough to accommodate one-fourth of G_i's vertices.) Assume further that each edge of G that has precisely one end in one of the subgraphs, is represented by a wire routed from the appropriate vertex of that subgraph to the top layer of the layout. Finally, assume that each one of these "dangling" edges terminates at a *port* at the top of the layout and that these ports are evenly distributed across the top layer of the layout. (By a *port* here we mean an end of a wire that can be extended upwards if additional layers are added to the top of the embedding; our assumption about even distribution means that the ports are spaced uniformly, as suggested in Fig. 3.) We show now how to construct from the layouts of these level-$(i+2)$ subgraphs of G_i an inductively consistent layout of G_i,

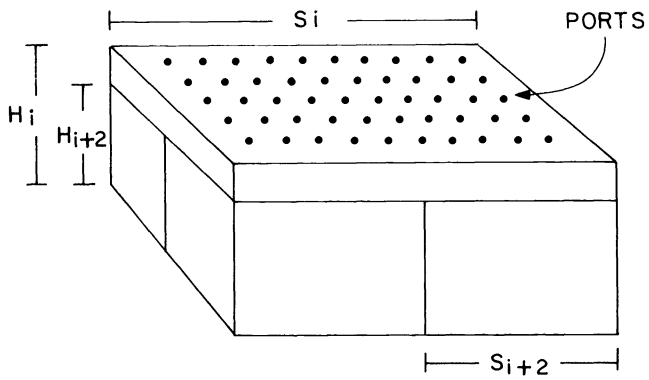


FIG. 4. *Coalescing the one-active-layer layouts of the subgraphs of* G_i.

having height

$$H_i = H_{i+2} + \frac{H}{d \log (N/F)}$$

for some suitably large constant d.

We begin by merging the layouts of the four subgraphs into a single "box" having height H_{i+2} and having a square base of side

$$S_i =_{\text{def}} h \frac{F \log (N/F)}{H} 2^{-i/2}.$$

We then add $H/(d \log (N/F))$ new (empty) layers to the top of the box, thereby building it up to height H_i; see Fig. 4. Next we establish the ports at the top of the new box, that will be needed to extend this construction to a yet-higher level of G's decomposition tree. By Lemma 2.2, no more than $bF/2^{i/2}$ edges of G have precisely one end in G_i, for some specified constant b; hence we need create at most this many ports at the top of the new box. We create these ports, spaced evenly throughout the top layer. Finally, we are ready to turn to the task of routing the wires incident to the ports of the original boxes (in layer H_{i+2}). Some of these wires must get routed to other ports in the same layer and some to the new ports at the top of the new box.

We effect the necessary routings by using each of the new layers to route an average of $S_i/2$ wires to their appropriate row and column (in one of the new layers). Final connections will then be a simple matter. Since we need to route at most $\frac{3}{2}bF/2^{i/2}$ wires in all, the allotted number of empty layers (namely, $H/(d \log (N/F))$) will suffice, provided that h was chosen sufficiently large. We now describe the details of the routing.

Layer assignment. The first phase of the routing assigns each wire to the layer of the embedding on which it will be routed. To this end, we temporarily superimpose layers H_{i+2} and H_i of the embedding; and we partition the resulting pseudo-layer into $S_i/4$ square regions of area $4S_i$ each. Let M denote the multigraph which has one vertex corresponding to each of these square regions and one edge linking vertices R_x and R_y of M for each wire that must be run in the embedding to connect a port of the square region R_x with a port of the square region R_y. Since the number of ports per unit area in the pseudo-layer is at most

$$3b \frac{F/2^{i/2}}{S_i^2} = \frac{3b}{h^2} \frac{2^{i/2} H^2}{F \log^2 (N/F)},$$

the maximum vertex-degree of M does not exceed

$$D = \frac{12b}{h} \frac{H}{\log (N/F)}$$

(= the number of ports per square region). Recall that (by Lemma 2.3) M can be edge-colored using at most $3D/2$ colors. Therefore, provided only that the constant h is chosen sufficiently large ($h > 18bd$ suffices), it is now an easy matter to allocate wires to layers: we use each layer $H_{i+2} + k$ of the embedding to route all wires that correspond to edges of M that received the color k.

Intra-layer routing. The second phase of the routing gets each wire to the appropriate row and column of its assigned layer. This is a two-dimensional problem consisting of routing $S_i/4$ wires in a square grid of side S_i. In the absence of further information, this might be an impossible task, since the endpoints of the wires to be routed might

be configured in a way that did not afford enough room to route the wires. In our case, however, we have distributed the wires' end-points sufficiently sparsely that the routing is guaranteed to be possible: at most four wires terminate in each square region of area $4S_i$. We have the luxury, therefore, to assign dedicated rows and columns to the wires to be routed. We leave to the reader the details of verifying that this phase of the routing can be accomplished. Note that this routing phase completes the processing of wires that correspond to edges of G_i: all connections are made at one of the levels $H_{i+2}+k$.

Port connections. The final phase of the routing connects those wires that correspond to edges having an endpoint outside of G_i to one of the ports at level H_i. This, however, is a triviality, since wires are already in the appropriate row and column, and there is no contention for the interlayer route that the wire must traverse.

The costs of the layout. It remains to assess the efficiency of our embedding. If we apply Lemma 2.2 carefully (as do Bhatt and Leighton [2], [13] when treating two-dimensional layouts), we find that we can always force the edges in a fully balanced decomposition tree of a graph having an $(F, 2^{1/2})$-bifurcator to stay in the top $c \log (N/F)$ levels of the tree, for some appropriate constant c. We find thereby that if we have chosen the constant d in the height-recurrence judiciously (it suffices that $d > c$), then the recurrence for the height H_0 of the final layout solves to $H_0 \leqq H$. The area of the base of the layout never changes throughout the construction: it is always

$$S_0^2 = \left[h \frac{F}{H} \log (N/F) \right]^2.$$

Since the Volume of the layout is just $H_0 S_0^2$, we have established the claimed upper bound on $\mathrm{VOL}_{1-AL}(G; H)$.

With regard to Wire-Length, it is straightforward to verify that the longest path an uninterrupted wire is stretched over, is proportional to the sum of the linear dimensions of the layout, namely $(\text{const})(S_0 + H_0)$, whence the claimed bound.

Finally, we remark that no appreciable further decrease in Volume can be obtained by further increasing the height: when H assumes its maximum value, the area of the base of the layout is just some small constant multiple of N. Since all of G's N vertices must reside on a single layer, this area cannot be decreased further, so subsequent increases in the height can only increase the Volume.

Computation time. The only part of the described layout procedure that cannot clearly be done in polynomial time is the generation of an $(F, 2^{1/2})$-decomposition tree for G. And, we assume that we are given such a tree as input to the layout procedure. As an equally efficient alternative to our being given the decomposition tree, we could be given a two-dimensional layout for G as our starting point. We expand on this momentarily. □

Theorem 3.1 affords us the following strengthened version of Rosenberg's [24] results about arbitrary graphs.

COROLLARY 3.2. *For any N-vertex graph G and any height $H \leqq hN^{1/2}$,*

$$\mathrm{VOL}_{1-AL}(G; H) \leqq (\text{const}) \frac{N^2}{H}$$

and

$$\mathrm{WL}_{1-AL}(G; H) \leqq (\text{const}) \frac{N}{H}.$$

At most $(\text{const})N^{1/2}$ *layers are needed to minimize* VOL_{1-AL}. *Constructions achieving these results can always be found in time polynomial in N.*

Proof. The worst case in Theorem 3.1 is when $F = N/2$, whence the claimed bounds. In this case, the bifurcator is trivial, and the recursive construction has just one level. □

By judiciously combining two-dimensional layout results with Theorem 3.1, it is not difficult to derive the following AREA-VOL_{1-AL} tradeoff.

THEOREM 3.3. The One-Active-Layer Area-Volume Tradeoff. *Let G be an N-vertex graph, and let* $A = \text{AREA}(G)$.

Height-H layouts. There is a constant $h > 0$ *such that for any height H in the range* $1 \leqq H \leqq h(A/N)^{1/2} \operatorname{loh}(N^2/A)$,

$$\max\left((NA)^{1/2}, \frac{A}{H}\right) \leqq \text{VOL}_{1-AL}(G; H) \leqq (\text{const})\frac{A}{H}\log^2(N^2/A);$$

and

$$(\text{const})\max\left(\frac{A^{1/2}}{N^{1/2}\log(N^2/A)}, \frac{A}{HN\log^2(N^2/A)}\right) \leqq \text{WL}_{1-AL}(G; H) \leqq (\text{const})\frac{A^{1/2}}{H}\log(N^2/A).$$

Unrestricted-height layouts.

$$(NA)^{1/2} \leqq \text{VOL}_{1-AL}(G) \leqq (\text{const})(NA)^{1/2}\log(N^2/A)$$

and

$$(\text{const})\frac{A^{1/2}}{N^{1/2}\log(N^2/A)} \leqq \text{WL}_{1-AL}(G) \leqq (\text{const})N^{1/2}.$$

Moreover, the value of H that minimizes VOL_{1-AL} *is at most*

$$(\text{const})\left(\frac{A}{N}\right)^{1/2}\log(N^2/A).$$

Finally, given the Area-A layout of G, the embeddings yielding the upper bounds can be found in time polynomial in N.

Proof.

The lower bounds. The lower bounds follow from the lower bound arguments of Theorem 3.1 and the fact [2], [13] that $A \leqq F^2\log^2(N^2/A)$.

The upper bounds. As in Theorem 3.1, we can establish both the restricted-height and unrestricted-height upper bounds simultaneously, since the latter bound follows from the former by merely plugging in the maximum permissible value for *H*. We obtain the restricted-height upper bound in stages. First we recall from Theorem 3.1 that

$$\text{VOL}_{1-AL}(G; H) \leqq (\text{const})\frac{F^2}{H}\log^2(N/F),$$

and

$$\text{WL}_{1-AL}(G; H) \leqq (\text{const})\frac{F}{H}\log(N/F).$$

We next note from Theorem 2.1 that

$$F^2 \leqq A.$$

Finally, we claim that $1/F < N/A$ so that

$$\log(N/F) < \log(N^2/A),$$

which completes the proof of the upper bound. This final claim is the culmination of the following sequence of inequalities, each following from its predecessors and/or Theorem 2.1.

For $x > 1$, $x < 2^x$; hence, $\log(N/F) < N/F$, so $F \log(N/F) < N$. By Theorem 2.1, then, $A < FN$, whence the claim.

The efficiency of actually computing the embeddings that yield the upper bounds follows as in Theorem 3.1, once one performs a recursive decomposition of G by cutting the two-dimensional layout recursively along the lines of the proof of the lower bound in Theorem 3.1. □

It is worth noting that the upper bounds in Theorem 3.1 are everywhere existentially tight (within constant factors) for every value of N, F, and H; i.e., the factors of $\log(N/F)$ cannot be avoided. To verify this, one needs to recall that Leighton [13] proved that the upper bounds in Theorem 2.1 are everywhere existentially tight:

For all N and F, there exist N-vertex graphs G whose smallest $2^{1/2}$-bifurcators have size F such that

$$c_1[F \log(N/F)]^2 \leqq \text{AREA}(G) \leqq c_2[F \log(N/F)]^2.$$

For any one of these maximal-AREA graphs G, the lower bounds of Theorem 3.3 assure us that $\text{VOL}_{1-AL}(G; H)$ is no smaller than some constant multiple of

$$\max\left(N^{1/2} F \log(N/F), \frac{F^2}{H} \log^2(N/F)\right).$$

This information combines with the upper bounds of Theorem 3.1 to establish the claimed tightness. We do not know that the upper bounds of Theorem 3.3 are similarly tight; indeed, we conjecture that they are not.

CONJECTURE 3.4. The One-Active-Layer Area-Volume Tradeoff. *Let G be an N-vertex graph, and let $A = \text{AREA}(G)$. There is a constant $h > 0$ such that, for any height H in the range*

$$1 \leqq H \leqq h(NA)^{1/2},$$

we have

$$\text{VOL}_{1-AL}(G; H) = (\text{const}) \frac{A}{H}.$$

For larger H, no additional decrease in Volume can be obtained.

3.2. Unrestricted layouts. We turn now to the task of proving analogs of Theorems 3.1 and 3.3 for many-active-layer three-dimensional layouts. We shall be less thorough in our pursuit of these analogs, for the following reasons:

1. The major ideas required to obtain compact three-dimensional embeddings appear already in the one-active-layer case, which we have looked at in great detail.

2. While the one-active-layer case can already be considered to have been realized (say, in IBM's TCM [10], [22]), it is not yet clear that many-active-layer three-dimensional layouts will ever be more than a theoretical construct.

3. The many-active-layer results that we develop suggest that only relatively minor gains are achieved by abjuring the one-active-layer restriction.

4. The technical details of obtaining height-restricted many-active-layer layouts are substantial and may not be worth the gains over the one-active-layer case.

The major change in layout strategy in the many-active-layer model is that we must use $2^{2/3}$-bifurcators in order to obtain compact layouts. Indeed, the feature that renders restricted-height layouts prohibitively complicated with the many-active-layer model is that one must play off $2^{1/2}$-bifurcators against $2^{2/3}$-bifurcators. We satisfy ourselves, therefore, with the following many-active-layer results.

THEOREM 3.5. The General Three-Dimensional Layout Theorem. *Let G be an N-vertex graph, let F be the size of its minimum $2^{2/3}$-bifurcator, and let $A = \text{AREA}(G)$. The many-active-layer three-dimensional layouts of G satisfy*

$$F^{3/2} \leqq \text{VOL}(G) \leqq (\text{const})[F \log (N/F)]^{3/2}$$

and

$$(\text{const}) \frac{F^{3/2}}{N} \leqq \text{WL}_3(G) \leqq (\text{const})[F \log (N/F)]^{1/2}.$$

Proof. We establish the lower and upper bounds in turn.

The lower bounds. Let G be laid out in an $H \times W \times L$ grid, where with no loss of generality, $H \leqq W \leqq L$.

We establish the lower bound on Volume by recursively bisecting the layout of G, much as we did in the lower-bound proof of Theorem 3.1. We slice the $H \times W \times L$ grid holding the layout into two $H \times W \times (L/2)$ grids, choosing purposely to bisect the biggest of the dimensions. We then recursively continue this bisecting, each time halving the biggest dimension of the grid being sliced. Now, at stage i of this procedure, we are bisecting boxes of volume $HWL/2^i$ (we started at stage 0). When slicing each such box, the plane of the slice has area at most $(HWL/2^i)^{2/3}$—the longest dimension is at least $(HWL/2^i)^{1/3}$, leaving only the indicated area for the plane. Since wires have all unit cross-sections, each slice cuts no more than $(HWL/2^i)^{2/3}$ wires. Since G can be so bisected recursively, and since we have been looking at an arbitrary three-dimensional layout of G, it follows that G has a $2^{2/3}$-bifurcator of size $(\text{VOL}(G))^{2/3}$. Since this bifurcator is at least as big as G's minimum such bifurcator F, we have thus shown that

$$F^{3/2} \leqq \text{VOL}(G).$$

The lower bound on Wire-Length is more complicated. We shall establish it by showing that the total volume of wire, $\omega(G)$, used to lay out G (i.e., the sum of the lengths of all G's edges) is at least $(\text{const})F^{3/2}$. This bound on $\omega(G)$ will yield the desired bound on $\text{WL}_3(G)$ since G, having degree at most four, has no more than $2N$ edges; hence, the average (all the moreso the maximum) length of a run of wire in the layout is at least $(\text{const})\omega(G)/N$.

To show that $\omega(G) \geqq (\text{const})F^{3/2}$, it suffices to show that there exists a fraction $0 < q < 1$ and a constant $r > 0$ such that one can partition any three-dimensional layout having wire-volume ω into two sublayouts, each with wire-volume at most $q\omega$, by removing at most $r\omega^{2/3}$ edges: by iterating such a partitioning, we could partition the original layout into two sublayouts each having wire-volume $\omega/2$ by removing at most $(\text{const})\omega^{2/3}$ edges. We could then repeat this latter partitioning recursively to construct a $2^{2/3}$-bifurcator for G of size $(\text{const})\omega^{2/3}$. It would thence follow (since F is the size of G's smallest $2^{2/3}$-bifurcator) that $\omega \geqq (\text{const})F^{2/3}$, as desired.

So, we set ourselves the goal of partitioning G's layout. Say that we are presented with appropriately chosen constants q and r and with an arbitrary three-dimensional

layout with wire-volume ω. Find that horizontal layer of the grid that partitions the layout into two pieces, each of wire-volume no greater than $q\omega$; call this the *middle layer* of the layout. Say first that the middle layer has passing through it at most $r\omega^{2/3}$ connections with the rest of the layout. Then we are done, since removing all wires that touch the middle layer effects the desired partition. Otherwise, if the middle layer touches too many wires, then find both the lowest layer above the middle layer and the highest layer below the middle layer that each contain at most $r\omega^{2/3}$ connections with the rest of the layout; call the former the *upper layer* and the latter the *lower layer* of the layout. Remove the upper and lower layers, thereby partitioning the layout into *upper*, *middle*, and *lower* regions. We consider two cases. If either the upper or the lower region contains at least $q\omega$ wire-volume, then we are done, since we shall choose as the two parts of our partition that "packed" region plus the union of the other two regions. If both the upper and lower regions are sparse, then most of the wire in the layout resides in the middle region, so it suffices to partition only that region. Since each layer of the middle region contains or is incident to more than $r\omega^{2/3}$ unit segments of wire, there can be no more than $(1/r)\omega^{1/3}$ layers in the region (or else there would be too much total wire). An analysis symmetric to the "horizontal" one we have been considering proves that the length and width of this region are also bounded above by $r\omega^{1/3}$. Hence, the entire layout volume of the middle region is at most $(\text{const})\omega$. By the earlier cutting plane argument, then, the middle region can be partitioned by removing at most $r\omega^{2/3}$ wires, provided only that r is chosen appropriately.

The upper bounds. As with Theorem 3.1, the upper bounds here are significantly more complicated to establish than the lower bounds. Once again, our proof is via an explicit inductive construction. In this case, the construction will lay out a subgraph G_i of G that resides at level i of a $2^{2/3}$-decomposition tree of G, by combining the layouts of the eight subgraphs of G_i that reside at level $i+3$ of the tree.

Let us assume: (1) that we are given layouts of the eight level-$(i+3)$ subgraphs, in "boxes" of dimensions $H_{i+3} \times W_{i+3} \times L_{i+3}$ each; (2) that the boxes are all similarly oriented in space (so we can talk about their fronts and tops, etc.); (3) that each of the boxes has on its front face a *connection grill*, which, letting

$$F_{i+3} =_{\text{def}} \frac{F}{(2^{i+3})^{2/3}},$$

is a set of

$$b\frac{F_{i+3}}{H_{i+3}}$$

length-H_{i+3} columns, spread evenly across the width $= W_{i+3}$ front face, collectively comprising the bF_{i+3} ports through which we shall wire these subgraphs to each other and to the remainder of G; see Fig. 5. As in the proof of Theorem 3.1, the constant b is derived from Lemma 2.2.

As the first step in laying out G_i, we place our eight boxes in a big box of dimensions $H_i \times W_i \times L_i$, where

$$H_i = 2H_{i+3},$$

$$W_i = 2W_{i+3} + 2b\frac{F_{i+3}}{H_{i+3}},$$

$$L_i = 2L_{i+3} + 3b\frac{F_{i+3}}{H_{i+3}} + 4c\frac{F_i}{W_i}$$

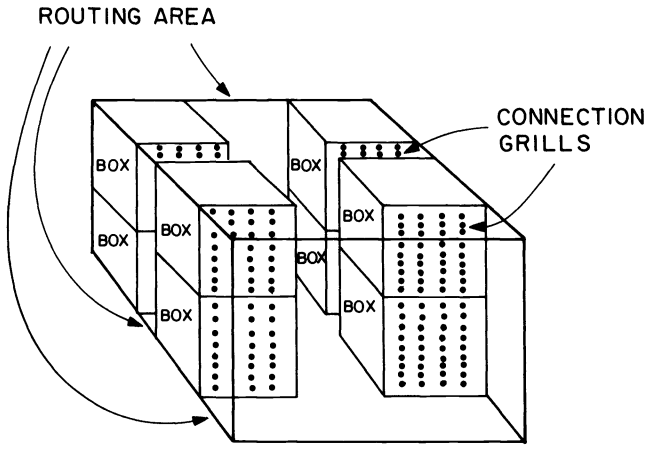


FIG. 5. *Coalescing the many-active-layer layouts of the subgraphs of* G_i.

for a sufficiently large constant c. We place the small boxes as follows. Four of the boxes go at the four corners of the back of the big box. In front of these boxes we reserve bF_{i+3}/H_{i+3} empty layers that will be used for wire-routing. In front of these empty layers, we place the four remaining small boxes, in the corners. In front of these boxes we reserve $2bF_{i+3}/H_{i+3}+4cF_i/W_i$ empty layers for wire-routing. We complete the placement phase by reserving $2bF_{i+3}/H_{i+3}$ empty layers between the left and right tiers of small boxes. Finally, on the front face of the big box, we uniformly spread out a new connection grill with bF_i/H_i columns of H_i ports each, containing the bF_i ports that are guaranteed to be sufficient to wire G_i up to the rest of G. See Fig. 5.

Now we turn to the task of routing the wires that leave the level-$(i+3)$ graphs (through their connection grills), both among each other and to the new connection grill. As the first step of this routing, we use the bF_{i+3}/H_{i+3} empty layers in front of the four rear boxes to route the wires from these boxes' connection grills into one big $[H_i \times 2(bF_{i+3}/H_{i+3})]$ rectangular *connection patch* in between and in front of the four rear boxes (a grill is spread out, while a patch is compressed); this can be accomplished by having the innermost columns of the grills move in on one layer to meet one another, the next-to-innermost move in on the next layer to be adjacent to the innermost ones, and so on; see Fig. 6. The next step is to run the connection patch, which occupies the $2bF_{i+3}/H_{i+3}$ routing layers between the left and right tiers of boxes, to the empty layers that we have placed front of all the boxes (also depicted in Fig. 6). Now we take the big connection patch we have just run from the back of the layout, and the small connection grills on the front tier of small boxes, and we use $2bF_{i+3}/H_{i+3}$ of the empty layers at the front of the box to distribute these columns of ports so that they are evenly spaced across the front of the layout, i.e., so that they become a connection grill. (There are $4bF_{i+3}/H_{i+3}$ columns to be distributed across a face of width W_i, so the columns are spaced with

$$\frac{W_i H_{i+3}}{4bF_{i+3}}$$

empty columns intervening between full columns.) As the next-to-final step, we assign the ports of this new connection grill to the layers on which they will be routed to their appropriate rows and columns. This assignment is done using the same device as in Theorem 3.1. We first superimpose the front face, with its (bF_i/H_i)-column

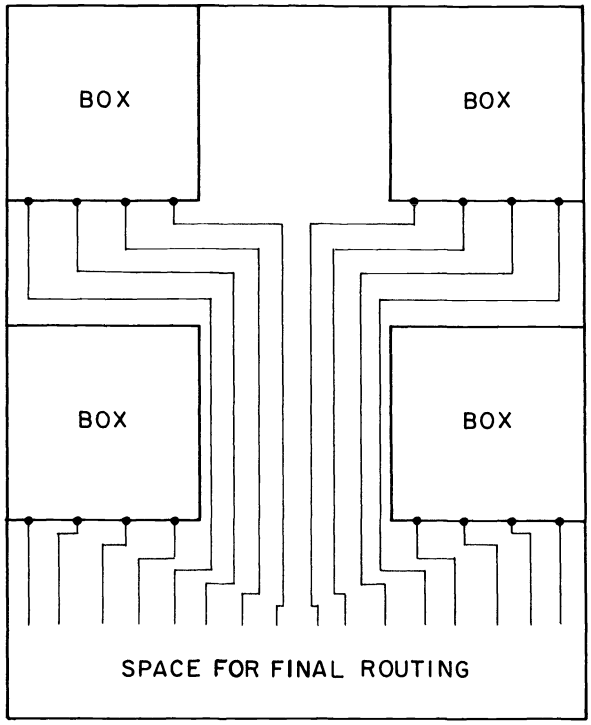


FIG. 6. *A view from above of the coalescing of the wires from the subgraphs.*

connection patch, on the just constructed $(4bF_{i+3}/H_{i+3})$-column connection patch. We then partition the superimposed faces into rectangles of height cF_i/W_i and width $(W_iH_{i+3})/(4bF_{i+3})$. By design, each of these rectangles contains no more than $2cF_i/W_i$ ports. Hence, when we follow our ploy from Theorem 3.1 and form the multigraph M from the partition, and edge-color M, we are assured by Lemma 2.3 that we need use no more than $3cF_i/W_i$ colors. Hence, when we assign wires to layers by their colors, as in Theorem 3.1, we need use no more than $3cF_i/W_i$ layers. Provided that $bW_i \leqq cH_{i+3}$ and $2b \leqq c$, it is an easy matter from this point to complete the routing using the as-yet-unused reserved routing layers, once having assigned layers. We leave details to the reader.

It remains to assess the efficiency of the layouts produced by the preceding construction. If we arbitrarily set

$$H = H_0 = h[F \log (N/F)]^{1/2},$$

which is acceptable provided that the constant h is chosen judiciously, then we find that the recurrences for length and width solve to

$$W = W_0 = (\text{const})[F \log (N/F)]^{1/2},$$

and

$$L = L_0 = (\text{const})[F \log (N/F)]^{1/2}.$$

(In solving these recurrences, one must keep in mind that F is the size of G's smallest $2^{2/3}$-bifurcator.) The claimed upper bound on Volume follows by just multiplying these linear dimensions; the upper bound on Wire-Length follows by summing them, for

our wire-routing scheme requires a wire to traverse each linear dimension only a small number of times. □

The bounds of Theorem 3.5 can be shown to be existentially tight by generalizing the methods and networks of [2], [13] to three dimensions.

As with one-active-layer embeddings, we can derive from our general layout theorem an Area-Volume tradeoff.

THEOREM 3.6. The General Area-Volume Tradeoff. *Let G be an N-vertex graph, and let $A = \text{AREA}(G)$. The many-active-layer three-dimensional layouts of G satisfy*

$$\max(N, (1/4)A^{3/4}) \leqq \text{VOL}(G) \leqq (\text{const})(NA)^{1/2}$$

and

$$\text{WL}_3(G) \leqq (\text{const})(NA)^{1/6}.$$

Proof sketch.

The lower bounds. The lower bounds follow by previously enunciated principles: the vertices of G alone, ignoring wires, consume volume N; and any $H \times W \times L$ layout of G (where $H \leqq L$, W) can be transformed into a $3WH \times 3LH$ two-dimensional layout for G, whence $\text{AREA}(G) \leqq 9\text{VOL}(G)^{4/3}$.

The upper bounds. The upper bounds follow from the construction of Theorem 3.5 and the fact that any graph admitting an Area-A layout has a $2^{1/2}$-bifurcator of size $A^{1/2}$, hence a $2^{2/3}$-bifurcator of size $(NA)^{1/3}$. If we plug this $2^{2/3}$-bifurcator into the upper bound of Theorem 3.5, we find that

$$\text{VOL}(G) \leqq (\text{const})(NA)^{1/2}\log^{3/2}(N^2/A)$$

and

$$\text{WL}_3(G) \leqq (\text{const})(NA)^{1/6}\log^{1/2}(N^2/A).$$

A careful analysis of the layout of G hidden in this upper bound indicates that we are really cutting more edges of G at each step than a smaller $2^{2/3}$-bifurcator would force us to (since our bound on F is very conservative). Hence, when we calculate in detail the dimensions of the layout produced with this big bifurcator, we find that we actually avoid the logarithmic factor, and we obtain the bounds of Theorem 3.6. □

4. Conclusions. The work in this paper leaves the reader with a number of messages, which we now encapsulate.

* Three-dimensional layouts can be appreciably more conservative of resources, both material and wire-length, than can two-dimensional layouts.

* Three-dimensional layouts are not appreciably harder to "compute" than are two-dimensional layouts; in fact the former can be produced from the latter via polynomial-time algorithms.

* For many classes of graphs, one-active-layer three-dimensional layouts are as efficient as many-active-layer layouts. In general the best bounds that can be proved in terms of A (optimal two-dimensional Area) and N (number of vertices) for the two classes of layouts differ by at most a logarithmic factor. Thus no general layout procedure is uniformly better than our one-active-layer layout procedure. If this phenomenon occurs also in practice, then the value of fabricating transistors on multiple levels is limited.

* Even in the one-active-layer model, only a limited number of layers are helpful. Roughly speaking, only $(A/N)^{1/2}$ or $F/N^{1/2}$ layers lead to increased efficiency of multilayer layouts: additional layers cannot further decrease Volume. It is worth noting

that the quantity $(A/N)^{1/2}$ is closely related to the complexity of two-dimensional placement and routing: this is the average channel width in a two-dimensional layout of the circuit. Although we did not prove it in this paper, we suspect that additional layers are also not useful in decreasing wire-length. We know this to be the case for many families of graphs.

* Although we have not paid attention to issues like the sizes of constants here, it seems likely that the method of layer assignment that we employed in the upper-bound proof of Theorem 3.1 can be adapted to produce computationally efficient assignments in practical situations.

REFERENCES

[1] V. E. BENES (1964), *Optimal rearrangeable multistage connecting networks*, Bell Syst. Tech. J., 43, pp. 1641-1656.

[2] S. N. BHATT AND F. T. LEIGHTON (1984), *A framework for solving* VLSI *graph layout problems*, J. Comp. Syst. Sci., 28, pp. 300-343.

[3] M. L. BRADY AND D. J. BROWN (1984), VLSI *routing: four layers suffice*, Advances in Computing Research 2, F. P. Preparata, ed., JAI Press, Greenwich, CT, pp. 245-257.

[4] R. D. ETCHELLS, A. D. CUMMINGS, J. GRINBERG AND G. R. NUDD (1982), *Cellular array architecture for microelectronic implementation*, Typescript.

[5] ———, (1982), *A yield-redundancy policy for wafer-scale integration*, Typescript.

[6] ———, (1982), *Power dissipation in* 3-D VLSI, Typescript.

[7] R. D. ETCHELLS, J. GRINBERG AND G. R. NUDD (1981), *Development of a three-dimensional circuit integration technology and computer architecture*, Soc. Photographic and Instrumentation Engineers, 282, pp. 64-72.

[8] J. F. GIBBONS (1982), *SOI—a candidate for* VLSI? VLSI Design III, pp. 54-55.

[9] L. S. HEATH (1983), *Multilayer channel routing*, MCNC Tech. Rpt. 83-06.

[10] C. W. HO (1982), *High performance* VLSI *computer packaging*, 1982 MIT Conference on Advanced Research in VLSI, p. 42.

[11] HUGHES RESEARCH LABORATORIES (1982), *A cellular* VLSI *architecture for image analysis and two-dimensional signal processing*, Typescript.

[12] H. W. LAM, A. F. TASCH, JR. AND T. C. HOLLOWAY (1980), *Characteristics of* MOSFETs *fabricated in laser-recrystallized polysilicon islands with a retaining wall structure on an insulating substrate*, Electron Dev. Let., EDL-1 (1980), pp. 206-208.

[13] F. T. LEIGHTON (1982), *A layout strategy for* VLSI *which is provably good*, 14th ACM Symposium on Theory of Computing, pp. 85-98.

[14] ———, (1983), *Complexity Issues in* VLSI: *Optimal Layouts for the Shuffle-Exchange Graphs and other Networks*, MIT Press, Cambridge, MA.

[15] F. T. LEIGHTON AND A. L. ROSENBERG (1983), *Automatic generation of three-dimensional circuit layouts*, 1983 IEEE International Conference on Computer Design: VSLI in Computers, pp. 633-636.

[16] C. E. LEISERSON (1983), *Area-Efficient* VLSI *Computation*, MIT Press, Cambridge, MA.

[17] D. B. LENAT, W. R. SUTHERLAND AND J. GIBBONS (1982), *Heuristic search for new microcircuit structures: An application of artificial intelligence*, The AI Magazine, 3, pp. 17-33.

[18] W. R. LOCKE (1983), *Three-dimensional integration: A critical survey*, MCNC Tech. Rpt. 83-06.

[19] E. W. MABY AND D. A. ANTONIADIS (1981), *Device structures for three-dimensional integration*, Lecture at MIT Research Review, December, 1981.

[20] F. P. PREPARATA (1981), *Optimal three-dimensional* VLSI *layouts*, Math. Systems Theory, 16, pp. 1-8.

[21] F. P. PREPARATA AND W. LIPSKI (1982), *Optimal three-layer channel routing*, 23rd IEEE Symposium on Foundations of Computer Science, pp. 350-357.

[22] D. ROSEN (1981), TCM—*it's a new word for density in logic packaging*, THINK, p. 23.

[23] A. L. ROSENBERG (1981), *Three-dimensional integrated circuitry*, in VLSI Systems and Computations, H. T. Kung, B. Sproull and G. Steele, eds., Computer Science Press, Rockville, MD, pp. 69-80.

[24] ———, (1983), *Three-dimensional* VLSI: *A case study*, J. ACM, 30, pp. 397-416.

[25] C. E. SHANNON (1949), *A theorem on coloring the lines of a network*, J. Math. Physics, 28, pp. 148-151.

[26] C. D. THOMPSON (1980), *A complexity theory for* VLSI, Ph.D. Thesis, Carnegie-Mellon University; see also Area-time complexity for VLSI, 11th ACM Symposium on Theory of Computing, 1979, pp. 81-88.

[27] L. G. VALIANT (1981), *Universality considerations in* VLSI *circuits*, IEEE Trans. Comp., C-30, pp. 135–140.

[28] Z. A. WEINBERG (1981), *Polysilicon recrystallization by* CO_2 *laser heating of* SiO_2, IBM Report RC-8835.

[29] D. S. WISE (1981), *Compact layouts of banyan/*FFT *networks*, VLSI Systems and Computations, H. T. Kung, B. Sproull and G. Steele, eds., Computer Science Press, Rockville, MD, pp. 186–195.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
013

PARALLEL ALGORITHMS FOR DEPTH-FIRST SEARCHES I. PLANAR GRAPHS*

JUSTIN R. SMITH†

Abstract. This paper presents an unbounded-parallel algorithm for performing a depth-first search of a planar undirected graph. The algorithm uses $O(n^4)$ processors and executes in $O(\log^3 n)$-time. It had previously been conjectured that the problem of computing a depth-first spanning tree was *inherently sequential.*

Key words. parallel algorithms, depth-first search, computational complexity, inherently sequential problems

Introduction. Let $G=(V, E)$ be an undirected connected graph. A *depth-first spanning tree*, T, of G is a spanning tree constructed as follows:

Starting at the root (some selected vertex of G) successively add edges to T until a vertex is reached all of whose exit edges (i.e. incident edges other than the one used to arrive at the vertex) are incident upon T. At this point backtrack a minimal distance until a vertex is reached that has exit edges not incident upon T and continue. Since the backtracking is allowed, it is clear that the process can continue until all of the vertices of G are exhausted.

Depth-first spanning trees of graphs have, in the past, been used as a basis for many other graph-theoretic algorithms—see [14] for a survey. More recently they have played an important part in the implementation of certain artificial intelligence languages like PROLOG.

Note that, if there are *several* exit edges from a vertex, more of which are incident upon T, we may arbitrarily choose the edge to be added to T next. The choice will have a drastic effect upon the later stages of the construction above, however. Because of this it has been conjectured (see [13]) that the construction of a depth-first spanning tree of a graph is *inherently sequential*—i.e. it cannot be performed in poly-logarithmic time with a polynomial number of processors. This is related to the question of whether the two classes of problems P and NC are the same—where P is the class of problems solvable in polynomial time and NC is the class of problems solvable in poly-logarithmic time in *parallel* with a polynomial number of processors. See [2, § 6].

The main result of the present paper is:

THEOREM (DFST). *If G is an undirected planar connected graph with n vertices, there exists a parallel algorithm for constructing a depth-first spanning tree of G that executes in $O(\log^3 n)$-time using $O(n^4)$ processors.*

Remarks. 1. We use the unbounded-parallelism model of computation here—this corresponds to the SIMD model in [4]. We assume that any number of processors can *read* a given memory location simultaneously but only one processor can *write* to a memory location at a time (i.e. the result of more than one processor writing to one location at a time is *undefined*). We also assume that each processor has a unique number that can be referred to in instructions that the computer executes.

2. It is straightforward to go from the depth-first spanning tree constructed by DFST above to a *depth-first search* of G. See Appendix A.

* Received by the editors December 20, 1983, and in final revised form July 10, 1985.

† Department of Mathematics and Computer Science, Drexel University, Philadelphia, Pennsylvania 19104.

3. Although a parallel algorithm for depth-first spanning trees was discovered by Eckstein and Alton (see [3])—its execution time was *linear* in the number of vertices so that it was not in the class NC like the algorithm presented in *this* paper.

4. The results of Eli Upfal in [16] imply that the algorithm presented here can be converted to a *probabilistic* (in the sense of execution time) version of the DFST algorithm that executes in $O(\log^5 n)$ expected time on an *ultracomputer* (a set of processors communicating through a bounded-degree network, with no common memory).

The actual construction of the depth-first spanning tree makes crucial use of the following result, which may be of independent interest:

THEOREM (PARTITION). *Let G be a planar, biconnected graph (equipped with an imbedding into the plane) with n vertices. Then there exists a parallel algorithm for finding a simple cycle in G with the property that its interior and exterior subgraphs have $\leqq 2n/3$ vertices. This algorithm executes in $O(\log^2 n)$-time using $O(n^4)$ processors.*

Remarks. 1. In general, the simple cycle produced by this algorithm will depend upon the planar imbedding of G. Section 1 of this paper describes how the PARTITION algorithm can be used to find a depth-first spanning tree for G, and § 2 describes the algorithm itself.

2. This is a generalization of Lemma 2 of Lipton and Tarjan (see [8] and [9]). That result was *sequential* and required that all of the mesh cycles of the graph be *triangular.*

3. Gary Miller recently proved a sequential version of the PARTITION result—it executes in *linear time* (thus it is superior to what one would get by making the PARTITION algorithm into a sequential one in the straightforward way) (see [10]).

Future installments of this paper will explore applications of these and generalizations to nonplanar graphs.

All graphs in this paper will be assumed to be *undirected.*

1. Proof of DFST using PARTITION. It is first necessary to recall a well-known characterization of depth-first spanning trees.

DEFINITION 1.1. Let T be a spanning tree of a graph G. Define the following partial order on the vertices of G:

$$v_1 < v_2 \text{ if } v_2 \text{ is an ancestor of } v_1 \text{ in } T.$$

This partial order will be called the *partial order induced by T.*

Remarks. In general, if v_1 and v_2 are arbitrary vertices of G they will not be comparable in this partial order, i.e. neither vertex will be an ancestor of the other.

The following result can be found in [14]:

LEMMA 1.2. *A spanning tree T in a graph G is a depth-first spanning tree if and only if for each edge in G the end vertices of the edge are comparable in the partial order induced by T.*

Now we will discuss a preliminary decomposition of a graph that must be performed before the main operations of the DFST algorithm can be carried out.

DEFINITION 1.3. A connected graph G is defined to be *biconnected* if it cannot be separated into more than one component by the deletion of one vertex. If G is not biconnected, the vertices whose deletion separates G are called *articulation points* and the components into which G separates (together with the articulation points connected to them via edges) are called biconnected components of G.

The characterization of depth-first spanning trees given by 1.3 implies that:

PROPOSITION 1.4. *Let G be a connected graph with biconnected components G_i, and with a selected vertex x. For each biconnected component G_i let (x_i) be the articulation point closest to x. If T_i is a depth-first spanning tree of G_i rooted at x_i then the union of the T_i is a depth-first spanning tree of G rooted at x.*

Remark. It is not difficult to see that *all* depth-first spanning trees of G can be gotten by taking the union of depth-first spanning trees of its biconnected components.

DEFINITION 1.5. Let G be a graph and suppose that P is a simple path in G with a distinguished end vertex called its root such that $G\backslash P$ is a disjoint union of components $\{G_i\}$. If e is an edge in G_i define e to:

a. *touch* P is precisely one end vertex of e is in P—call this vertex the *point which e touches P*;

b. be *inessential for* G_i if it touches P and there exists an edge e' in G_i that touches P at a point further (*in* P) from the root than the point at which e touches P;

c. be *essential for* G_i if it touches P and the condition of statement *b is not satisfied.*

Remark. We are using the standard notation $A\backslash B$ to denote the *complement of B in A*, or the *set difference.*

DEFINITION 1.6. Assume the conditions of Definitions 1.5 above and suppose G' is the result of deleting all of the inessential edges (for the components of $G\backslash P$) from G (and not deleting their end vertices). Then G' is defined to be *the reduction of G by P, or the result of reducing G by P.*

Remarks. The result of reducing G by P will consist of the union of:

i. P;

ii. the essential edges of G;

iii. the components $\{G_i\}$ of the graph that results from deleting all of the vertices of P from G.

This graph *will not be biconnected*—the points at which the essential edges of the G_i touch P will be *articulation points.*

The following result will be the basis of the proof of the validity of the DFST algorithm.

PROPOSITION 1.7. *Assume the conditions of Definition* 1.5 *and the remarks following Definition* 1.6. *Let e_i be an essential edge for G_i connecting G_i to P and let T_i be a depth-first spanning tree of G_i rooted at the end vertex of e_i that is in G_i. Then the union of P, the $\{e_i\}$ and the $\{T_i\}$ is a depth-first spanning tree of G.*

Proof. We will verify the conclusions of Lemma 1.2. Let T denote the union described in the conclusion of the proposition. Clearly if an edge connects two vertices of a T_i its end vertices will be comparable in the partial ordering induced by T because the T_i are depth-first spanning trees of the G_i. If an edge connects a T_i to P, it will either be *inessential* or *essential* for G_i. If it is inessential, then its end vertices are comparable because all of the vertices of the T_i are descendants of any vertex of P. If it is essential, the same will be true because no edge touches P at a lower point, at least in G_i—i.e. the essential edge in question and the edge used to connect T_i to P will have a *common end vertex* and the *other* end vertex of the essential edge will be *lower* (in the partial ordering in G_i) than the other end of the edge that connects T_i to P. Furthermore, *no* edges can connect a T_i to a T_j with $i \neq j$ since they will be contained in *distinct components* of $G\backslash P$. □

Before we can present the DFST algorithm, it is necessary to make one more definition:

DEFINITION 1.8. Let G be a connected graph. Define:

1. G_s, the *solid subgraph*, to consist of all edges of G that are contained in *some cycle*;

2. G_t, the *treelike subgraph*, to consist of all edges of G that are not in any cycle.

Remarks. 1. If G is a planar graph equipped with an imbedding into the plane, G_s, is clearly the union of all of the mesh cycles.

2. G itself is clearly the union of G_s and G_t.

3. G can be decomposed into G_s and G_t in $O(\log^2 n)$ time using $O(n^2)$ processors. This follows from the determination of a fundamental system of cycles in [1].

We are now in a position to present the main result of this section:

ALGORITHM 1.9 (DFST). *Let G be a connected planar graph and let e be a selected vertex (which will be the root of the depth-first spanning tree). All edges that are selected will appear in the output (i.e. in the depth-first spanning tree).*

1. *Imbed G in the plane—this can be done in $O(\log^2 n)$ time using $O(n^4)$ processors—*[6];

2. *Determine the biconnected components of G—this can be done in $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ processors (see* [15]);

3. *Determine G_s and G_t and select G_t—the remainder of the algorithm is performed upon G_s;*

4. *For each biconnected component, C, do in parallel:*

4.1. *For C_s find a partitioning cycle using the* PARTITION *algorithm described in* § 2 *of the present paper—this requires $O(\log^2 n)$ time using $O(n^4)$ processors.*

4.2. *Find a path connecting the entry vertex to the partitioning cycle. This can be done in $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ processors:*

a. *find a spanning tree using the algorithm of* [1];

b. *direct the spanning tree with the entry vertex as its root (and edges pointing away) using the Euler Tour method of* [15, § 5]*—this requires $O(\log n)$-time using $O(n)$ processors;*

c. *delete all branches of the spanning tree going away from the partitioning cycle—this can be done in $O(\log n)$-time using $O(n)$ processors;*

d. *select a branch of the directed spanning tree coming into the partitioning cycle and propagate a marker backwards (i.e. in the reverse of the directed edges) to the root. This is the path connecting the root to the partioning cycle. This step can be done in $O(\log n)$ time using $O(n)$ processors.*

4.3. *Delete an edge of the partitioning cycle found in* 4.1 *that is adjacent to the point where the path found in* 4.2 *intersects it—this requires unit time with $O(n^2)$ processors—call the resulting path P and select it.*

4.4. *Compute the set $\{C_i\}$ of components of $C_s \setminus P$—this can be done in $O(\log^2 n)$-time using $O(n^2/\log^2 n)$ processors (see* [1]);

4.5. *For each of the C_i found in the previous step compute the set of all edges in C_s that touch P and determine which are essential—see* 1.5. *This requires that the vertices of P be numbered, which requires $O(\log n)$ time using $O(n)$ processors; and requires that the edge with the lowest numbered end vertex be picked—this can be done in $O(\log n)$ time with $O(n^2)$ processors;*

4.6. *Select any one of the essential edges found in* 4.5 *and determine the end vertex that is not in P—call this vertex v;*

4.7. *Apply steps* 3 *and* 4 *of this algorithm to each component using vertex v found in* 4.6 *as the entry vertex.*

Remarks. 1. The proof of the validity of this algorithm follows immediately from 1.7 and induction.

2. Since each of the components found in step 4.4 has $\leqq 2n/3$ vertices, it is clear that this whole process need only be carried out $O(\log n)$ times.

3. The depth-first spanning tree constructed above can be converted to a depth-first *search* (see Appendix A).

4. Note that step 4.2d above requires that *both* orientations of each directed edge be saved—one is marked as the *forward direction* and the other is marked as the *reverse.*

5. Here is an example of this procedure. Consider the graph in Fig. 1.1. The PARTITION algorithm will give a partitioning cycle—after steps 4.2 and 4.3 we will get the graph in Fig. 1.2, where the dark shaded lines represent the edges of the *partioning cycle* (after (2.8) was deleted in step 4.3) and the path connecting the root to the partioning cycle. In the second iteration of steps 2 through 4 of 1.9 we will get Fig. 1.3.

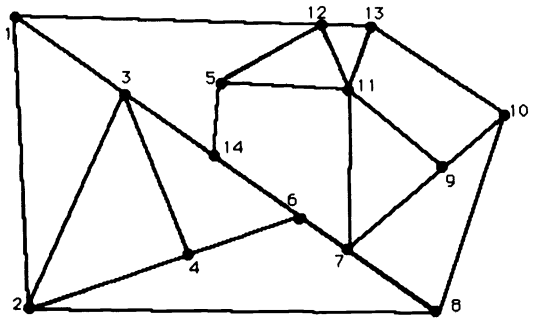


FIG. 1.1

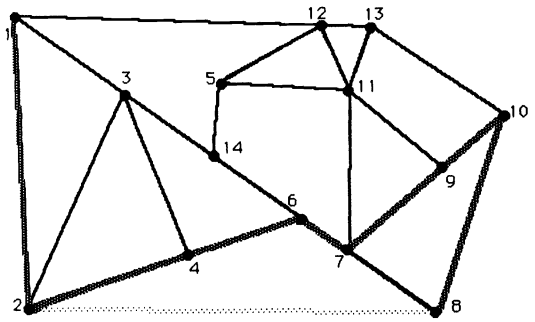


FIG. 1.2

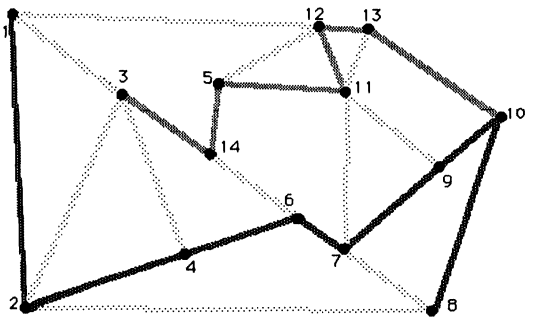


FIG. 1.3

Here the *light-shaded* edges represent edges that were *deleted* and the slightly darker edges resulted from the second iteration of steps 2 through 4 of 1.9. Edge (10, 13) was the essential edge connecting path P to the complementary subgraph. Edge (11, 13) was deleted in the second iteration of 4.3 and edges (9, 11), (7, 11), (1, 12), etc. were *inessential.* We now have the complete depth-first spanning tree.

2. The PARTITION algorithm. In this section we will describe the PARTITION algorithm and prove its validity. In this section G will be assumed to be a planar biconnected graph equipped with an imbedding into the plane. This essentially means that there will be given a decomposition of G into mesh cycles. Throughout this section all statements whose numbers are prefixed with a P will be steps in the PARTITION algorithm.

Here is a brief outline of the initial stages of the PARTITION algorithm:

Essentially, in order to compute the partitioning cycle, we must form unions of mesh cycles of G until we find one whose boundary is a simple cycle that encloses the appropriate number of vertices. The simple-minded way of doing this is sequential (and slow, at that)—in order to carry out the procedure in parallel, it is necessary to compute a large number of such unions simultaneously and to order the partial unions in a way that, at least roughly, corresponds to the number of vertices enclosed by the boundary.

This is accomplished by regarding G as the union of an increasing sequence of subgraphs $G[i]$ (i.e. each term in the sequence is a subgraph of the next term). This must be done in such a way that the terms in the sequence have boundaries that are simple cycles. This last requirement will turn out to be the chief technical problem in constructing the partitioning cycle. Appendix B studies this question and shows that, essentially, a subgraph G' of G that is a union of mesh cycles has a boundary that is a simple cycle if and only if G' and its complement have connected images in the dual of G. Recall that the dual, D, of G is a graph whose vertices correspond to the mesh cycles of G and whose edges are defined in such a way that an edge in D connected two vertices if and only if the corresponding mesh cycles in G have a common edge (see [5, Chap. 11]).

We will construct the sequence $G[i]$ of subgraphs by defining a function on the mesh cycles of G and defining the terms of the sequence to be unions of all mesh cycles on which the value of the function is $\leqq$ some integer. This construction was motivated by the methods of Morse Theory (see [11]).

It turns out that there does not seem to be any easy way to guarantee that the boundaries of the $G[i]$ are *simple cycles*—the most we can accomplish is to construct the $G[i]$ in such a way that they can be *decomposed* into *subgraphs* that have boundaries that are simple cycles (these are the uniform components referred to in Appendix B). Even *this much* makes the construction of the function used to define the $G[i]$ fairly complicated.

The function is defined in the following way:

DEFINITION 2.1. Let G be a planar graph that is equipped with an imbedding into the plane.

1. G is defined to be:
   a. *labeled* if a number $\geqq 0$ is assigned to each vertex;
   b. *filtered with respect to a labeling* if it is labeled and a number $\geqq 0$ is assigned to each mesh cycle equal to the *maximum* of the labels of the vertices of the mesh cycle. The numbers assigned to mesh cycles in a filtration will be called their *filtration indices*;

2. If S is a set of mesh cycles in a graph that is filtered with respect to some labeling, its *closure* is defined to be the subgraph of G spanned by all vertices in the mesh cycles of S.

3. If i is an integer such that $0 \leqq i$ and G is filtered with respect to some labeling, then $G[i]$, the *i-section* of G is defined to be the closure of the set of mesh cycles whose filtration indices are $\leqq i$ and $G\{i\}$, the *residual i-section* of G is defined to be the closure of the set of mesh cycles whose filtration indices are $> i$.

4. If G' is a subgraph of G that is a union of mesh cycles, then G' is defined to be *uniform* if its image in the dual of G is *connected.*

Remarks. 1. Note that G' has a natural image in the dual of G since it is a union of mesh cycles of G.

2. Throughout this section the following convention will be in effect:

> If S is a subgraph of G, then $|S|$ will denote the number of vertices in S. If S is a set of mesh cycles $|S|$ will denote the number of vertices in the closure of S.

3. Note that a planar graph that is filtered with respect to some labeling has a natural induced labeling of its dual graph. This will be important in the determination of the *uniform components* of the $G[i]$—see Appendix B.

Here is an outline of some of the latter portions of the PARTITION algorithm:

a. We will construct a labeling of G and an induced filtration (as defined above)—this is done in P.1 below;

b. With respect to this filtration the $G[i]$ will consist of a number of uniform components (see Appendix B—these are *somewhat* similar to biconnected components), each of which will turn out to have a boundary that is a *simple cycle.* These are the subgraphs in the decomposition of G alluded to above.

c. We will test each of these components to determine whether they enclose a suitable number of vertices. If one of them *does* then *its* boundary can be used as the partitioning cycle. This is done by steps P.2 through P.7.

d. If not we find the *smallest* component that is *too large*—this is step P.8. It will contain a number of sub-components, each of which will be *too small,* and there will be a particularly simple combinatorial relationship between the large component and the small sub-components. This combinatorial relation is essentially the result of the way a filtration of G was defined—all of the sub-components that are too small will either contain a common vertex, v, or be wedged between a set of mesh cycles that contain v (here the term "wedged between" means that all boundary edges of the sub-components that do not contain v are in the mesh cycles that contain v). Step P.9 checks to see if any of the mesh cycles here are suitable partitioning cycles.

e. *Aggregate* (form unions of) the mesh cycles that contain v and the sub-components wedged between them until a union is found whose boundary encloses a suitable number of vertices of G—this is steps P.10 through P.12.

This is *possible* because the union of all of these mesh cycles and sub-components is too large, as mentioned above, and each of the items being aggregated is too small. Although this procedure would appear to be a sequential one, it can actually be carried out in parallel by using a labeling and filtering process somewhat similar to the one used to arrive at the components in the first place.

DEFINITION 2.2. Let G' be a subgraph of G. Define the *boundary* of G' to be the ring sum of all the mesh cycles in G'. This boundary will, in general, consist of the union of a number of simple cycles, some enclosing others (i.e. separating them from the unbounded mesh cycle).

We will begin with the initial labeling algorithm. Basically we want to label the vertices of G in such a way that each vertex has a *unique* label and any two vertices are connected by a path such that the labels of all the vertices in the path are $>$ at least one of the labels of the end vertices. Furthermore, we want every vertex (except the highest labeled one) to be *adjacent to a higher-labeled vertex*—i.e. we do not want label values to have any *local maxima.* This will turn out to ensure that the sets $G(i)$ (with respect to the filtration associated with this labeling) will be *uniform* (i.e. their images in the *dual* of G will be connected). This will guarantee that the uniform components of the $G[i]$ will have boundaries that are *simple cycles.* We will adopt the following scheme:

a. Build an arbitrary *rooted spanning tree*;

b. Number the vertices in *preorder*;

c. For each vertex compute the label value as $|G|+1-$(preorder index).

That this scheme has the required properties follows by a straightforward induction on the height of the spanning tree: it is trivially true for a tree of height 1; it is also easy to see that it is true for a larger tree if it is true for all subtrees of the root.

P.1. 1. *Construct a spanning tree—this can be done in* $O(\log^2 n)$*-time with* $O(n^2/\log^2)$ *processors* (*see* [1]);

2. *Root the spanning tree and give it the preorder numbering—this can be done in* $O(\log n)$*-time using* $O(n)$*-processors using the Euler Tour technique of Tarjan and Vishkin* (*see* [15]).

3. *Compute the labels as described above—this requires* $O(1)$*-time using* $O(n)$*-processors*

Remarks. 1. I am indebted to the referee for pointing out the Euler Tour technique.

2. Applied to the graph in Fig. 1.1, this algorithm gives Fig. 2.1 (where the edges of the spanning tree used have been highlighted).

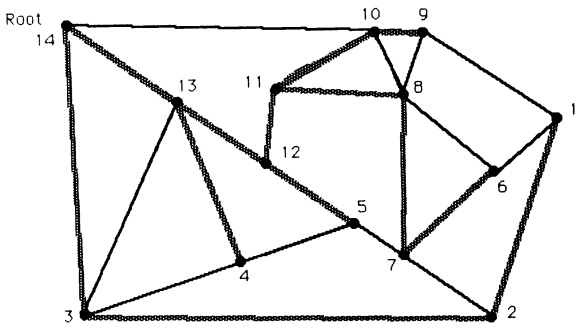


FIG. 2.1

3. In the remainder of this paper we will assume that the vertices of G have been labeled by the procedure described above and that G has the corresponding filtration (see 2.1). Construction of the filtration from the labeling can be accomplished in $O(\log n)$-time using $O(n^2)$ processors. If v is a vertex of G let $f(v)$ denote its label—a nonnegative integer. The following two propositions give the main reason for using the procedure P.1 to order the vertices.

PROPOSITION 2.3. *Let* v_1 *and* v_2 *be two vertices of G. Then there exists a path connecting* v_1 *and* v_2 *with the property that if v is any vertex of the path* $f(v) \geqq \min(f(v_1), f(v_2))$.

PROPOSITION 2.4. *The residual sections, $G\{j\}$, are uniform for all j.*

Remarks. 1. See Definition 2.1 for the definition of the $G\{j\}$.

2. Recall the definition of uniformity in statement 5 of Definition 2.1. Also see Appendix B.

3. If we number the mesh cycles of the graph in Fig. 2.1 with the maximum of the labels of their vertices, we get the graph in Fig. 2.2 (where the numbers in ovals are the numbers associated with the corresponding mesh cycles).

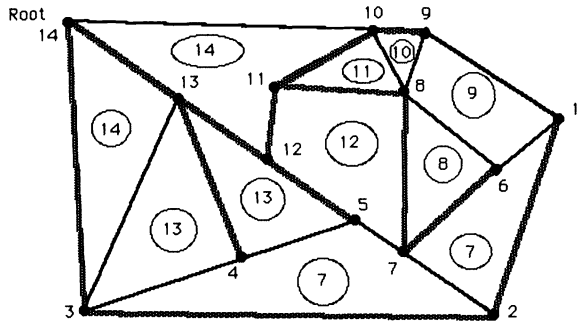


FIG. 2.2

The residual sections are easy to make out in this example—they are the subgraphs spanned by all edges in mesh cycles such that the numbers in the ovals are $\geqq k$ for each value of k that occurs in the graph.

Proof. This follows immediately from the definition of uniformity and a filtration with respect to a labeling. Given any two mesh cycles, M_1 and M_2, of $G\{j\}$, the definition of a filtration with respect to a labeling implies the existence of vertices v_1 in M_1 and v_2 in M_2 that have labels $> j$. Proposition 2.3 implies that there is a *path* connecting v_1 with v_2, *all of whose vertices* have labels $> j$. But the definition of a filtering with respect to a labeling then implies that *every mesh cycle* containing a vertex of this path is in $G\{j\}$, and it isn't hard to see that these mesh cycles connect the original two mesh cycles M_1 and M_2 in the *dual graph.* □

P.2. *Form the dual, D, of G and assign labels to the dual vertices.*

Remarks. This can be done in $O(\log n)$ time with $O(n^4)$ processors (see [7]). Here, the labels assigned to vertices of the dual are equal to the filtration indices of the corresponding mesh cycles.

P.3. *Form the subgraphs, $D[i]$, of D for all $i \geqq 0$, where $D[i]$ is the image of $G[i]$.*

Remarks. 1. Recall that the vertices of D correspond to the *mesh cycles* of G and inherit their labels from the mesh cycles of G—i.e. the *label* of a vertex of D is equal to the *filtration index* of the corresponding *mesh cycle* of G.

2. The construction of the $\{D[i]\}$ can be done in unit time with $O(n^3)$ processors.

P.4. *Compute functions d_i:* {the vertices of $D[i]$} $\to \{D\{i,j\}\}$ *where, for a fixed i, $\{D[i,j]\}$ is the set of components of $D[i]$.*

Remarks. 1. This can be done in $O(\log^2 n)$ time with $O(n^3/\log^2 n)$ processors, using the algorithm of Chin, Lam and Chen of [1]—which is carried out in *parallel* for each $D(i)$, $i = 1, \cdots, n$. That algorithm constructs a function mapping each vertex into the *smallest-indexed* vertex in the *same component.*

2. There will be, at most $O(n^2)$ of the $D[i,j]$.

3. In this step it will not be enough to just determine the set of connected components of the $D[i]$—we will actually use the functions $\{d_i\}$ in later steps—i.e. all parts of the algorithm from P.8 on.

P.5. *Map the $D[i,j]$ back into G to get the uniform components $G[i,j]$ of the $G[i]$.*

Remarks. 1. *These uniform components* will all have boundaries that are *simple cycles* by Proposition B.5 in Appendix B and Proposition 2.4 above. This is the reason we form the $D\{i,j\}$ (and why we deal with the dual graph *at all*).

2. This involves several steps:

   a. Form a boolean array $A_{x,y,z}$, where x runs over all of the $D[i,j]$, y runs over all of the vertices of D (i.e. the mesh cycles of G), and s runs over all of the vertices of G—this array has $O(n^4)$ elements. For each of the n vertices (u_t) of D in the n^2 sets $\{D[i,j]\}$, found in P.4, set the entries in $A_{x,y,z}$ to 1 that correspond to the vertices in G contained in the *mesh cycle* that u_t represents. This can be done in unit time with $O(n^4)$ processors (i.e., there is one processor for each possible combination of the subscripts of the A-matrix).

   b. Now, for each value of x and z, form the result of taking the binary *OR* operation of $A_{x,y,z}$. This can be done in $O(\log n)$ time with $O(n^4)$ processors. This matrix (with *two* subscripts) represents the set of vertices in $G[i,j]$,

P.6. *For each pair (i,j) compute the set of edges and vertices in the boundary of $G[i,j]$, and the set of vertices in the interior of $G[i,j]$.*

Remarks. 1. We proceed as in P.5, except that the last subscript of the matrix now represents *edges* rather than *vertices* and we use the *exclusive OR* operation over the middle subscript rather than the *OR* operation. The same estimates for time and number of processors apply. After computing edges in the boundary we compute all of the vertices that appear in the edges using a matrix operation similar to that in P.5 (except the middle subscript represents *edges* and the last subscript corresponds to vertices that appear as *endpoints* of those edges). In the last step we delete all of the vertices in $G[i,j]$ that also appear in the *boundary* matrix to get a matrix representing the interior of $G[i,j]$—this can be done in unit time using $O(n^3)$ processors.

2. By B.5 in Appendix B, the boundaries computed here will be *simple* cycles.

P.7. *Determine whether, for any pair (i,j):*

the number of vertices in the interior of $G[i,j]$ is $\leqq 2|G|/3$, the *and number of vertices in $G[i,j]$* is $\geqq |G|/3$.

If so, use the boundary of $G[i,j]$ as the partitioning cycle and halt the algorithm.

Remarks. 1. This happens to occur for the graph in Fig. 2.1. In this case $G[7,1]$ has 8 vertices so we can use *its* boundary as the partitioning cycle; see Fig. 2.3.

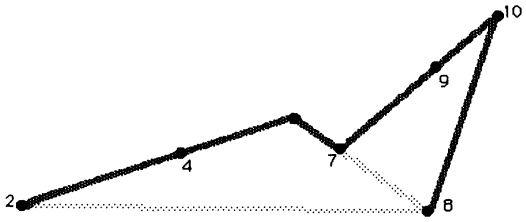


FIG. 2.3

2. It is straightforward to compute the number of vertices in the interior of $G[i,j]$ given the information now at our disposal.

3. If step P.7 *is not successful* then all of the $G[i,j]$ have the property that *either*:

the number of vertices in the interior of $G[i,j] > 2|G|/3$ *or*

the number of vertices in $G[i,j] < |G|/3$.

We execute the *following* steps:

P.8. *Determine the lowest value, h, such that the number of vertices of G in the interior of* $G[h,j]$ *is* $<2|G|/3$.

Remarks. 1. Note that, as h increases the number of vertices in $G[h]$ also increases.

2. This can clearly be accomplished in $O(\log n)$ time using $O(n^2)$ processors.

3. In the remainder of this section we will restrict our attention to $G[h,j]$ and certain subgraphs of it. $G[h,j]$ will consist of:

a. mesh cycles of filtration index h;

b. some uniform components, $G[h-1,j_k]$—i.e. $G[h-1]$ is a subgraph of $G[h]$ and we are considering the uniform components of $G[h-1]$ that are subgraphs of $G[h,j]$.

The $G[h-1,j_k]$ can be determined by plugging vertices of the components of $D[h-1]$ into the function d_h computed in step P.4 above. The remarks following P.6 imply that the number of vertices in each of the $G[h-1,j_k]$, $i=1,\ldots,j$ is $<|G|/3$.

4. Note that the subgraphs $\{G[h-1,j_k]\}$ are *edge-disjoint* since they are represented by *disjoint* subgraphs of D.

P.9. *Determine whether any one of the mesh cycles in* $G[h,j]$ *with filtration index equal to that of h has* $\geqq|G|/3$ *vertices. If so, use that mesh cycle as the partitioning cycle, and halt the algorithm.*

PROPOSITION 2.5. *Each of the subgraphs* $G[h-1,j_k]$ *has edges in common with at least one mesh cycle of filtration index equal to h.*

Remarks. 1. This follows from the fact that:

i. $G[h,j]$ is *biconnected.*

ii. $G[h,j]$ contains only $G[h-1,j_k]$ and mesh cycles whose filtration indices are equal to that of h;

iii. the $G[h-1,j_k]$ are *edge disjoint.*

2. Note that all of the mesh cycles of $G[h,j]$ with filtration index equal to h have a *common vertex* (it is the unique vertex of G that was assigned the label equal to h in step P.1), when we will call x throughout the remainder of this section.

3. At this point the image to bear in mind is that of a "wagon wheel"—the *hub* is the vertex x, each *wedge* or *sector* of the wheel (i.e. cycle formed by two adjacent spokes and a portion of the rim) is a mesh cycle of filtration index h. Here the wheel may be "incomplete" in the sense that *some* of the wedges may be *missing*—they might not be included in $G[h,j]$. In fact, it is not hard to see that the wheel will only be complete if x is the vertex with the highest label in all of G (the way that the graph was labeled implies that the label values will not have any "local maxima"). Nevertheless the whole object will be *uniform* (see statement 5 of Definition 2.1 in this section or Definition B.1 in appendix B for the definition) so that if wedges are missing they must be *adjacent* to each other or the *hub* would become an *articulation point* (*uniformity* implies *biconnectivity*—as pointed out in Appendix B, it is a *strictly stronger* condition on a subgraph of G).

Figure 2.4 illustrates these points:

Here the *shaded regions* are in $G[h,j]$. The *lightly shaded* regions are mesh cycles of filtration index h, and the *darkly shaded* regions (a, b, and c) are $G[h-1,j_k]$'s. The latter *cannot include* x because then their filtration indices would be h. Note that the regions labeled a and b are *distinct* $G[h-1,j_k]$'s—they are *edge-disjoint.* We are assuming that $y>x$ so that the mesh cycles containing y cannot be in $G[h,j]$. In this example the "wagon wheel" is "incomplete".

PROPOSITION 2.6. *Let M be a mesh cycle that contains x. Then M has filtration index equal to h. In particular, none of the* $G[h-1,j_k]$ *contains x.*

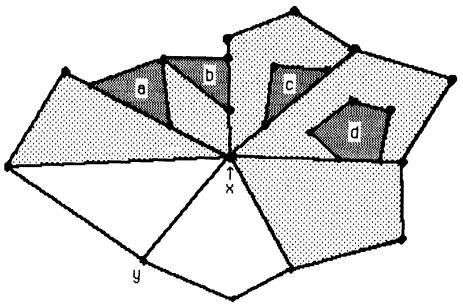


FIG. 2.4

Proof. This follows immediately from the definition of filtration index. □

Our strategy at this point will be to form unions of mesh cycles and $G[h-1, j_k]$'s until we find one that encloses the appropriate number of vertices. This is done, again, by labeling the mesh cycles and $G[h-1, j_k]$'s.

Incidentally, in this portion of the algorithm, the $G[h-1, j_k]$'s are treated as *single objects*—we never make further use of the "fine structure" of the $G[h-1, j_k]$'s except in counting the number of vertices enclosed by a simple cycle. They might be thought of as "weighted mesh cycles" where the weight of a $G[h-1, j_k]$ is equal to the number of vertices it has—and this may be *strictly greater* than the number of vertices in the *boundary* of the $G[h-1, j_k]$.

After labeling, we consider subgraphs of $G[h, j]$ that are unions of objects whose labels are $\leqq$ various label values. (Note: this labeling procedure is *similar* to, but *not the same as* that introduced at the beginning of this section—that labeling procedure has given us $G[h, j]$, the mesh cycles of filtration index h, and the $G[h-1, j_k]$ and will *not be used for anything further in the remainder of this section.*) We want these subgraphs to have the following properties:

C.1: *each subgraph must be uniform and have a uniform complement in G. This will imply that its boundary is a simple cycle, by Proposition* B.5 *in Appendix* B.

C.2: *if, in the labeling scheme used, the label value v' is the immediate successor of v then (the subgraph associated with v)\(that associated with v) must have* $\leqq |G|/3$ *vertices. This will imply (by a kind of "discrete mean value theorem" or pigeonhole principal) that at least one of the subgraphs in question will have the property that the number of vertices in the interior or the whole of the subgraph is* $\geqq |G|/3$.

We will begin by labeling the mesh cycles of $G[h, j]$ that contain x (the "sectors of the wagon wheel"). We will make use of the dual of G, computed in step P.2. Let W denote the image in the dual of the mesh cycles of $G[h, j]$ that contain x.

Since all of the vertices of W (in the dual of G) represent cycles with a common vertex (x) in G it follows that W is either a *simple path* or a *simple cycle*—these possibilities correspond to the cases, respectively, where the "wagon wheel is incomplete" and where the "wagon wheel is complete" mentioned above.

P.10: *If W is a path, then number its vertices, beginning at either end. If it is a simple cycle then*:

a. *delete the vertex that represents the unbounded mesh cycle*;

b. *number the vertices in the resulting path, beginning at either end and label the unbounded mesh cycle with a value higher than any of the other label values assigned.*

Remarks. 1. In the first case we can number the vertices by measuring the distance of each vertex from the end—this case can be done in $O(\log^2 n)$ time with $O(n^3)$ processors.

2. In the second case we use the fact that the unbounded mesh cycle will be in W—this follows from the way the original labeling was done in P.1—the root of the spanning tree used to label the vertices of G was picked to be *on the unbounded mesh cycle*, and this root ends up being the vertex x.

P.11: *Assign a pair of numbers* (e_1, e_2) *to each mesh cycle of* $G[h, j]$ *as follows*:

a. *if* m *is a mesh cycle in* W *set* e_1 *to the label computed in* P.10 *and set* e_2 *to zero*;

b. *If* m *is in one of the* $G[h-1, j_k]$ *set* e_1 *to the minimum of the labels computed in* P.10 *for the mesh cycles of* W *that have edges in common with* $G[h-1, j_k]$ *and set* e_2 *to* k.

Remarks. 1. This can be done in $O(\log n)$ time with $O(n^2)$ processors.

2. Step b can be easily done in the dual since an image of a $G[h-1, j_k]$ in the *dual* will be *adjacent* to one of the vertices of W if and only if the *subgraphs in* G have a *common edge.*

DEFINITION 2.7. Define the $L(e_1, e_2)$ to consist of the closure of the set of mesh cycles of $G(h)$ with labels $\leqq (e_1, e_2)$, where we assume that these pairs of labels are given the *lexicographic ordering.*

PROPOSITION 2.8. *The* $L(e_1, e_2)$ *are uniform and have uniform complements in* G. *Consequently their boundaries are simple cycles.*

Proof. Uniformity of $L(e_1, e_2)$ follows from the fact that $L(e_1, e_2)$ is a *union* of mesh cycles along *common edges.* This is immediate for the mesh cycles containing x since their image in the dual will simply be a sub-path of W. For the $G[h-1, j_k]$ this follows from the fact that the $G[h-1, j_k]$ are *uniform* and we adjoin to a mesh cycle of W *all* of the $G[h-1, j_k]$ that are adjacent to it in D (which implies a *common edge*). We prove uniformity of the complements by induction on the pair (e_1, e_2). Since $L(0, 0)$ is a *single mesh cycle* the second statement is clearly true in the lowest possible case. Now suppose it is known to be true up to (e_1, e_2). There are two cases to be considered:

a. *The next higher pair has a higher value of* e_2. In this case we are adjoining one of the $G[h-1, j_k]$ to a union of mesh cycles whose boundary is a simple cycle. Since the $G[h-1, j_k]$ are *edge-disjoint* this procedure will not *separate* any $G[h-1, j_{k'}]$ with $k \neq k'$ from the *complement* (actually, *the images in the dual graph* will not be separated—an equivalent way to think of this is that it will not make them *edge-disjoint* (unless *all* edges of $G[h-1, j_{k'}]$ are in common with mesh cycles of $L(e_1, e_2)$. This can happen in two ways:
   i. $G[h-1, j_{k'}]$ edge-adjoints a mesh cycle of filtration degree h with a lower label than e_1. But is this case $G[h-1, j_{k'}]$ will be included in $L(e_1, e_2)$ because the pairs of labels are regarded as being lexicographically ordered—so the $G[h-1, j_{k'}]$ would have first index $< e_1$ and its overall label value would be $< (e_1, e_2)$.
   ii. All of the boundary edges of $G[h-1, j_{k'}]$ are shared with the mesh cycles in $L(e_1, e_2)$ with label equal to $(e_1, 0)$ (i.e. the *top-labeled* mesh cycle), and this contradicts the biconnectivity of G.

b. *The next higher pair has a higher value of* e_1. In this case we are adjoining a mesh cycle represented by a vertex of W. If any of the $G[h-1, j_k]$ is caught *between* this new mesh cycle and $L(e_1, e_2)$ (i.e. if its image in the dual is separated from the image of the complement of $L(e_1, e_2)$ in the dual) it will be *edge-adjacent* to $L(e_1, e_2)$ (since it must be adjacent to at least one of the mesh cycles containing x) and will, consequently, be *included* in it (because it will have a label that is $\leqq (e_1, e_2)$.

The remainder of the algorithm consists in:

P.12: *Compute the number of vertices in G in the interior of each of the $L(e_1, e_2)$ and select an $L(e_1, e_2)$ with the property that it or its interior has $\geqq |G|/3$ and $\leqq 2|G|/3$ vertices. Use its boundary as the partitioning cycle.*

Remarks. The computation of the number of vertices can be done by a process similar to that used in P.7—alternatively it can be accomplished by labeling the vertices of G with the minimum of the filtration indices of all the mesh cycles containing them and counting the number of vertices whose labels are $\leqq(e_1, e_2)$. This can be done in $O(\log n)$ time with $O(n^4)$ processors.

That it is *possible* to find an $L(e_1, e_2)$ with the required number of vertices follows from:

a. $L(0, 0)$ is a single mesh cycle whose interior has *no* vertices.
b. Each of the $G[h-1, j_k]$ and each of the other mesh cycles in $G[h-1, j]$ has $<|G|/3$ vertices, so that each time a $G[h-1, j_k]$ is adjoined to the union of mesh cycles already present the number of vertices increases by $<|G|/3$.
c. The total number of vertices in $G[h-1, j]$ is $>2|G|/3$.

Appendix A. In this appendix we will show how to convert the depth-first spanning tree found at the end of § 1 into a *depth-first search.* In other words the spanning tree will be labeled so that each vertex has a number associated with it equal to the *order* in which it would have been encountered during a depth-first search. Frequent references will be made to the results and notation of § 1, particularly 1.9.

First we consider the case in which the original graph was *biconnected.* The idea here is that vertices encountered along P found in step 4.3 of Algorithm 1.9 should get *higher* labels the *further* they are from the *entry vertex* (within an iteration of steps 2 through 4 in Algorithm 1.9). In addition essential edges coming out of P represent *backtracking* so their descendants should get higher labels—and the closer they are to the entry vertex the *later* the backtracking took place. Consequently, in the case where G is biconnected the vertices of G are ordered as follows:

DEFINITION A.1. Let v_1 and v_2 be distinct vertices of G.

a. If v_1 and v_2 are on a path P constructed in an iteration of steps 2 through 4 to 1.9 then $v_1 > v_2$ if v_1 is further from the entry vertex.
b. If v_1 is on a path P as before and v_2 is a descendant of an essential edge coming out of P, found in the same iteration of steps 2 through 4 of 1.9, then $v_2 > v_1$.

Remarks. 1. In any iteration of steps 2 through 4 of Algorithm 1.9, if two essential edges are found that are the *same* distance from the entry vertex an *arbitrary ordering* must be imposed and used in the statement above.

2. This order relation can be easily implemented as follows:

Associate a list of numbers with each vertex and some edges of G—the maximum length of any such list is $2\times$(the number of iterations of steps 2 through 4 of Algorithm 1.9). The lists will be ordered lexicographically and each vertex will be labeled with the ordinal position of its associated list.

1. Associate the empty list with the first edge of the depth-first spanning tree;
2. If v is a vertex of G encountered in an iteration of step 4.3 of Algorithm 1.9, associate with v the *concatenation* of the list associated with the entry edge (i.e. the essential edge leading out of the previous iteration of step 4.5) with the pair of numbers $(-\infty,$ distance of v in P from entry vertex);
3. If e is an essential edge found in an iteration of step 4.5 determine the list associated with e as follows: if the list $(L_1, -\infty, d)$ is associated with the end

vertex of e in P (where L_1 is some set of terms) then associate the list $(L_1, -d)$ with e.

Remarks. 1. In the first iteration of Algorithm 1.9 we regard the *root* of the tree as the *entry vertex* and the *first edge* (with the empty list associated with it) as the *entry edge.*

2. $-\infty$ represents some number that compares *low* with all other numbers encountered in this algorithm—i.e., if G has n vertices we could use $-n-1$ in place of $-\infty$.

3. In step 3 if several essential edges have the same end vertex in P they must be ordered in some way so that we get distinct lists associated with them—we could use rational numbers for the distance, d, in statement 3 above.

In the rest of this appendix we will regard an order relation as a boolean function of two variables that measures whether the first variable is larger than the second.

DEFINITION A.2. If v_1 and v_2 are in G_s, define $v_1 > v_2$ or $\mathcal{S}(v_1, v_2) = 1$ if the list associated with v_1 is lexicographically $>$ the list associated with v_2.

Now we will extend this ordering to an ordering, r, defined for all pairs of vertices of G in the case when G *is not biconnected.* If v is a vertex of G_t in a component C, let $\mathcal{A}(v)$ denote the vertex of G_s to which C is attached. Now define

a. If v_1 and v_2 are both G_s set $r(v_1, v_2) = \mathcal{S}(v_1, v_2)$.
b. If v_1 is in G_s and v_2 is in G_t then $r(v_1, v_2) = \mathcal{S}(v_1, \mathcal{A}(v_2))$, unless $v_1 = \mathcal{A}(v_2)$—in this case $v_2 > v_1$. The case where v_2 is in G_s and v_1 is in G_t is covered by anti-symmetry.
c. If v_1 and v_2 are in *distinct components* of G_t define $r(v_1, v_2) = \mathcal{S}(\mathcal{A}(v_1), \mathcal{A}(v_2))$ unless $\mathcal{A}(v_1) = \mathcal{A}(v_2)$—in this case order these components in some arbitrary way (that is fixed throughout the algorithm) and order the vertices correspondingly.
d. If v_1 and v_2 are in the same component of G_t define r to be an order-relation that is the inverse of the relation defined over a tree in P.1 in § 2 of this paper.

The last step is to simply sort the vertices via the order-relation defined here—this can be done in $O(\log n)$ time using $O(n)$ processors.

Appendix B. In this appendix we will prove some technical results concerning when the boundary of a subgraph of a graph is a simple cycle. Throughout this appendix we will assume that G is a planar, biconnected graph that is equipped with an imbedding into the plane and D is its *dual.*

DEFINITION B.1. Let G_1 be a subgraph of G that is a union of mesh cycles. Then

a. G_1 will be said to be *uniform* if its image in the dual of G is *connected.*
b. A subgraph, G_2, will be called the *complement* of G_1 in G if G_2 is the union of all mesh cycles of G *not* in G_1.
c. The *boundary* of G_1 is the ring sum of all the mesh cycles in G_1;

Remarks. 1. This definition depends upon the imbedding of G into the plane. Clearly, since G_1 is a union of mesh cycles it has a natural image in D.

2. It is not difficult to see that any uniform subgraph of G will be *biconnected*—i.e. it will be a union of mesh cycles along *common edges.*

Not all biconnected subgraphs of G are uniform, however. Consider the case where G is the graph in Fig. B.1 and G_1 be the shaded subgraph. Then G_1 is biconnected but clearly not uniform.

PROPOSITION B.2. *Let G_1 be a subgraph of G that is a union of mesh cycles and let G_2 be its complement in G. Then the boundary of G_1 is equal to the boundary of G_2 and equal to the intersection of G_1 and G_2.*

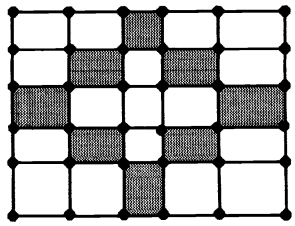


FIG. B.1

Proof. This follows from the definition of the boundary and the fact that every edge of G is contained in *precisely two mesh* cycles because G is *planar*—see [5, p. 115]. If an edge is contained in the boundary of G_1 *one* of these mesh cycles must be in G_1 and the *other* must be in G_2. □

The main result of this appendix is:

PROPOSITION B.3. *Let G_1 be a subgraph of G that is a union of mesh cycles. Then the boundary of G_1 is a simple cycle if and only if G_1 and its complement are both uniform.*

Proof. Let G_2 be the complement of G_1 in G and suppose the boundary of G_1 (and that of G_2) is a *simple cycle.* Let D_1 be the image of G_1 in the dual. Suppose D_1 has distinct components E and F (there might be more). These components correspond to mesh cycles in G that have no common edges. Consequently, when we form the boundary of G_1 we will get boundary components that are either *completely disjoint* or only have *vertices in common.* In any case the boundary will not be a simple cycle. Since this boundary is also the boundary of G_2, the same line of reasoning applies to the complement.

In order to prove the *if* part of the statement, we will assume that the boundary of G_1 is not a simple cycle and show that D_1 or D_2 (or both) *cannot be connected.* If the boundary is not a simple cycle, it will be a union of simple cycles. Furthermore two distinct cycles in this union must only intersect at *common vertices,* by the definition of the boundary (since the boundary is formed by taking a mod 2 sum of edges so any edge common to *two* boundary cycles would be in *neither* of them).

Now we will make explicit use of the imbedding of G into the plane. Regard each mesh cycle M_i as a *face* of the plane—i.e. label the portion of the plane it encloses—see [5, p. 103]. Delete the image of the boundary of G_1 in the plane from the plane. Since the boundary is not a simple cycle, the result will have $\geqq 3$ components—this follows from the *Euler polyhedron formula*—[5, p. 103] (here we temporarily replace G by the boundary of G_1 and consider the set of *its* mesh cycles when it is regarded as a planar graph). Call these components P_i and define U_i to be the union of the mesh cycles of G contained in P_i (if we regard a mesh cycle to be a face of the plane).

Since there are more than two of the U_i at least one of G_1 and G_2 must be a union of more than one of the U_i—assume that G_1 has this property and it contains U_1 and U_2. Proposition B.2 implies that whenever an edge of G is contained in U_1 and U_i $(1 \neq i)$ and U_1 is in G_1 then U_i must be in G_2. But this implies that the image of U_1 and U_2 in the dual can't be connected by a path contained in the image of G_1 since such a path would correspond to a sequence of mesh cycles in G that are connected by *common edges.* □

DEFINITION B.4. If G_1 is a subgraph of G that is a union of mesh cycles, the *uniform components* $\{C_i\}$ of G_1 are defined as follows:

a. map G_1 into the dual of G;
b. If $\{D_i\}$ are the connected components of the image of G_1 then C_i is the union of the mesh cycles of G corresponding to the vertices of D_i.

Remark. G_1 is clearly equal to the *union* of the C_i.

PROPOSITION B.5. *Let G_1 be a subgraph of G that is a union of mesh cycles and let G_2 be its complement in G. If G_2 is uniform then the boundary of each of the uniform components of G_1 is a simple cycle.*

Proof. Let U be a uniform component of G_1. Proposition B.3 implies that we only have to verify that the *complement* of U is uniform. The complement of U is the union of G_2 with the other uniform components of G_1. That result must be uniform because each of the other uniform components of G_1 must have at least one edge in common with G_2. □

Acknowledgments. I am indebted to Professors Jane Cameron and Gomer Thomas for encouraging me to enter this field. I am also indebted to Drexel University for providing me with a research grant.

I am also indebted to the referees for an extremely careful reading of this paper and many helpful comments.

REFERENCES

[1] F. Y. CHIN, J. LAM AND I. CHEN, *Optimal parallel algorithms for connected-component problems*, Proc. 1981 International Conference on Parallel Processing, 1981, pp. 170–175.

[2] S. COOK, *An overview of computational complexity*, Comm. ACM, 26 (1983), pp. 401–408.

[3] D. ECKSTEIN AND D. ALTON, *Parallel graph processing using depth-first search*, Proc. Conference on Theoretical Computer Science, University of Waterloo, 1977, pp. 21–29.

[4] M. FLYNN, *Very high-speed computing systems*, Proc. IEEE, 54(1066), pp. 1901–1909.

[5] F. HARARY, *Graph Theory*, Addison-Wesley, Reading, MA, 1972.

[6] J. JA'JA' AND J. SIMON, *Parallel algorithms in graph theory: Planarity testing*, this Journal, 11 (1982), pp. 314–328.

[7] D. JOHNSON AND S. VENKATESAN, *Parallel algorithms for minimum cuts and maximum flows in planar graphs*, Proc. IEEE Symposium on Foundations of Computer Science, 1982, pp. 244–254.

[8] R. LIPTON AND R. TARJAN, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 17–189.

[9] ———, *Applications of a planar separator theorem*, this Journal, 9 (1980), pp. 615–627.

[10] G. L. MILLER, *Finding small simple-cycle separators for 2-connected planar graphs*, Proc. 16th Annual ACM Symposium on the Theory of Computing, 1984, pp. 376–382.

[11] J. MILNOR, *Morse Theory*, Ann. Math. Studies 51, 1963.

[12] F. PREPARATA, *Parallelism in sorting*, International Conference on Parallel Proc., Bellair, MI, 1977.

[13] E. REGHBATI AND D. CORNEIL, *Parallel computations in graph theory*, this Journal, 7 (1978), pp. 230–237.

[14] R. TARJAN, *Depth-first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146–160.

[15] R. E. TARJAN AND U. VISHKIN, *Finding biconnected components and computing tree functions in logarithmic parallel time*, 25th Annual Symposium on Foundations of Computer Science (1984), pp. 12–20.

[16] E. UPFAL, *A probabalistic relation between desirable and feasible models of parallel computation*, Proc. 16th Annual ACM Symposium on the Theory of Computing, 1984, pp. 258–265.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986

© 1986 Society for Industrial and Applied Mathematics
014

RECURSION SCHEMES AND RECURSIVE PROGRAMS ARE EXPONENTIALLY HARD TO ANALYZE*

H. B. HUNT III† AND D. J. ROSENKRANTZ‡

Abstract. Deterministic exponential lower time bounds are presented for analyzing recursion schemes and recursive programs. The lower bounds for recursion schemes hold for any interpretation with a *nontrivial* predicate, i.e. a predicate that is neither identically true nor identically false. The lower bounds for recursive programs hold for very simple programs in any recursive programming language with a nontrivial predicate. These lower time bounds hold for the executability, computational identity, totality, divergence, partial correctness, and total correctness problems.

Key words. recursion, recursion schemes, recursive programming languages, computational complexity, exponential time, decision problems, strong equivalence, totality, total and partial correctness

AMS(MOS) subject classifications. 68Q55, 68Q60, 68N15, 68Q25

1. Introduction. Often one wishes to analyze computer programs for properties of interest such as isomorphism, strong equivalence, totality, partial correctness, total correctness, etc. Unfortunately, most interesting questions about programs in general computer languages are undecidable. However, there are two obvious ways in which the decidability of program analysis might be circumvented. First, the undecidability of such problems may depend on the programming language features present in the programs considered; and these problems may be decidable for sufficiently stripped down programming languages [2], [8], [13], [15]. Second and more likely, it may be possible to analyze those programs that are *actually* written in practice. Hopefully, a person who writes a "good" program understands how and why the program works. Since such "good" programs are written and understood by people, they may also be analyzable by machine. Thus, it is desirable to characterize classes of programs that can be analyzed efficiently and that are broad enough to contain many practical programs [14].

Here, we show that *no* such class of easily analyzable recursive programs can be described solely in terms of semantic restrictions on the predicates and functions allowed in a program. We show that the only semantic restriction on the predicates and functions of a recursive program that suffices to guarantee deterministic polynomially time-bounded analyzability is—

"All predicates are *trivial*, i.e. either identically true or identically false."

Our results provide an answer to the following question—

1.0. "What is the inherent complexity of recursive program analysis due only to the presence of recursive function calls and predicate tests?"

They show that the answer to the question 1.0 is—

"Analyzing recursive programs requires deterministic exponential time."

* Received by the editors July 17, 1984, and in final revised form August 1, 1985. A preliminary version of some of these results was presented at the 21st Annual IEEE Symposium on Foundations of Computer Science, Syracuse, New York, October 1980.

† Computer Science Department, State University of New York at Albany, Albany, New York 12222. The research of this author was supported in part by the National Science Foundation under grants MCS 77-27197 and 80-03353.

‡ Computer Science Department, State University of New York at Albany, Albany, New York 12222. The research of this author was supported in part by the National Science Foundation under grants MCS 78-03157 and 82-03237.

This answer holds for very simple programs in any recursive programming language with a *nontrivial predicate*, and is independent of any other properties of the language such as the data types, constants, basis functions, or actual predicates of the language.

Our paradigm for proving these results is of independent interest. Unlike most other work on program or recursion schemes [5], [6], [9], [11], [12] etc., our deterministic exponential lower time bounds do not depend upon consideration of arbitrary Herbrand interpretations. In order to state our paradigm, we need the following definitions and notation. Readers who are unfamiliar with the definition of *recursion schemes* should see § 2.

DEFINITION 1.1. An *interpretation* I of a recursion scheme S consists of the following:

1. a nonempty set D called the *domain* of the interpretation;
2. an assignment of a function from D^n to D to each n-ary function symbol of S;
3. an assignment of a predicate from D^n to {True, False} to each n-ary predicate symbol of S;

and

4. an assignment of an element of D to each constant symbol of S.

An *augmented interpretation* I', written *a-interpretation* I', of a recursion scheme S consists of an interpretation I of S together with an assignment of an element of the domain of I to each input variable symbol of S. An a-interpretation I' of a recursion scheme S is said to be *compatible with* interpretation I of S if and only if I' consists of I together with an assignment of an element of the domain of I to each input variable symbol of S.

Let I be an interpretation or a-interpretation of a recursion scheme S. Let p, f, x, and c be a predicate symbol, basis function symbol, input symbol, and constant symbol of S, respectively. Then p_I, f_I, x_I, and c_I are the predicate, function, element, and element associated by I to p, f, x, and c, respectively. □

Our paradign consists of the following. We construct a class C of very simple recursion schemes. Each element of C is total, has only one predicate symbol p, has only two input variable symbols x and y, and has no occurrences of basis function or constant symbols. (A scheme in C can only shuffle around the values of its input variables. Thus, it cannot compute a value other than the value of one of its input variables.) We show that, for all classes $\mathbf{I}$ of a-interpretations such that

$$(\exists J \in \mathbf{I})[\, p_J(x_J) = \text{True and } p_J(y_J) = \text{False}],$$

the problem of determining for $S \in C$ if there exists an a-interpretation I in $\mathbf{I}$ such that the defined function symbol B is called (i.e. expanded) during the computation of S under I requires deterministic exponential time. In particular our deterministic exponential lower time bound holds even when $\mathbf{I}$ consists of a single a-interpretation I such that $p_I(x_I) = \text{True}$ and $p_I(y_I) = \text{False}$. Any recursive programming language with a nontrivial predicate embodies such a class $\mathbf{I}$ of a-interpretations. Thus our deterministic exponential lower time bounds for analyzing recursion schemes imply deterministic exponential lower time bounds for analyzing very simple programs in any such recursive programming language. We used a similar paradigm in [8] to prove PSPACE-hard lower time bounds for analyzing program schemes and very simple programs in any flowchart programming language with a nontrivial predicate.

The rest of this section consists of definitions, notation, and basic properties of strings, relations, computational complexity, and auxiliary pushdown machines. Section 2 consists of definitions and basic properties of recursion schemes and recursive programs. We assume that the reader is familiar with proofs of lower complexity

bounds that involve efficient reducibility between decision problems, otherwise see [1], [4].

We denote the *length* of a string S or the *cardinality* of a set S by $|S|$; we denote the *empty string* by λ.

DEFINITION 1.2. Let D be a nonempty set. Let ρ, σ and τ be binary relations on D such that

(i) if $x\rho y$ then $x\sigma y$, and

(ii) if $x\sigma y$ then $x\tau y$.

Then we say that the relation σ is *between* ρ *and* τ.

DEFINITION 1.3. Let Σ and Δ be finite nonempty alphabets. Let $L \subset \Sigma^*$, and let $M \subset \Delta^*$. We say that L is *polynomially reducible to* M (L is $O(n \log n)$ *time reducible to* M) if and only if there exists a function f from Σ^* to Δ^* computable by a deterministic polynomially time-bounded Turing machine (computable by a deterministic $O(n \log n)$ time-bounded Turing machine) such that

$$\text{for all } x \in \Sigma^*,\ x \in L \text{ if and only if } f(x) \in M.$$

By a *linearly space-bounded deterministic auxiliary pushdown machine*, abbreviated *linearly space-bounded* DAPDM, we mean a deterministic linearly bounded automaton augmented with an auxiliary pushdown store. Such a machine M is specified in terms of—

1. a finite nonempty set Q of *states*,
2. a finite nonempty *input tape alphabet* Σ,
3. a finite nonempty *pushdown store alphabet* Γ,
4. a *start state* $q_0 \in Q$,
5. two distinct *endmarkers* $\vdash$ and $\dashv$ not in Σ,
6. a *bottom of stack marker* $Z_0 \in \Gamma$,
7. a finite set $F \subset Q$ of *accepting states*, and
8. a *transition function* δ from

$$Q \times (\Sigma \cup \{\vdash, \dashv\}) \times \Gamma \text{ to } Q \times (\Sigma \cup \{\vdash, \dashv\}) \times \Gamma^* \times \{0, 1, -1\}.$$

We interpret the transition $\delta(s, a, Z) = (t, b, \eta, d)$ as follows—when in state s, scanning input tape symbol a, and having top stack symbol Z, the machine in one *move*

(i) changes state to t,

(ii) replaces its scanned input symbol by b,

(iii) replaces its top stack symbol by the (possibly empty) string η, and

(iv) moves its input tape head one cell to the left, moves its input tape head one cell to the right, or keeps its input tape head stationary if $d = -1$, 1, or 0, respectively.

The transition function δ preserves the integrity of the endmarkers $\vdash$ and $\dashv$ as follows—

(a) if $a = \vdash$ then $b = \vdash$ and $d \in \{0, 1\}$,

(b) if $a = \dashv$ then $b = \dashv$ and $d \in \{0, -1\}$, and

(c) if $a \in \Sigma$ then $b \in \Sigma$.

DEFINITION 1.4. A *configuration* of a linearly space-bounded DAPDM $M = (Q, \Sigma, \Gamma, q_0, \{\vdash, \dashv\}, Z_0, F, \delta)$ is a four-tuple (s, z, η, j) where $s \in Q$, $z \in \{\vdash\} \cdot \Sigma^* \cdot \{\dashv\}$, $\eta \in \Gamma^*$, and $1 \leqq j \leqq |z|$. A *transition holds between configurations* $(s, \vdash x \dashv, \eta, j)$ and $(t, \vdash y \dashv, \xi, k)$, denoted by

$$(s, \vdash x \dashv, \eta, j) \vdash_M (t, \vdash y \dashv, \xi, k),$$

if and only if, letting the jth symbol of $\vdash x \dashv$ be a and the rightmost symbol of the

string η be Z,

(i) $\delta(s, a, Z) = (t, b, \gamma, d)$,

(ii) the string $\vdash y \dashv$ is the string $\vdash x \dashv$ with its jth symbol replaced by b,

(iii) the string ξ is the string η with its rightmost symbol replaced by the string γ, and

(iv) $k = j + d$.

For configurations α and β of M, if β is obtained from α by means of a sequence of i transitions, we denote this by

$$\alpha \vdash_M^i \beta, \text{ and}$$

we say that there is a *computation of M from α to β*. The *language that M accepts by final state and empty pushdown store*, denoted by $L(M)$, is the set of all strings $x \in \Sigma^*$ such that

$$(q_0, \vdash x \dashv, Z_0, 2) \vdash_M^i (q_f, \vdash y \dashv, \lambda, k)$$

for some integer $i \geqq 1$, state $q_f \in F$, string $y \in \Sigma^*$, and integer $k \geqq 1$. □

The deterministic exponential lower time bounds presented here are based upon well-known time hierarchy results for deterministic Turing machines [7] and the following property of linearly space-bounded DAPDMS due to Cook [3].

PROPOSITION 1.5. *The class of all languages accepted by linearly space-bounded* DAPDM*s equals the class of all languages accepted by deterministic Turing machines that operate within time 2^{cn} for some $c > 0$.* □

Inspection of the proof of this result (in particular the proof that (c) implies (a) of [3, Thm. 1]) and known time hierarchy results for deterministic Turing machines [7] yield the following corollary of Proposition 1.5 and its proof.

PROPOSITION 1.6. *There exists $c > 0$ and a linearly space-bounded* DAPDM *M such that*

1. *any deterministic Turing machine that accepts the language $L(M)$ makes more than 2^{cn} moves infinitely often*;

2. *M's input tape alphabet is* $\{0, 1\}$;

3. $L(M) \subset \{00\} \cdot \{01, 10\}^* \cdot \{11\}$ *and, thus, is prefix-free*;

4. *M halts for all inputs with an empty pushdown store*;

5. *for all inputs in* $\{00\} \cdot \{01, 10\}^* \cdot \{11\}$, *$M$ never moves its input tape head onto an input tape cell containing an endmarker*; *and*

6. *M pushes at most one stack symbol at a time.*

Assertions 3–6 of the statement of Proposition 1.6 are included to simplify the proof of Theorem 3.1 below. The automaton M treats strings in $\{00\} \cdot \{01, 10\}^* \cdot \{11\}$ as images of strings in $\{\vdash\} \cdot \{0, 1\}^* \cdot \{\dashv\}$ under the string homomorphism h defined by $h(\vdash) = 00$, $h(0) = 01$, $h(1) = 10$, and $h(\dashv) = 11$.

2. Definitions and properties of recursion schemes and recursive programs. We present the basic definitions and properties of recursion schemes and recursive programs needed to read this paper. Our definitions closely follow those in [6] and [12].

The *alphabet* Σ of a recursion scheme is a finite subset of the following set of symbols:

1. n-ary *basis function symbols* $f_i^n (i \geqq 1, n \geqq 0)$, basis function symbols f_i^0 are called *constant symbols*;

2. n-ary *predicate symbols* $p_i^n (i \geqq 1, n \geqq 0)$;

3. *input variable symbols* $x_i (i \geqq 1)$;

4. *program variable symbols* $y_i (i \geqq 1)$;

5. an *output variable symbol* z; and
6. *function variable symbols* $F_i (i \geqq 1)$.

A *quantifier-free wff* π *over* Σ is a quantifier-free wff in the sense of first-order logic construction from basis function symbols, predicate symbols, input variable symbols, program variable symbols, and the output variable symbol z. A *term* t *over* Σ is a term in the sense of first-order logic constructed from basis function symbols, input variable symbols, program variable symbols, the output variable symbol z, and function variable symbols. A *conditional term* τ *over* Σ is defined recursively as follows:

(a) Each term over Σ is a conditional term.

(b) If π is a quantifier-free wff over Σ and τ_1 and τ_2 are conditional terms over Σ, then

$$\textit{if } \pi \textit{ then } \tau_1 \textit{ else } \tau_2$$

is a conditional term over Σ.

A *recursion scheme* S over Σ is of the form

$$z = \tau_0(\bar{x}, \bar{F}) \quad \text{where}$$

$$F_1(\bar{x}, \bar{y}) \coloneqq \tau_1(\bar{x}, \bar{y}, \bar{F})$$

$$\cdots$$

$$F_N(\bar{x}, \bar{y}) \coloneqq \tau_N(\bar{x}, \bar{y}, \bar{F})$$

where $\bar{x}$ and $\bar{y}$ denote finite sets of input variable symbols and program variable symbols, respectively, and $\bar{F} = \{F_1, \cdots, F_N\}$. Here, $\tau_0(\bar{x}, \bar{F})$ is a conditional term over Σ that contains no input variable symbols other than those in $\bar{x}$, contains no function variable symbols other than those in $\bar{F}$, and contains no program variable symbols and no occurrences of z. Also for $1 \leqq i \leqq N$, $\tau_i(\bar{x}, \bar{y}, \bar{F})$ is a conditional term over Σ that contains no input variable symbols other than those in $\bar{x}$, contains no program variable symbols other than those in $\bar{y}$, contains no predicate symbols other than those in $\bar{F}$, and contains no occurrences of z.

For $1 \leqq i \leqq N$, the statement

$$F_i(\bar{x}, \bar{y}) = \tau_i(\bar{x}, \bar{y}, \bar{F})$$

is called the *defining statement for* F_i. We assume that each function variable symbol F_i appearing in a recursion scheme S has exactly one defining statement for it in S.

We denote the set of all recursion schemes by R.

DEFINITION 2.1. The *size* of a recursion scheme M, denoted by $\|M\|$, is the number of occurrences of symbols appearing in M where we consider *where*, *if*, *then*, *else*, and $\coloneqq$ to be single symbols.

Example 2.2. Let S be the recursion scheme—

$$Z = F(x, x) \quad \text{where}$$

$$F(y_1, y_2) \coloneqq \text{if } p(y_1) \text{ then } f_1(y_1) \text{ else } f_2(y_2).$$

Then $\|S\| = 31$.

Recursive programs and recursive programming languages are defined in terms of R as follows.

DEFINITION 2.3. Let I be an interpretation. Let $S \in R$. The pair $P = (S, I)$ is called a *recursive program*. The pair $L = (R, I)$ is called a *recursive programming language*.

The *basis function symbols, constant symbols, predicate symbols, input variable symbols, program variable symbols, output variable symbols, function variable symbols,*

and *defining statement for* the function variable symbol F_j of a recursive program $P=(S, I)$ are the basis function symbols, constant symbols, predicate symbols, input variable symbols, program variable symbols, output variable symbol, function variable symbols, and defining statement for the function variable symbol F_j, respectively, of S. The *size* of a recursive program $P=(S, I)$, denoted by $\|P\|$, is the size of S.

The definitions of a *computation* of a recursion scheme S under an a-interpretation I and of a *computation* of a recursive program S, given values ξ to its input variables, are standard and will not be duplicated here. The sequence of function symbols called in a computation is inside-out, left-to-right, as in [6].

DEFINITION 2.4. Let T be a recursion scheme or recursive program. Let F_j be a function variable symbol of T. Let the defining statement for F_j in T be $F_j(\bar{x}, \bar{y}) := \tau_j(\bar{x}, \bar{y}, \bar{F})$.

1. We say that F_j is *called during the computation of* recursion scheme T *under a-interpretation* I if some occurrence of F_j is replaced by $\tau_j(\bar{x}, \bar{y}, \bar{F})$ during the computation of T under I. We say that F_j is *called during a computation of* T if there exists an a-interpretation I such that F_j is called during the computation T under I.

2. We say that F_j is *called during the computation of* recursive program T, *given values ξ to its input variables* if some occurrence of F_j is replaced by $\tau_j(\bar{x}, \bar{y}, \bar{F})$ during the computation of T, given values ξ to its input variables. We say that F_j is *called during a computation of* T if there exists values ξ to its input variables such that F_j is called during the computation of T, given values ξ to its input variables. □

In §§ 3 and 4 below we consider the computational complexity of a variety of problems for various classes S of recursion schemes or of recursive programs. These problems include the following:

1. The *Executability Problem* or EP: Given P in S and function variable symbol F_j, is F_j called during a computation of P?

2. The *Computational Identity Problem*: Given P and Q in class S of recursion schemes, for all a-interpretations I of P and Q, during the computations of P and Q under I, are the sequences of function variable symbols called, together with their associated defining statements, identical? Given P and Q in class S of recursive programs, for all given values ξ to P and Q's input variables, during the computations of P and Q given ξ, are the sequences of function variable symbols called, together with their associated defining statements, identical?

3. The *Strong Equivalence Problem*: Given P and Q in class S of recursion schemes, for all a-interpretations I of P and Q, do the computations of P and Q under I both diverge or both halt with the same values for their output variables? Given P and Q in a class S of recursive programs, for all given values ξ to P and Q's input variables, do the computations of P and Q given ξ both diverge or both halt with the same values for their output variables?

4. The *Totality Problem*: Given P in S, do all computations of P halt?

5. The *Divergence Problem*: Given P in S, do all computations of P diverge?

6. The *Containment Problem*: (defined in [6] or [12]).

7. The *Weak Equivalence Problem*: (defined in [6] or [12]).

We denote computational identity, strong equivalence, containment, and weak equivalence by $=$, $\equiv$, $\subseteq$, and $\simeq$, respectively. For a class $\mathbf{I}$ of interpretations or assignments of values ξ to input variables, we also consider the restrictions of problems 1-7 to $\mathbf{I}$. Thus, for example, for recursion schemes P and Q in S, we write $P =_{\mathbf{I}} Q$ if and only if, for all a-interpretations I in $\mathbf{I}$ of P and Q, the sequences of function variable symbols called during the computations of P and Q under I, together with their associated defining statements, are identical. We define $\equiv_{\mathbf{I}}$, $\subseteq_{\mathbf{I}}$, and $\simeq_{\mathbf{I}}$,

analogously. Let $k \geqq 1$. If $\mathbf{I}$ consists of all a-interpretations with domains of cardinality k, we denote $=_{\mathbf{I}}$, $\equiv_{\mathbf{I}}$, $\subseteq_{\mathbf{I}}$, and $\simeq_{\mathbf{I}}$ by $=_k$, $\equiv_k$, $\subseteq_k$, and $\simeq_k$, respectively.

In § 4 below we also consider the computational complexity of partial and total correctness for various input and output assertions [6], [12] and for various classes S of recursion schemes or of recursive programs.

DEFINITION 2.5. Let S be a recursion scheme with set $\bar{x} = \{x_1, \cdots, x_m\}$ of input variables and output variable z. Let P and Q be wffs in the sense of first-order logic such that all free variables in P are elements of $\bar{x}$ and such that all free variables in Q are elements of $\bar{x} \cup \{z\}$. Then P and Q are said to be *uninterpreted input* and *output assertions*, respectively, *for S*.

DEFINITION 2.6. Let S be a recursion scheme with set $\bar{x} = \{x_1, \cdots, x_m\}$ of input variables and output variable z. Let P and Q be uninterpreted input and output predicates, respectively, for S. Let I be an interpretation of S such that under I, P and Q become predicates denoted by P_I and Q_I, respectively. Let I' be an a-interpretation compatible with I. We say that the scheme S is *partially correct with respect to P, Q,* and I' if

(i) $P_{I'}((x_1)_{I'}, \cdots, (x_m)_{I'}) = \text{True}$,

and, provided that the computation of S under I' terminates,

(ii) $Q_{I'}((x_1)_{I'}, \cdots, (x_m)_{I'}, z_{I'}) = \text{True}$.

We say that S is *totally correct with respect to P, Q,* and I' if S is partially correct with respect to I' and the computation of S under I' terminates.

The definitions of partial and total correctness for recursive programs are analogous and will not be duplicated here, see [6] and [12].

In §§ 3 and 4 below, we show that variants of the EP for a class C of very simple recursion schemes are efficiently reducible to a variety of problems for R and for any recursive programming language with a nontrivial predicate. These problems include problems 2–7 above, together with the problems of determining partial and/or total correctness with respect to many input and output assertions.

3. Simple recursion schemes with "Hard" EPs. We construct a class C of very simple recursion schemes with a provably deterministic exponential time hard EP. Each member $S \in C$ has only a single predicate symbol p, has only two input symbols x and y, has *no* basis function or constant symbols, and is total. In addition the following are equivalent for S:

3.1. the defined function symbol B is executable in S, i.e. there is an a-interpretation I such that the defined function symbol B is called during the computation of S under I;

3.2. there is an a-interpretation I for which $p_I(x_I) = \text{True}$ and $p_I(y_I) = \text{False}$ such that the defined function symbol B is called during the computation of S under I; and

3.3. for all a-interpretations I for which $p_I(x_I) = \text{True}$ and $p_I(y_I) = \text{False}$, the defined function symbol B is called during the computation of S under I.

All our deterministic exponential lower time bounds in this paper follow directly from the properties of the class C.

In order to construct the class C, we show how to simulate the behavior of a linearly space-bounded DAPDM M of the form of Proposition 1.6 on an input $w \in \{00\} \cdot \{01, 10\}^* \cdot \{11\}$ by a recursion scheme M_w. In order to obtain the deterministic exponential lower time bound of Theorem 3.1, the size of M_w must be $\leqq k \cdot |w|$ for some constant k independent of w. The construction of M_w and a detailed discussion of how it simulates M on w appears in the proof of Theorem 3.1 and in the Appendix. The construction of M_w is complicated by the fact that we can only assume that calls

of its defined function symbols return single bits of information. This restriction is necessitated by our requirement that conditions 3.1, 3.2, and 3.3 be equivalent for M_w.

THEOREM 3.1. *There exist a class C of recursion schemes such that, for all* $S \in C$,

(i) *the only predicate symbol of S is the monadic predicate symbol p*;

(ii) *the only input variables of S are x and y*;

(iii) *B is a defined function symbol of S*;

(iv) *conditions* 3.1, 3.2, *and* 3.3 *are equivalent for S*; *and*

(v) *S is total.*

In addition, there exist constants $c, d > 0$ *such that the problem of determining for* $S \in C$ *if the defined function symbol B is called during some computation of S requires more than* $d \cdot 2^{c \cdot \|S\|}$ *steps infinitely often on any deterministic Turing machine.*

Proof. The proof is by explicit construction. Let M be a linearly space-bounded DAPDM such that

(a) M's input tape alphabet is $\{0, 1\}$;

(b) $L(M) \subset \{00\} \cdot \{01, 10\}^* \cdot \{11\}$;

(c) M halts for all inputs with empty pushdown store;

(d) for all inputs in $\{00\} \cdot \{01, 10\}^* \cdot \{11\}$, M never shifts its input tape head off its input tape;

and

(e) M pushes at most one stack symbol at a time.

Noting assumption (c) we also assume that

(f) M has only a single accepting state.

Thus, $M = (Q, \{0, 1\}, \Gamma, q_1, \{\vdash, \dashv\}, Z_0, F, \delta)$ where $m = |Q|$ and $F = \{q_m\}$.

We show that there is a constant $k > 0$ and a function f computable by a deterministic $O(n \log n)$ time-bounded Turing machine such that, for all $w \in \{00\} \cdot \{01, 10\}^* \cdot \{11\}$,

(g) $M_w = f(w)$ is a recursion scheme of size $\leqq k \cdot |w|$;

(h) $w \in L(M)$ if and only if B is called during some computation of M_w; and

(i) M_w satisfies (i)-(v) of the statement of Theorem 3.1.

The proof has three parts.

Part 1. Construction and explanation of the recursion scheme M_w: Let $n = |w|$. The defining statements of M_w are given in Fig. 1.

1. The defining statements of (1) and (2) guarantee that, for all a-interpretations I, either

1.1. $p_I(x_I) \neq$ True or $p_I(y_I) \neq$ False in which case the computation of M_w halts without calling B

or

1.2. $p_I(x_I) =$ True and $p_I(y_I) =$ False.

2. For each stack symbol Z of M, M_w has a defined function symbol F^Z. A call on F^Z during a computation of M_w under an a-interpretation I satisfying 1.2 corresponds to a point π in the computation of M on w when Z is the top stack symbol. The defined function symbol F^Z has $4n+2m+2$ parameters as follows—

$$F^Z(x, y, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m}).$$

If at point π, M is in state q_i, then formal parameter s_i is passed an actual parameter equal to x_I, otherwise s_i is passed an actual parameter equal to y_I. Similarly, parameter t_j is passed an actual parameter equal to x_I or to y_I if and only if the jth input tape cell of M at point π contains 1 or 0, respectively. Also parameter h_k is passed an actual parameter equal to x_I or to y_I if and only if at point π the input tape head of M is reading cell k, respectively. Hence, the state and input tape of M at point π are encoded into the value of the $2n+m$ parameters s_i for $1 \leqq i \leqq m$, t_j for $1 \leqq j \leqq n$, and

Defining Statements of M_w:

$z \coloneqq F(x, y)$ where

(1) $F(x, y) \coloneqq$ if $p(x)$ then $F_1(x, y)$ else x

$F_1(x, y) \coloneqq$ if $p(y)$ then x else $F_2(F^{Z_0}(x, y, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m}))$

where Z_0 is the bottom of stack marker of M;

$$s_j = \begin{cases} x & \text{if } j = 1, \text{ and} \\ y & \text{otherwise;} \end{cases}$$

$$t_k = \begin{cases} x & \text{if the } k\text{th symbol of } w \text{ is 1, and} \\ y & \text{if the } k\text{th symbol of } w \text{ is 0;} \end{cases}$$

$$h_1 = \begin{cases} x & \text{if } l = 1, \text{ and} \\ y & \text{otherwise; and} \end{cases}$$

$$r_p = \begin{cases} x & \text{if } p = m, \text{ and} \\ y & \text{otherwise.} \end{cases}$$

(2) $F_2(u) \coloneqq$ if $p(u)$ then $B(u)$ else u

$B(u) \coloneqq u$.

(3) For all $Z \in \Gamma$,

$F^Z(x, y, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

$\coloneqq$ if $p(s_1)$ then $F^{Z,q_1}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

else if $p(s_2)$ then $F^{Z,q_2}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

else $F^{Z,q_m}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$.

(4) For all $Z \in \Gamma$ and $q \in Q$,

$F^{Z,q}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

$\coloneqq \text{FINDHEAD}^{Z,q}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_n, x, r_1, \cdots, r_{2n+m})$

$\text{FINDHEAD}^{Z,q}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\coloneqq$ if $p(h_1)$ then if $p(t_1)$ then

$F^{Z,q,1}(x, y, t_2, \cdots, t_n, t_1, h_2, \cdots, h_n, h_{n+1}, h_1, r_1, \cdots, r_{2n+m})$

else $F^{Z,q,0}(x, y, t_2, \cdots, t_n, t_1, h_2, \cdots, h_n, h_{n+1}, h_1, r_1, \cdots, r_{2n+m})$

else $\text{FINDHEAD}^{Z,q}(x, y, t_2, \cdots, t_n, t_1, h_2, \cdots, h_n, h_{n+1}, h_1, r_1, \cdots, r_{2n+m})$.

(5) For all $Z \in \Gamma$, $q \in Q$, and $a \in \{0, 1\}$ where

$\delta(q, a, Z) = (p, b, \gamma, d)$,

(5.1) $F^{Z,q,a}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\coloneqq G^{\gamma,p,b,d}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$;

(5.2) $G^{\gamma,p,0,d}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\coloneqq G^{\gamma,p,d}(x, y, t_1, \cdots, t_{n-1}, y, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$ if $b = 0$, and

$G^{\gamma,p,1,d}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\coloneqq G^{\gamma,p,d}(x, y, t_1, \cdots, t_{n-1}, x, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$ if $b = 1$;

FIG. 1

(5.3) $G^{\gamma,p,0}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\quad := H^{\gamma,p}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$ if $d = 0$,

$G^{\gamma,p,1}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\quad := H^{\gamma,p}(x, y, t_2, \cdots, t_n, t_1, h_2, \cdots, h_n, y, x, r_1, \cdots, r_{2n+m})$ if $d = 1$, and

$G^{\gamma,p,-1}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\quad := H^{\gamma,p}(x, y, t_n, t_1, \cdots, t_{n-1}, y, h_1, \cdots, h_{n-1}, x, r_1, \cdots, r_{2n+m})$ if $d = -1$;

(5.4) $H^{\gamma,p}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\quad :=$ if $p(h_1)$ then $G^{\gamma,p}(x, y, t_1, \cdots, t_n, h_2, \cdots, h_{n+1}, r_1, \cdots, r_{2n+m})$

$\quad\quad$ else $H^{\gamma,p}(x, y, t_2, \cdots, t_n, t_1, h_2, \cdots, h_{n+1}, h_1, r_1, \cdots, r_{2n+m})$;

and

(5.5) $G^{\gamma,p}(x, y, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

$\quad := G^{\gamma}(x, y, q_1, \cdots, q_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

$$\text{where } q_i = \begin{cases} x & \text{if } i = p \text{ and} \\ y & \text{if } i \neq p. \end{cases}$$

(6) For each G^{γ} appearing in a defining statement of the form of (5.5),

(6.1) if γ is a single stack symbol A, then

$G^{A}(x, y, q_1, \cdots, q_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

$\quad := F^{A}(x, y, q_1, \cdots, q_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$;

(6.2) if γ is the empty string, then

$G^{\gamma}(x, y, q_1, \cdots, q_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

$\quad :=$ POP $(q_1, \cdots, q_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

POP $(v_1, \cdots, v_{2n+m}, r_1, \cdots, r_{2n+m})$

$\quad :=$ if $p(r_1)$ then v_1 else POP $(v_2, \cdots, v_{2n+m}, v_1, r_2, \cdots, r_{2n+m}, r_1)$;

and

(6.3) if $\gamma = BC$, then

$G^{BC}(x, y, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$

$\quad :=$ PUSH1^{BC} $(x, y, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m}, \underbrace{y, \cdots, y}_{2n+m \ y\text{'s}}, \underbrace{y, \cdots, y}_{2n+m-1 \ y\text{'s}}, x)$

PUSH1^{BC} $(x, y, u_1, \cdots, u_{2n+m}, r_1, \cdots, r_{2n+m}, v_1, \cdots, v_{2n+m}, c_1, \cdots, c_{2n+m})$

$\quad :=$ PUSH2^{BC} $(x, y, u_1, \cdots, u_{2n+m}, r_1, \cdots, r_{2n+m}, F^{C}(x, y, u_1, \cdots, u_{2n+m}, c_1, \cdots, c_{2n+m}),$

$\quad\quad v_2, \cdots, v_{2n+m}, c_1, \cdots, c_{2n+m})$,

and

PUSH2^{BC} $(x, y, u_1, \cdots, u_{2n+m}, r_1, \cdots, r_{2n+m}, v_1, \cdots, v_{2n+m}, c_1, \cdots, c_{2n+m})$

$\quad :=$ if $p(c_1)$ then $F^{B}(x, y, v_1, \cdots, v_{2n+m}, r_1, \cdots, r_{2n+m})$

$\quad\quad$ else PUSH1^{BC} $(x, y, u_1, \cdots, u_{2n+m}, r_1, \cdots, r_{2n+m}, v_{2n+m}, v_1, \cdots, v_{2n+m-1},$

$\quad\quad c_2, \cdots, c_{2n+m}, c_1)$.

FIG. 1 (*cont.*)

h_k for $1 \leqq k \leqq n$. Parameter r_l for $1 \leqq l \leqq 2n+m$ is passed an actual parameter x_I if the call on P^Z should return the lth bit of the encoding

$$s'_1, \cdots, s'_m, \quad t'_1, \cdots, t'_n, \quad h'_1, \cdots, h'_n$$

of the state and input tape of the configuration of M that occurs immediately after Z is popped off the stack. Otherwise, r_l is passed an actual parameter y_I.

Corresponding to a single push of Z by M, M_w calls F^Z $2n+m$ times. Each of these calls of F^Z is passed the values of the s, t, and h parameters that encode the state and input tape of M at the time Z is pushed on the stack. The kth call of F^Z has r_k equal to x_I and all other r parameters equal to y_I. The effect of the $2n+m$ calls of F^Z is to compute, bit by bit, the encoding of the state and input tape of the configuration of M that occurs immediately after Z is popped off the stack. (We sketch the proof of this in the Appendix.)

3. The defined function symbol F^Z determines the state, input tape head position, and input tape symbol being read at point π. This is accomplished by the defining statements of (3) and (4). The determination of which input tape symbol is being read is accomplished by the defining statement for FINDHEAD^{Z,q}. This is accomplished by simultaneous rotation of the t and h parameters until the first h parameter equals x_I, at which point the first t parameter encodes the input tape symbol being read at point π. (*Note* the additional h parameter added in the call of FINDHEAD^{Z,q}. This additional h parameter serves as a right endmarker.) When $F^{Z,q,0}$ or $F^{Z,q,1}$ is called (in the defining statement for FINDHEAD^{Z,q}), it is the last t and h parameters that represent the input tape cell being read and the input tape head position, respectively.

4. Let $\delta(q, a, Z) = (p, b, \gamma, d)$ be a state transition of M. The scheme M_w simulates this state transition by means of the defining statement of (5) and (6). The defining statements of (5.2) for $G^{\gamma,p,b,d}$ simulate the rewriting of the currently read input tape cell. The defining statements of (5.3) for $G^{\gamma,p,d}$ simulate the change of position of the input tape head. This is accomplished so that the t and h parameters, that represent the input tape cell being read and the input tape head position after the state transition, are simultaneously rotated to the ends of the t parameters and the h parameters, respectively. (See Fig. 2.)

Parameter Rotation in Defining Statements (5.3)

Case 1. $d = -1$.

Input tape		t_j parameters	h_k parameters
Before:	1101011	$yxxxxyx$	$yyx\underline{y}yy\underline{y}x$
After:	11$\underline{0}$1011	$xyxxxxy$	$yyyy\underline{x}yyx$

Case 2. $d = 1$.

Before:	1101011	$yxxxxyx$	$yyy\underline{x}yyyx$
After:	1101$\underline{0}$11	$xxxxyxy$	$yy\underline{x}yyyyx$

In the above figure $x = x_I$; $y = y_I$; the underlined tape cell is the tape cell being read; and the underlined h parameter is the "right endmarker".

FIG. 2

5. When $H^{\gamma,p}$ is called in the defining statements of (5.3), the "right endmarker" occurs earlier in the sequence of h parameters than that h parameter that represents the current input tape head position. (See Fig. 2.) Thus if the t and h parameters are simultaneously rotated as in the defining statement for $H^{\gamma,p}$, the first h_j encountered that equals x_I, equivalently such that $p_I(h_j) = \text{True}$, represents the "right endmarker".

The defining statement for $H^{\gamma,p}$ rotates the t and h parameters, finds the "right endmarker", and then dispenses with it. Upon dispensing with the "right endmarker", the t and h parameters of $G^{\gamma,p}$ encode the input tape of the configuration of M after execution of the state transition.

6. The call of G^{γ} in the defining statement for $G^{\gamma,p}$, defining statement (5.5), causes the q parameters of G^{γ} to correctly encode the state of M after execution of the state transition. Thus upon execution of the defining statement of (5.5), the parameters of G^{γ} correctly encode the state, tape contents, and tape head position of M after execution of the state transition.

7. The defining statements of (6) simulate the stack change made by M at point π. The length of the stack string γ is 0 (corresponding to a POP), 1 (corresponding to *no* change in stack height) or 2 (corresponding to a PUSH). The form of the defining statement of G^{γ} depends upon γ.

7.1. If γ is a single stack symbol A, the defined function F^{A} is called (see defining statement (6.1)).

7.2. If γ is the empty string λ, the appropriate bit of the encoded state and input tape are returned (see defining statement (6.2)).

7.3. If γ is BC where B replaces the old stack symbol and C is to be pushed above the B, the defined function symbol PUSH1^{BC} calls the defined function symbol F^{C} $2n+m$ times to compute the $2n+m$ bit encoding of the state and input tape of the configuration produced by processing stack symbol C. (See the defining statements of (6.3).) The function symbol PUSH1^{BC} has parameters u_i for $1 \leqq i \leqq 2n+m$ to record the state and input tape of the configuration at the time C is first pushed, parameters v_i for $1 \leqq i \leqq 2n+m$ to record the values returned by the series of calls on F^{C}, and parameters c_i for $1 \leqq i \leqq 2n+m$ to keep track of the number of calls on F^{C} and to ensure that each call returns the appropriate bit of the encoding of the state and input tape. (See Fig. 3.)

Parameter Rotations in Defining Statements (6.3)

Function called	with v parameters	and c parameters
PUSH1^{BC}	$y\,y\,y$	$y\,y\,x$
PUSH2^{BC}	$F_3^C\,y\,y$	$y\,y\,x$
PUSH1^{BC}	$y\,F_3^C\,y$	$y\,x\,y$
PUSH2^{BC}	$F_2^C\,F_3^C\,y$	$y\,x\,y$
PUSH1^{BC}	$y\,F_2^C\,F_3^C$	$x\,y\,y$
PUSH2^{BC}	$F_1^C\,F_2^C\,F_3^C$	$x\,y\,y$

Here, x denotes x_I, y denotes y_I, and F_i^C denotes the value returned by the call.

$$F^C(x, y, u_1, \cdots, u_{2n+m}, c_1^i, \cdots, c_{2n+m}^i)$$

where $c_i^i = x_I$ and $c_j^i \neq y_I$ for $j = i$. Thus

F_1^C denotes $F^C(x, y, u_1, u_2, u_3, x_I, y_I, y_I)$,

F_2^C denotes $F^C(x, y, u_1, u_2, u_3, y_I, x_I, y_I)$, and

F_3^C denotes $F^C(x, y, u_1, u_2, u_3, y_I, y_I, x_I)$.

FIG. 3

Part 2. Verification that M_w satisfies assertions (i)-(v) of the statement of the theorem:

By inspection of the defining statements of M_w, M_w satisfies assertions (i)-(iii). By inspection of the defining statements (1) and (2) if I is an interpretation such that $p_I(x_I) \neq$ True or $p_I(y_I) \neq$ False, then the computation of M_w under I halts without

calling B. The proof that M_w satisfies assertions (iv) and (v) follows from the manner in which the computation of M_w under an a-interpretation I such that

$$p_I(x_I) = \text{True and } p_I(y_I) = \text{False}$$

simulates the computation of M on w. This simulation is described as follows.

II.1. Let I be an a-interpretation such that $p_I(x_I) = \text{True}$ and $p_I(y_I) = \text{False}$. Let $\alpha = (q_i, \vdash x \dashv, A, l)$ and $\beta = (q_j, \vdash y \dashv, \lambda, k)$ be configurations of M such that $|x| = n$ and $\alpha \vdash_M^* \beta$. (Note by our definition of linearly space-bounded DAPDMs, at no earlier point in the computation of M from α to β is the stack empty.) Let

$$a = (F^A)_I(x_I, y_I, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$$

where $s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n$ is the encoding of the state and input tape of configuration α and exactly one of the r-parameters, say r_{i_0}, equals x_I and all other r-parameters equal y_I. Then a equals the i_0th entry in the encoding of the state and input tape of the configuration β.

(A proof of II.1 is sketched in the Appendix.)

Let $\alpha = (q_1, \vdash w \dashv, Z_0, 1)$. Let $\beta = (q, \vdash y \dashv, \lambda, t)$ be that configuration with empty stack such that $\alpha \vdash_M^+ \beta$. (By the properties of M such a configuration β exists.) Then $w \in L(M)$ if and only if $q = q_m$. By II.1 $q = q_m (q \neq q_m)$ if

$$F^{Z_0}(x_I, y_I, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m}) = x_I (= y_I),$$

where $s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n$ is the encoding of the state and input tape of the configuration α, $r_m = x_I$, and all other r parameters equal y_I. Hence by inspection of the defining statements of M_w as described in Part 1 of this proof, B is called during the computation of M_w under I if and only if $w \in L(M)$. Additionally, the computation of M_w under I halts since the computation of M on w halts by assumption.

Part 3. Definition of the class C and verification of lower time bound:

Let N be a linearly space-bounded DAPDM satisfying conditions 1-6 of the statement of Proposition 1.6. Without loss of generality we assume that N has m states where q_1 is N's start state and q_m is N's single accepting state. By inspection of the defining statements in Fig. 1 for $M = N$, there exists a constant k depending upon N but *not* on w such that, for all $w \in \{00\} \cdot \{01, 10\}^* \cdot \{11\}$, $|N_w| \leqq k \cdot |w|$. Also by inspection of the defining statements in Fig. 1, there is a deterministic $O(n \log n)$ time-bounded Turing machine that, given input $w \in \{00\} \cdot \{01, 10\}^* \cdot \{11\}$, outputs N_w. By the argument of Part 2 for all $w \in \{00\} \cdot \{01, 10\}^* \cdot \{11\}$, $w \in L(N)$ if and only if B is called during some computation of N_w. Thus there exist $c, d > 0$ such that the problem of determining for $S \in C$ if the defined function symbol B is called during some computation of S requires more than $d \cdot 2^{c \cdot \|S\|}$ steps infinitely often on any deterministic Turing machine. Otherwise, the recognition of the language $L(N)$ would not require deterministic exponential time, contradicting Proposition 1.6. □

4. Exponential lower deterministic time bounds. Theorem 3.1 and its proof easily imply exponential lower deterministic time bounds for a variety of decision problems for very simple recursion schemes for any class **I** of a-interpretations such that

$$(\exists J \in \mathbf{I})[p_J(x_J) = \text{True and } p_J(y_J) = \text{False}]. \tag{4.0}$$

Any recursive programming language with a nontrivial predicate can be viewed as such a class **I** of a-interpretations. Thus Theorem 3.1 and its proof also easily imply exponential lower deterministic time bounds for a variety of decision problems for very simple programs in recursive programming languages with a nontrivial predicate.

We present examples of such exponential lower deterministic time bounds for decision problems for very simple recursion schemes in the first six results of this section. Each of these exponential lower deterministic time bounds follows easily from Theorem 3.1 and its proof.

THEOREM 4.1. *Let* **I** *be any class of a-interpretations satisfying condition* (4.0). *Then there exist constants* $c, d > 0$ *such that the recognition of each of the following sets requires more than*

(i) $d \cdot 2^{c \cdot (\|S\| + \|T\|)}$ *steps for the sets of* 1 *and*

(ii) $d \cdot 2^{c \cdot \|S\|}$ *steps for the sets of* 2 *and of* 3

infinitely often on any deterministic Turing machine:

1. *for all binary relations* ρ *on* R *between* $=$ *and* $\simeq_{\mathbf{I}}$, $\{S, T \in C \mid S \rho T\}$ *and* $\{S, T \in R \mid S \rho T\}$,

2. $\{S \in R \mid S$ *halts for all a-interpretations in* $\mathbf{I}\}$, *and*

3. $\{S \in R \mid S$ *diverges for all a-interpretations in* $\mathbf{I}\}$.

Proof. 1. For all $M_w \in C$, let M_w^1 and M_w^2 be defined as M_w except that the defining statement for F_1 is replaced by

$$F_1(x, y) \coloneqq \text{if } p(y) \text{ then } x \text{ else}$$
$$F_2(x, y, F^{Z_0}(x, y, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m}))$$

in both of M_w^1 and M_w^2, and that the defining statements for F_2 and B are replaced by

$$F_2(u, v, w) \coloneqq \text{if } p(w) \text{ then } B(u, v, w) \text{ else } u,$$
$$B(u, v, w) \coloneqq u$$

in M_w^1 and are replaced by

$$F_2(u, v, w) \coloneqq \text{if } p(w) \text{ then } B(u, v, w) \text{ else } u,$$
$$B(u, v, w) \coloneqq v$$

in M_w^2. Then, for all interpretations I, the following are equivalent:

(a) B is called during the computation of M_w under I;

(b) B is called during the computation of M_w^1 under I; and

(c) B is called during the computation of M_w^2 under I.

Thus, $M_w^1 = M_w^2$, if B is not called during some computation of M_w, and $\sim(M_w^1 \simeq_{\mathbf{I}} M_w^2)$, otherwise.

2. For all $M_w \in C$, let M_w^1 be defined as M_w except that the defining statement for B is replaced by

$$B(u) \coloneqq B(u).$$

Then, for all interpretations I, the computation of M_w^1 under I halts if and only if B is not called during the computation of M_w under I. Thus M_w^1 halts for all interpretations I in $\mathbf{I}$ if and only if B is not called during some computation of M_w.

3. For all $M_w \in C$, let M_w^1 be defined as M_w except that the defining statements for F, F_1, and F_2 are replaced by

$$F(x, y) \coloneqq \text{if } p(x) \text{ then } F_1(x, y) \text{ else } F(x, y),$$
$$F_1(x, y) \coloneqq \text{if } p(y) \text{ then } F_1(x, y) \text{ else}$$
$$F_2(F^{Z_0}(x, y, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})),$$

and

$$F_2(u) \coloneqq \text{if } p(u) \text{ then } u \text{ else } F_2(u).$$

Then, for all interpretations I, the computation of M_w^1 under I diverges if and only if B is not called during the computation of M_w under I. Thus M_w^1 diverges for all interpretations I in $\mathbf{I}$ if and only if B is not called during some computation of M_w. □

COROLLARY 4.2. *Let ρ be any binary relation on R between $=$ and $\simeq_2$. Then there exist constants $c, d > 0$ such that the recognition of the sets*

$$\{S, T \in C | S \rho T\} \quad \text{and} \quad \{S, T \in R | S \rho T\}$$

requires more than $d \cdot 2^{c \cdot (\|S\| + \|T\|)}$ steps infinitely often on any deterministic Turing machine. These binary relations ρ include all binary relations τ and, for all $k \geqq 2$, τ_k for $\tau \in \{=, \equiv, \subseteq, \simeq\}$.

Proof. The corollary follows immediately from Theorem 4.1 since the class of all a-interpretations with domains of cardinality 2 satisfies condition (4.0). □

THEOREM 4.3. *Let $\mathbf{I}$ be any class of a-interpretations that includes an a-interpretation J that satisfies condition* (4.0). *Let S_0 be any fixed recursion scheme with input variables $x_1, \cdots, x_k$ $(k \geqq 2)$ whose computation under J terminates. Let ρ be any binary relation on R between $\equiv$ and $\simeq_{\mathbf{I}}$. Then there exist constants $c, d > 0$ such that the recognition of the set $\{S \in R | S \rho S_0\}$ requires more than $d \cdot 2^{c \cdot \|S\|}$ steps infinitely often on any deterministic Turing machine.*

Proof. Without loss of generality we assume that the variables x and y are input variables of S_0, i.e. $x, y \in \{x_1, \cdots, x_k\}$. For all $M_w \in C$, let M_w^1 be the recursion scheme that results from S_0 and M_w as shown in Fig. 4. By inspection of the defining statements of (4.1) of Fig. 4, for all a-interpretations I not satisfying condition (4.0), either both z_I and z'_I are undefined or $z_I = z'_I = G_I((x_1)_I, \cdots, (x_k)_I)$. Let I be an a-interpretation in $\mathbf{I}$ that satisfies condition (4.0). By assumption at least one such a-interpretation J exists for which the computation of S_0 under J terminates. By inspection of the defining statements of (4.1) and (4.2) of Fig. 4 and by the proof of Theorem 3.1, the following two assertions hold.

The scheme S_0:

$$z = G(x_1, \cdots, x_l) \text{ where}$$

(4.0) {Defining statements of S_0}

The scheme M_w^1:

$$z' = F(x_1, \cdots, x_k) \text{ where}$$

(4.1) $F(x_1, \cdots, x_k) \coloneqq \text{if } p(x) \text{ then } F_1(x_1, \cdots, x_k) \text{ else } G(x_1, \cdots, x_k)$

$F_1(x_1, \cdots, x_k) \coloneqq \text{if } p(y) \text{ then } G(x_1, \cdots, x_k)$

$\text{else } F_2(x_1, \cdots, x_k, F^{Z_0}(x, y, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m}))$

(4.2) $F_2(x_1, \cdots, x_k, u) \coloneqq \text{if } p(u) \text{ then } B(x, y, G(x_1, \cdots, x_k)) \text{ else } G(x_1, \cdots, x_k)$

$B(u, v, w) \coloneqq \text{if } p(w) \text{ then } v \text{ else } u$

(4.3) {Defining statements of (3) through (6) of M_w}

(4.4) {Defining statements of S_0}

FIG. 4

(i) If B is not called during a computation of M_w, then B is not called during the computations of M_w and of M_w^1 under I. Thus, either both z_I and z'_I are undefined or $z_I = z'_I = G_I((x_1)_I, \cdots, (x_k)_I)$.

(ii) If B is called during a computation of M_w, then B is called during the computations of M_w and of M_w^1 under I. Thus, either

(a) both z_I and z'_I are undefined,

(b) $p_I(G_I((x_1)_I, \cdots, (x_k)_I)) = \text{True}$ and $z'_I = y_I$, or

(c) $p_I(G_I((y_1)_I, \cdots, (x_k)_I)) = \text{False}$ and $z'_I = x_I$.

Thus if B is not called during some computation of M_w, then $M^1_w \equiv S_0$. Otherwise, the computations of M^1_w and of S_0 under J both terminate but $p_J(z'_J) \neq p_J(z_J)$. Hence $z'_J \neq z_J$ and, thus $\sim(M^1_w \simeq_1 S_0)$. □

THEOREM 4.4. *Let p be a monadic predicate symbol; let x and y be distinct input variable symbols; and let f be a monadic basis function symbol. Let $m \geqq 2$; and let $x_1, \cdots, x_m$ be input variable symbols where $x, y \in \{x_1, \cdots, x_m\}$. Let $P(x_1, \cdots, x_m)$ and $Q(x_1, \cdots, x_m, z)$ be uninterpreted input and output predicates, respectively. Let I_0 be any a-interpretation such that*

1. $P_{I_0}((x_1)_{I_0}, \cdots, (x_m)_{I_0}) = \text{True}$,
2. $p_{I_0}(x_{I_0}) \neq p_{I_0}(y_{I_0})$, *and*
3. $Q_{I_0}((x_1)_{I_0}, \cdots, (x_m)_{I_0}, f_{I_0}(x_{I_0})) \neq Q_{I_0}((x_1)_{I_0}, \cdots, (x_m)_{I_0}, f_{I_0}(y_{I_0}))$.

Then there exist constants $c, d > 0$ such that the recognition of each of the sets

(i) $\{S \in R \mid S$ *is partially correct with respect to P, Q, and I_0*$\}$ *and*

(ii) $\{S \in R \mid S$ *is totally correct with respect to P, Q, and I_0*$\}$

requires more than $d \cdot 2^{c\|S\|}$ steps infinitely often on any deterministic Turing machine.

Proof. Without loss of generality we assume that $p_{I_0}(x_{I_0}) = \text{True}$. For all $M_w \in C$, let M^1_w be the recursion scheme that results from M_w by replacing the defining statement for F_2 in (2) of Fig. 1 by

$$F_2(u) := f(u).$$

By inspection of the defining statement for F_1 in (1) of Fig. 1, F_2 is called during the computations of M_w and of M^1_w under I_0. Moreover by the proof of Theorem 3.1 if B is called during the computation of M_w under I_0, then the value of the parameter u when F_2 is called during the computation of M^1_w under I_0 is x_{I_0}. Otherwise, the value of the parameter u when F_2 is called during the computation of M^1_w under I_0 is y_{I_0}. Thus $\text{Val}_{I_0}(M^1_w) = f_{I_0}(x_{I_0})$, if B is called during the computation of M_w under I_0; and $\text{Val}_{I_0}(M^1_w) = f_{I_0}(y_{I_0})$, otherwise. By assumption one of $Q_{I_0}((x_1)_{I_0}, \cdots, (x_m)_{I_0}, f_{I_0}(x_{I_0}))$ and $Q_{I_0}((x_1)_{I_0}, \cdots, (x_m)_{I_0}, f_{I_0}(y_{I_0}))$ equals True and the other equals False. □

COROLLARY 4.5. *Let $p, f, x, y, x_1, \cdots, x_m, P, Q$, and I_0 be as in the statement of Theorem 4.4. Let D be the class of recursion schemes S such that*

1. *the input variables of S are $x_1, \cdots, x_m$,*
2. *the only predicate symbol of S is p,*
3. *there are no occurrences of constant symbols in S,*
4. *the only basis function symbol of S is f, and*
5. *there is only one occurrence of the basis function symbol f in S.*

Then there exist constants $c, d > 0$ such that the recognition of each of the sets

(i) $\{S \in D \mid S$ *is partially correct with respect to P, Q, and I_0*$\}$ *and*

(ii) $\{S \in D \mid S$ *is totally correct with respect to P, Q, and I_0*$\}$

requires more than $d \cdot 2^{c\|S\|}$ steps infinitely often on any deterministic Turing machine.

Proof. For all $M_w \in C$, the recursion scheme M^1_w of the proof of Theorem 4.4 is an element of D. □

Theorem 4.4 and Corollary 4.5 show how Theorem 3.1 and its proof can be used to derive exponential lower deterministic time bounds for the partial and total correctness problems for very simple recursion schemes for many different uninterpreted input and output predicates P and Q. The next proposition shows how simple such uninterpreted input and output predicates can be.

PROPOSITION 4.6. *Let p be a monadic predicate symbol, and let x and y be distinct variable symbols. Let I_0 be any a-interpretation such that $p_{I_0}(x_{I_0}) = \text{True}$ and $p_{I_0}(y_{I_0}) =$*

False. Let $P(x, y)$ be the uninterpreted input predicate

$$p(x) \wedge \sim p(y).$$

Let $Q(z)$ be the uninterpreted output predicate

$$p(z).$$

Then there exist constants $c, d > 0$ such that the recognition of each of the sets

(i) $\{S \in C \mid S$ *is partially correct with respect to* P, Q, *and* $I_0\}$ *and*

(ii) $\{S \in C \mid S$ *is totally correct with respect to* P, Q, *and* $I_0\}$

requires more than $d \cdot 2^{c \cdot \|S\|}$ steps infinitely often on any deterministic Turing machine.

Proof. The proof closely follows that of Theorem 4.4. For all $M_w \in C$, let M_w^1 be the recursion scheme that results from M_w by replacing the defining statement for F_2 in (2) of Fig. 1 by $F_2(u) := u$. Then $\mathrm{Val}_{I_0}(M_w^1)$ equals x_{I_0}, if B is called during the computation of M_w under I_0, and equals y_{I_0}, otherwise. □

The simplicity of the recursion schemes and the generality of the classes **I** of a-interpretations of results 4.1 through 4.6 enable us to derive analogous lower time bounds for decision problems for very simple programs in any recursive programming language with a nontrivial predicate test. Such decision problems include the totality, divergence, computational identity, strong equivalence, weak equivalence, and containment problems as well as the partial and total correctness problems for many fixed input and output predicates. For example, the following theorem is an easy corollary of Theorem 4.1 and its proof.

THEOREM 4.7. *Let L be any recursive programming language with a nontrivial predicate test π. Let C' be the class of all programs P in L such that*

(a) *the only predicate occurring in P is π, and*

(b) *no basis functions or constants appear in P.*

Then there exist constants $c, d > 0$ such that the recognition of each of the following sets requires more than

(i) $d \cdot 2^{c \cdot (\|P\| + \|Q\|)}$ *steps for the sets of* 1 *and*

(ii) $d \cdot 2^{c \cdot \|P\|}$ *steps for the sets of* 2 *and of* 3

infinitely often of any deterministic Turing machine:

1. *for all binary relations ρ on L between $=$ and $\simeq$, $\{P, Q \in C' \mid P \rho Q\}$,*
2. *$\{P \in C' \mid P$ is total$\}$, and*
3. *$\{P \in C' \mid P$ is divergent$\}$.*

Additionally, Theorem 3.1 and its proof also imply exponential lower deterministic time bounds for program testing for many fixed inputs for very simple programs in any recursive programming language L with a nontrivial predicate test π. Finally with a little additional information about a recursive programming language L many additional exponential lower deterministic time bounds can be obtained. For example, the following theorem holds.

THEOREM 4.8. *Let L be any recursive programming language with a monadic predicate p and constants a and b such that $p(a) \neq p(b)$. Let π be any predicate on the set of partial functions computed by the programs of L for which*

there exist programs P_1 and P_2 of L
each with $i \geqq 0$ input variables, such that
$\pi(F_{P_1}) \neq \pi(F_{P_2})$.

Then there exist constants $c, d > 0$ such that the problem of determining, for $P \in L$, if $\pi(F_P) =$ True requires more than $d \cdot 2^{c \cdot \|P\|}$ steps infinitely often on any deterministic Turing machine.

Proof. For simplicity we assume that $p(a) = \pi(F_{P_1}) = \text{True}$. Let G_0 and H_0 be the initial defined functions of P_1 and P_2, respectively. For all $M_w \in C$, let M_w^1 be the program in L that results from M_w as follows:

1. The defining statements of (1) and (2) of Fig. 1 are replaced by

$$z := F(x_1, \cdots, x_i) \text{ where}$$
$$F(x_1, \cdots, x_i) := F_1(x_1, \cdots, x_i, F^{Z_0}(a, b, s'_1, \cdots, s'_m, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n, r'_1, \cdots, r'_{2n+m}))$$

where

$$s'_j = \begin{cases} a & \text{if } j = 1, \text{ and} \\ b & \text{otherwise,} \end{cases}$$

$$t'_k = \begin{cases} a & \text{if the } k\text{th symbol of } w \text{ is 1, and} \\ b & \text{if the } k\text{th symbol of } w \text{ is 0,} \end{cases}$$

$$h'_l = \begin{cases} a & \text{if } l = 1, \text{ and} \\ b & \text{otherwise,} \end{cases}$$

and

$$r'_p = \begin{cases} a & \text{if } p = m, \text{ and} \\ b & \text{otherwise.} \end{cases}$$

$$F_1(x_1, \cdots, x_i, u) := \text{if } p(u) \text{ then } G_0(x_1, \cdots, x_i) \text{ else } H_0(x_1, \cdots, x_i).$$

2. All occurrences of x and y on the right-hand sides of the defining statements of (3) through (6) of Fig. 1 are replaced by a and b, respectively.

3. The defining statements of P_1 and P_2 are added after renaming defined function symbols as necessary to prevent name duplications. Then, $F_{M_w^1} = F_{P_1}$, if B is called during a computation of M_w; and $F_{M_w^1} = F_{P_2}$, otherwise. □

The results of (4.1) through (4.8) enable us to answer the question 1.0 above as follows:

Recursive program analysis due *only* to recursion and predicate tests requires deterministic exponential time.

Finally, there are recursive programming languages with nontrivial predicate tests for which such decision problems as divergence, totality, strong equivalence, etc. are decidable deterministically in exponential-time, e.g. the language FMP^{REC} in [10]. Thus, our answer is the best possible answer in general.

5. Conclusion. PSPACE-hard lower bounds for problems concerning procedure-valued parameters in recursive programs appears in [16]. Deterministic exponential lower time bounds for the divergence and strong equivalence problems for a particular simple programming language with recursion appear in [10]. Our results generalize the results in [10] in three important ways. First, our lower time bounds hold for pure recursive programming languages. The language FMP^{REC} in [10] is a flowchart programming language augmented with recursion. Second, our lower time bounds hold for very simple programs in *any* recursive programming language with a nontrivial predicate test. Third, our lower time bounds hold for the executability problem and, thus, are applicable to many other decision problems besides divergence and strong equivalence. These decision problems include computational identity, isomorphism, totality, and partial and total correctness.

It is pointless to include predicate tests as a language feature if the only predicate tests allowed are identically true or identically false. Thus, we have shown that deterministic exponential lower time bounds hold for analyzing very simple programs in recursive programming languages. Avoiding these lower bounds by imposing restrictions on the predicates and basis functions in the programming language can only be done by restricting all predicates to be trivial, which is unreasonable for any language with tests. Our constructions, however, do assume that there is no restriction on the number of parameters of defined functions.

Intuitively, *any* system in which recursion is used seems to require a predicate to stop the recursion, where the predicate is applied to a parameter or other data object. Furthermore in the recursive call when the recursion stops, the predicate produces a different truth value than in preceding calls and, thus, is *nontrivial* in the sense of this paper. This suggests that deterministic exponential lower time bounds may be intrinsic in the analysis of more general classes of recursively defined and/or presented objects.

Appendix. Proof of II.1. The proof is by induction on the number k of steps in the computation of M from α to β. Let the first state transition τ executed during this computation be $\delta(q, a, Z) = (p, b, \gamma, d)$. By the discussions in 2-6 of Part 1 of the proof of Theorem 3.1,

$$(F_A)_I(x_I, y_I, s_1, \cdots, s_m, t_1, \cdots, t_n, h_1, \cdots, h_n, r_1, \cdots, r_{2n+m})$$
$$= (G^\gamma)_I(x_I, y_I, s'_1, \cdots, s'_m, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n, r_1, \cdots, r_{2n+m})$$

where $s'_1, \cdots, s'_m, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n$ is the encoding of the state and input tape of the configuration η of M that follows from α by a single application of the state transition τ.

If $k = 1$, then $\beta = \eta$ and β follows from α by a single POP move. Thus $\gamma = \lambda$. By the defining statement (6.3) of Fig. 1, a equals the (i_0+2)th parameter of the call of $(G^\lambda)_I$. Thus, a equals the i_0th entry in the encoding of the state and input tape of the configuration β.

Now suppose claim II.1 is true for all computations with at most $k \geqq 1$ steps. Suppose that $\alpha \vdash_M^{k+1} \beta$. Thus

$$\alpha \vdash_M^1 \eta \vdash_M^k \beta.$$

Since $k+1 > 1$, $\gamma \neq \lambda$. Thus, either $\gamma = B$ or $\gamma = BC$ for some stack symbols B and C.

If $\gamma = B$, then by the defining statement (6.1) of Fig. 1,

$$a = (G^B)_I(x_I, y_I, s'_1, \cdots, s'_m, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n, r_1, \cdots, r_{2n+m})$$
$$= (F^B)_I(x_I, y_I, s'_1, \cdots, s'_m, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n, r_1, \cdots, r_{2n+m}),$$

which by the induction hypothesis equals the i_0th entry in the encoding of the state and input tape of configuration β.

If $\gamma = BC$ there is a configuration ξ such that $\eta \vdash_m^i \xi \vdash_M^j \beta$, $i+j = k$, the stack of η equals BC, the stack of ξ equals B, and at no earlier point in the computation from η to ξ is the stack equal to B. Define

$$z_i = (F^C)_I(x_I, y_I, s'_1, \cdots, s'_n, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n, c^i_1, \cdots, c^i_{2n+m})$$

where c^i_i equals x_I and c^i_j equals y_I for $j \neq i$. Since $s'_1, \cdots, s'_m, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n$ is the encoding of the state and input tape of the configuration η, the induction hypothesis implies that z_i equals the ith entry in the encoding of the state and the input tape of configuration ξ. Thus $z_1, \cdots, z_{2n+m}$ is the encoding of the state and input

tape of configuration ξ. By the defining statements (6.3) of Fig. 1,

$$\begin{aligned} a &= (G^{BC})_I(x_I, y_I, s'_1, \cdots, s'_n, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n, r_1, \cdots, r_{2n+m}) \\ &= (\mathrm{PUSH1}^{BC})_I(x_I, y_I, s'_1, \cdots, s'_n, t'_1, \cdots, t'_n, h'_1, \cdots, h'_n, r_1, \cdots, r_{2n+m}) \\ &= (F^B)_I(x_I, y_I, z_1, \cdots, z_{2n+m}, r_1, \cdots, r_{2n+m}). \end{aligned}$$

Since $\xi \vdash_M^j \beta$ where $j \leqq k$, the induction hypothesis implies that a equals the i_0th entry of the encoding of the state and the input tape of the configuration β.

REFERENCES

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

[2] J. C. CHERNIAVSKY, *Simple programs realize exactly Presburger formulas*, this Journal, 4 (1976), pp. 666-677.

[3] S. A. COOK, *Characterizations of pushdown machines in terms of time-bounded computers*, J. Assoc. Comput. Mach., 18 (1971), pp. 4-18.

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

[5] S. J. GARLAND AND D. C. LUCKHAM, *Program schemes, recursion schemes, and formal languages*, J. Comput. System Sci., 7 (1973), pp. 119-160.

[6] S. A. GREIBACH, *Theory of Program Structures: Schemes, Semantics, Verification*, Lecture Notes on Computer Science 36, G. Goos and J. Hartmanis, eds., Springer-Verlag, Berlin, 1975.

[7] J. E. HOPCROFT AND J. D. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, MA, 1969.

[8] H. B. HUNT, III, *On the complexity of flowchart and loop program schemes and programming languages*, J. Assoc. Comput. Mach., 29 (1982), pp. 228-249.

[9] H. B. HUNT, III AND D. J. ROSENKRANTZ, *The complexity of monadic recursion schemes: exponential time bounds*, J. Comput. System Sci., 28 (1984), pp. 395-419.

[10] N. D. JONES AND S. S. MUCHNICK, *The complexity of finite memory programs with recursion*, J. Assoc. Comput. Mach., 25 (1978), pp. 312-321.

[11] D. C. LUCKHAM, D. M. R. PARK AND M. S. PATERSON, *On formalized computer programs*, J. Comput. System Sci., 4 (1969), pp. 220-249.

[12] Z. MANNA, *Mathematical Theory of Computation*, McGraw-Hill, New York, 1974.

[13] A. R. MEYER AND D. M. RITCHIE, *The complexity of loop programs*, Proc. 22nd National ACM Conference, Washington, DC, 1967, pp. 465-470.

[14] N. SUZUKI AND D. JEFFERSON, *Verification decidability of Presburger array programs*, J. Assoc. Comput. Mach., 27 (1980), pp. 191-205.

[15] D. TSICHRITZIS, *The equivalence problem of simple programs*, J. Assoc. Comput. Mach., 17 (1970), pp. 729-738.

[16] K. WINKLMANN, *On the complexity of some problems concerning the use of procedures*, Acta Informatica, *Part* I, 18 (1982), pp. 299-318; *Part* II, 18 (1983), pp. 411-430.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986

© 1986 Society for Industrial and Applied Mathematics

A BINARY SEARCH WITH A PARALLEL RECOVERY OF THE BITS*

BENJAMIN ARAZI†

Abstract. In this paper we consider the problem of assigning a numerical value of each of N elements which are arranged in a linear array, for the purpose of being able to recover the location of any specified element (with respect to the first element), with this aim being achieved using a small number of possible different assigned values. An obvious solution is using a binary search approach, characterized by the features: (a) At least $[\log_2 N]+2$ different values are needed for implementing the process. (b) The bits forming the binary representation of the specified location are recovered sequentially (starting with the least significant bit).

A new approach in assigning values to elements, for the purpose of recovering efficiently their location, is presented in this paper, for the case where N is of the form 2^n-1. Its main features are (a) less than $[\log_2 N]+2$ different values are sufficient for achieving this task (even 2 are enough!). (b) The bits forming the binary representation of the specified location are recovered in parallel. This is achieved by very simple means, where the value of each bit in the binary representation is recovered by considering the numerical values assigned to 4 elements having fixed location with respect to the element whose location is to be found.

Key words. location recovery in ordered sets, binary search, modular arthmetic, finite fields

AMS(MOS) subject classifications. 11T21, 12J15, 11S20

1. General. Consider the problem of assigning different numerical values to N elements arranged in a linear array, for the purpose of being able to recover the location of any specified element (with respect to a defined starting point), based on the numerical value assigned to it and the numerical values assigned to a small number of other elements whose relative location with respect to the specified one are well defined. This task has to be carried out using a small number of possible different values. (The problem is trivially solved if we are allowed to use N different values.) An obvious way for performing the described task would be to assign one unique value to all the elements having an odd location index. Another unique value will be assigned to all the elements having an index congruent to 2, when taken modulo 4, and in general: all elements with location index X, for $X \bmod 2^i \equiv 2^{i-1}$, are assigned a unique value, $i=1, 2, \ldots, [\log_2 N]+1$. An additional unique value should be assigned to the first element (location index 0). As an example take the case where $N=13$. Assigning respectively the values A, B, C, D for $i=1, 2, 3, 4$, and assigning E to location 0 yields the following structure.

0	1	2	3	4	5	6	7	8	9	10	11	12
E	*A*	*B*	*A*	*C*	*A*	*B*	*A*	*D*	*A*	*B*	*A*	*C*.

Specifying any element, the parity of its location index I is instantly recovered from the value assigned to it. If I is even, then $I \bmod 4$ is also recovered. Otherwise, $I \bmod 4$ is recovered from the value assigned to the element in the $I-1$ place. The process then continues by going to the element in the $(I-1)-2$ place (if and when needed), etc, until all the bits in the binary representation of I are recovered.

DEMONSTRATION 1. The location of the element underlined in the preceding demonstrated structure (of 13 elements) is recovered as follows.

Least significant bit: Since the value assigned to the element is A, I is odd. (LSB is 1) Go now to location $I-1$.

* Received by the editors March 27, 1984, and in revised form July 17, 1985.

† Department of Electrical and Computer Engineering, Ben Gurion University, Beer Sheva, Israel.

More significant bit: The value B in location $I-1$ means that $(I-1) \bmod 4 = 2$. The more significant bit in the binary representation of I is then 1. Go now to location $(I-1)-2$, etc.

The way just described by which the bits of the binary representation of I are recovered, requires a reference to a table which indicates which value was assigned to any of the described $[\log_2 N]+2$ sets of indices. However, it is possible to recover these bits without knowing which specific value was assigned to each set of indices. If I is the location whose value is to be recovered, note that I is odd iff the value assigned to locations I and $I-2$ is the same (for $I-2>0$). If I is even then the next bit in its binary representation is 1 iff the values assigned to locations I and $I-4$ are the same. If I is odd compare the values assigned to locations $I-1$ and $I-5$, etc.

The described process (either the recovery based on specific values or the recovery based on comparing values) is characterized by: (a) $[\log_2 N]+2$ different values are needed for its implementation. (b) The process is sequential. For example, the more significant bits in the binary reprsentation of I are recovered only after recovering the least significant ones.

In this paper we treat the following problems.

(a) Is it possible to assign numerical values to the elements such that the bits in the binary representation of I can be recovered in parallel? (For example, each bit is recovered independently, and simultaneously with the other.)

(b) Is it possible to recover I by assigning less than $[\log_2 N]+2$ different values to the N elements? (The recovery will then be based, of course, on value comparisons.)

(c) If the answer to any of the above questions is positive, what is the trade-off between the ability to perform the described task and the complexity of performing it?

It is shown in this paper that a parallel recovery of the bits of I is possible and can be performed easily even if only two different values are used. The penalty paid is the necessity of dealing with the values assigned to 4 elements when recovering each bit, rather than comparing 2 values when each bit is recovered using the standard binary search described above. The offered solution is valid for the case where N is of the form 2^n-1.

It should be clarified that for a given n there is a known way of constructing sequences up to length 2^n having the property that the patterns consisting of n consecutive elements in such a sequence are all different. These sequences which also include the binary case are the DeBruijn sequences [1]. A DeBruijn sequence can then be used for the purpose of recovering the location of elements in an array by assigning the values of such a sequence, in order, to the elements.

Given then the values assigned to n consecutive elements, it is possible in principle to recover their location since this combination of n values is unique. However, given a pattern consisting of n consecutive elements of a DeBruijn sequence there is not a known algorithm with polynomial complexity for recovering their location with respect to a reference point. The sequences presented in this paper (which also include the binary case) have the property that the algorithm for recovering the location of elements based on their values is of complexity $O(n)$ (for 2^n-1 being the length of the sequence).

2. Theory.

2.1. Constructions based on cyclotomic cosets.

DEFINITION. The cyclotomic coset modulo 2^n-1 of an integer $x \in [0, 2^n-2]$ consists of all the distinct numbers from the set $\{x, 2x \bmod (2^n-1), 4x \bmod (2^n-1), \cdots, 2^{n-1}x \bmod (2^n-1)\}$.

DEMONSTRATION 2. The cyclotomic cosets modulo 31 are:

$$\{0\},\ \{1, 2, 4, 8, 16\},\ \{3, 6, 12, 24, 17\},\ \{5, 10, 20, 9, 18\},\ \{7, 14, 28, 25, 19\},$$
$$\{11, 22, 13, 26, 21\},\ \{15, 30, 29, 27, 23\}.$$

DEFINITION. A sequence B of length m is obtained from a sequence A of the same length by h-decimation of A, if B is constructed by taking everty hth element of A and repeating the process cyclically (the last element is considered to be followed by the first one) until m elements are obtained.

It should be noted that if $(n, m) = 1$, the elements of B consist of all the elements A. If $(h, m) > 1$ then B consist of repetitions of certain elements of A. This means, as a special case, that if the length of A is odd and $h = 2$, then B consists of all the elements of A.

Notation. C_n denotes an ordered sequence of $2^n - 1$ numerical values, where the same values are assigned to locations whose indices belong to the same cyclotomic coset modulo $2^n - 1$.

The proof of the following theorem is self explanatory and is omitted.

THEOREM 1. *The sequence obtained by 2^i-decimation of C_n starting with location X, equals the sequence obtained by scanning C_n continuously (1-decimation) starting with location $(X/2^i) \bmod (2^n - 1)$, for any $0 \leqq X \leqq 2^n - 2$ and $0 \leqq i \leqq n - 1$.*

DEMONSTRATION 3. The general structure of C_5 (based on the cyclotomic cosets listed in demonstration 2) is:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	*B*	*B*	*C*	*B*	*D*	*C*	*E*	*B*	*D*	*D*	*E*	*C*	*E*	*E*	*F*

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
B	*C*	*D*	*E*	*D*	*E*	*E*	*F*	*C*	*E*	*E*	*F*	*E*	*F*	*F*.

4-decimating the sequence, starting with location $X = 13$ for example, yields the sequence $ECEEFBC\cdots$. This same sequence is obtained by scanning C_5 continuously, starting with location $(13/4) \bmod 31 = 11$.

Remarks. (a) According to the definition of C_n, the values $A, B, C, D\cdots$ are not necessarily different.

(b) Starting with $X = 0$, the 2^i-decimated sequence is identical to the continuous sequence starting at the same point. If C_n is arranged in a circular form, such that its starting point is not indicated, the starting point can be defined as being the one starting with which all the sequences obtained by 2^i-decimation $i = 0, 1, \cdots, n - 1$ are identical. (Note that a decimation process performed on a sequence arranged in a circular form is well defined even without a reference to a starting point since the process is cyclic by definition.)

(c) We treat in this paper only those sequences C_n with a single starting point having the property described in (b).

2.2. A special case of the sequences C_n.

Notation. $HW(x)$ denotes the Hamming weight of the binary representation of a nonnegative integer x.

If the binary representation of x consists of n bits, then the binary representation of $y = (2^i \cdot x) \bmod 2^n - 1$ consists of that of x shifted cyclically to the left for i places. It then follows that $HW(x) = HW(y)$. This means that if a and b belong to the same cyclotomic coset modulo $2^n - 1$, then $HW(a) = HW(b)$. This trivial property will be used extensively later.

Notation. B_n denotes any sequence of 2^n-1 elements where $(B_n)_i=(B_n)_j$ for $HW(i)=HW(j)$. ($(B_n)_i$ denotes the ith element of B_n where the first one is no. 0.)

DEMONSTRATION 4. The structure of B_5 is

$$a\ b\ b\ c\ b\ c\ c\ d\ b\ c\ c\ d\ c\ d\ d\ e\ b\ c\ c\ d\ c\ d\ d\ e\ c\ d\ d\ e\ d\ e\ e.$$

Since $HW(i)=HW(j)$ for i and j belonging to the same cyclotomic coset modulo 2^n-1, it follows that B_n has equal elements in locations belonging to the same cyclotomic coset modulo 2^n-1. We then have the following theorem.

THEOREM 2. *The sequences B_n are a subclass of the sequences C_n.*

A connection between $HW(x)$ and the parity of x is shown next. The final aim will be to show how to recover the location of a specified element in B_n.

THEOREM 3. *If $HW(x)\neq HW(x+1)$ and $HW(x+2)\neq HW(x+3)$ for a certain x, then x is even.*

Proof. It is simply shown that $HW(x)=HW(x+1)$ for $x \bmod 4=1$. Since $HW(x)\neq HW(x+1)$ and $HW(x+2)\neq HW(x+3)$ it follows that $x \bmod 4\neq 1$ and $(x+2) \bmod 4\neq 1$. Since one integer out of x, $x+1$, $x+2$, $x+3$ modulo 4 equals 1, it must be either $x+1$ or $x+3$ and in any case, x is even.

Based on Theorem 3 we have:

THEOREM 4. *If a, b, c, d are four successive elements of a sequence B_n where $a\neq b$ and $c\neq d$ then the location index of a (in the sequence B_n) is even.*

THEOREM 5. *Let B_n^* be a sequence B_n having the property $(B_n)_{2i-1}\neq(B_n)_{2i+1_{-1}}$, $i=0, 1, \cdots, n-2$. Given at most 4 successive values in B_n^*, starting with index x, the parity of x is recovered.*

Proof. B_n^* has the property that its values in locations i and j are different for $|HW(i)-HW(j)|=1$. Based on the fact that $HW(x)=HW(x+1)-1=HW(x+2)-1=HW(x+3)-2$ for $x \bmod 4=0$, it follows that it is impossible to have more than two successive identical values in B_n^*. Having two successive identical values, the first out of the two has an odd location index. Given any 4 successive elements from B_n^*, then at least one out the following two possibilities occurs. (a) The 4 values satisfy the condition specified in Theorem 4. (b) Two successive elements have identical values. In both cases the parity of location indices of the elements are recovered.

The above result includes, of course, the special case where B_n^* is binary. In a binary B_n^*, the i and j elements have the same value iff $HW(i)$ and $HW(j)$ have the same parity. There are two such sequences, one being the complement of the other. (One of these sequences is actually the last column of a Hadamard matrix of order $2^n\times 2^n$, with its last element dropped.)

DEMONSTRATION 5. One of the binary B_5^* is

$$0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0.$$

The case treated from now onwards is the one where B_n^* is scanned cyclically, i.e., it is considered to be arranged on the circumference of a circle, where the last element in the original order is considered to be followed by the first element. Each element is however indexed in the original linear order. When trying to apply Theorem 5 under such an arrangement of B_n^*, the theorem still holds of course, except maybe for the case where the 4 elements contain the original last one followed by the first one. Taking, for example, the binary B_n^* listed in Demonstration 5, it is observed that under a cyclic arrangement the last two 0's are followed by a 0. The argument that two successive identical values indicate odd-even location indices is invalid here. However, this is the only possible place in the cyclic arrangement where 3 successive identical elements can appear, and the location of these elements is then instantly recovered. The only

troublesome case is the one where 4 successive elements in the cyclically arranged sequence, (selected for recovering the parity of some x, according to Theorem 5) start with the last original element. There is a possibility that the first two elements out of the four will be identical, in which case the parity of the first element out of the four (which was the last element in the original sequence with index 2^n-2) will be recovered wrongly as being odd. Even if the first two elements out of the described four are not equal, its last two are equal for sure. (They have indices 1 and 2 in the original sequence and belong to the same cyclotomic coset.)

CONCLUSION 1. Theorem 5 applies to a cyclically arranged B_n^* except for the case where $x=2^n-2$, in which case its parity is recovered as being odd.

The following theorem is based directly on Theorems 2, 1, 5 and Conclusion 1.

THEOREM 6. *The first* 4 *elements obtained by* 2^i *decimation of a sequence* B_n^*, *starting with* X, *yield the parity of* $(X/2^i) \bmod (2^n-1)$.

This applies to all $X \in [0, 2^n-2]$ except for the case where $(X/2^i) \bmod (n^n-1) = 2^n-2$. (In which case the parity of $X/2^i$ is recovered as being odd.)

The binary representation of $(X/2^i) \bmod 2^n-1$ is obtained by shifting cyclically the binary representation of X, i places to the right. In other words, the coefficient of 2^0 (i.e. the parity bit) in the binary representation of $(X/2^i) \bmod 2^n-1$, is the coefficient of 2^i in the binary representation of X. This observation and Theorem 6 lead to the following major result.

Result. The first 4 elements in the 2^i-decimated sequence B_n^*, starting with location X, yield the coefficient of 2^i in the binary representation of X, $i=0, 1m \cdots, n-1$. This enables recovering X by n different decimations of B_n^*, starting with element No. X. If the recovered binary representation of X is "all 1", this indicates that $X \in \{2^n-1-2^i | i=0, 1, \cdots, n-1\}$, in which case X is recovered by recovering $X+1$ or $X-1$.

The result stated above offers a direct solution to the first two problems out of the three posed in § 1. It presents a method for recovering independently the bits in the binary representation of X. Each bit is recovered from the values assigned to 4 elements whose location is well defined with respect to X. This recovery can be achieved by assigning only two possible values (the case of a binary B_n^*).

REFERENCE

[1] S. W. GOLOMB, *Shift Register Sequences*, Holden-Day, San Francisco, 1967.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986

© 1986 Society for Industrial and Applied Mathematics
016

RELATIVE INFORMATION CAPACITY OF SIMPLE RELATIONAL DATABASE SCHEMATA*

RICHARD HULL†

Abstract. Fundamental notions of relative information capacity between database structures are studied in the context of the relational model. Four progressively less restrictive formal definitions of "dominance" between pairs of relational database schemata are given. Each of these is shown to capture intuitively appealing, semantically meaningful properties which are natural for measures of relative information capacity between schemata. Relational schemata, both with and without key dependencies, are studied using these notions. A significant intuitive conclusion concerns the informal notion of relative information capacity often suggested in the conceptual database literature, which is based on accessibility of data via queries. Results here indicate that this notion is too general to accurately measure whether an underlying semantic connection exists between database schemata. Another important result of the paper shows that under any natural notion of information capacity equivalence, two relational schemata (with no dependencies) are equivalent if and only if they are identical (up to re-ordering of the attributes and relations). The approach and definitions used here can form part of the foundation for a rigorous investigation of a variety of important database problems involving data relativism, including those of schema integration and schema translation.

Key words. relational database, relative information capacity, calculous dominance, generic dominance, internal dominance, absolute dominance

AMS(MOS) subject classifications. 68, 94

1. Introduction. A central issue in the area of databases is that of data "relativism", that is, the general activity of structuring the same data in different ways. Considerable effort has been directed at understanding data relativism as it arises in the areas of user view construction [10], view integration [19], [21], [31], [32], [40], "derived" data [17], [24], [33], schema "simplification" [8], [9], [24], translation between data models [7], [11], [12], [22], [25], [26], [27], and relational database normalization theory [4], [6], [13], [28], [41]. A predominant theme in much of this work has been to build new schemata from existing ones using various structural manipulations [8], [19], [24], [31], [32], [40]. The new schemata are intended to have equivalent information capacity with the original schema, or to "subsume" the information capacity of the original schemata in some sense. In these investigations there is typically no formal definition of the notions of equivalent or dominant information capacity. The intuitively appealing approach usually taken is to say the one schema is dominated by another if any query directed at the first can be translated into an equivalent query of the second [7], [8], [13], [17], [24], [31], [32], [40]. (In addition, it is often assumed implicitly that data structured according to the first schema can be transformed into the second schema by some "nice" mapping, for instance, a fixed query which maps instances of the first schema into instances of the second.) We informally call this[1] "query-dominance". The objective of the current paper is to introduce and use simple but rigorous theoretical tools for studying this and related measures of relative informa-

* Received by the editors October 1, 1984, and in revised form August 15, 1985. This work was supported in part by the National Science Foundation grants IST-81-07480 and IST-83-06517. An extended abstract of this paper appeared in Proc. Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, April, 1984, pp. 97-109. Copyright 1984, Association for Computing Machinery, Inc.

† Department of Computer Science, University of Southern California, Los Angeles, California 90089-0782.

[1] This notion has its roots in the notion of *θ-equivalence* introduced by Codd [13], and a variant of query-dominance was used to formally investigate horizontal and vertical decomposition in [2].

tion capacity in the very simple context of schemata from the relational database model.[2]

The present investigation makes two fundamental contributions. The first consists in several theoretical results which yield conceptual insights into the area of relative information capacity. For example, one result (Theorem 5.2) indicates that the notion of query-dominance described above does not correspond to a natural, semantically meaningful type of information capacity dominance. In particular, it appears that the notion of query-dominance is too broad to accurately measure whether an underlying semantic connection exists between database schemata (Example 5.3). A second result (Corollary 6.3) shows that with virtually any reasonable measure of relative information capacity, two relational schemata without dependencies are equivalent if and only if they are identical (up to re-ordering of the attributes and relations). This substantiates the intuition that the relational model in the absence of dependencies does not provide enough data structuring mechanisms to represent a given data set in more than one way.

The second fundamental contribution of the paper is to develop a solid mathematical foundation upon which to base an extensive theoretical investigation of relative information capacity between database schemata. This foundation finds its roots in some early work on query-dominance (usually in connection with relational database normalization [2], [3], [13]), and in the more recent, abstract work of [20]. In the present work, notions from these earlier works (along with one new one) are presented in a simple, rigorous manner and shown to correspond to intuitive and significant properties of natural measures of relative information capacity. The approach here provides mechanisms for studying relative information capacity using a variety of different mathematical techniques, including mathematical logic, combinatorics, and finite permutation group theory. Although the scope of the current paper is somewhat limited, it is clear that the definitions for measuring relative information capacity presented here can be generalized to other contexts within the relational model, and also to other database models. Thus, the approach here can serve as part of the foundation for theoretical investigations of many aspects of data relativism.

In the paper, four progressively less restrictive formal measures of relative information capacity are defined.[3] Suppose that **P** and **Q** are two relational database schemata. Speaking informally, we say that **Q** *dominates* **P** if there are functions σ and τ such that (i) σ maps the family of instances of **P** into the family of instances of **Q**, (ii) τ maps the family of instances of **Q** into the family of instances of **P**, and (iii) the composition of σ followed by τ is the identity on the family of instances of **P**. Three of the measures of information capacity are based directly on this fundamental notion, and are obtained by making certain restrictions on the maps σ and τ. The first, called[4] *calculous dominance*, is the measure which arises if σ and τ are required to be

[2] The investigation here is fundamentally different than investigations such as [5], [30] and [15] into the equivalence of relational database schemes. The basic concern in [5], [30] is the equivalence of two views of an underlying universal relation, where the views are defined simply by projections. In [15] the focus is the equivalence and dominance between relational views constructed from a given underlying relational scheme using projection and join. In the current paper we do not restrict ourselves to views of a fixed underlying schema, nor to schema manipulation via projection and join alone.

[3] The first of these, calculous dominance, has its roots in the notion of "query equivalence" as described in [13], and a variant of calculous dominance has been formally studied in [2], [3]. Another two of these measures, namely absolute and generic dominance, were originally introduced in the more general Format Model [20]. Generic dominance was also studied in [34].

[4] We choose to call this type of dominance "calculous" rather than "algebraic", because it appears that a definition based on the first order predicate calculus is easier to generalize to other data models than one based on the relationally-based algebraic operators.

(essentially) expressions of the relational calculus. (It is known [3] that this notion is equivalent to query-dominance, although easier to work with.) The second notion, called *generic dominance*, is less restrictive than calculous dominance and captures the notion that "natural" database transformations treat domain elements as "essentially uninterpreted objects" [1], [20]. (This is accomplished by requiring that σ and τ commute with essentially all permutations of the underlying set of basic domain elements.) The third measure, *internal dominance* (which is even less restrictive), captures the intuitive notion that (at a logical or conceptual level) "natural" database transformations are not based on numeric computations or string manipulations.[5] This is accomplished by requiring that σ and τ do not "invent" or "construct" new domain elements (or data values) from the set of domain elements already occurring in an instance (aside from a finite set of data values, which corresponds to the set of constants occurring in a relational expression). The fourth measure, *absolute dominance*, is based on a family of cardinality conditions implied by internal dominance, and is relatively easy to work with.

This report is organized as follows. In § 2 the slightly modified version of the relational model used for this investigation is described. (The modification allows us to easily express the fact that some attributes of a relation share the same set of possible domain values, while other attributes have fundamentally distinct sets of possible domain values.) In § 3 the four measures of relative information capacity are formally defined and motivated. Section 4 presents some basic results concerning the four measures. In particular, several results are obtained which demonstrate that one schema is not query-dominated by another schema. In § 5, results indicating that query-dominance does not accurately measure the presence of semantic correspondence between schemata are given, and the result concerning equivalent schemata is given in § 6. Concluding remarks are made in § 7. (Finally, some of the more technical proofs are presented in three Appendices.)

2. Relation specifiers and schemata. The purpose of this short section is to introduce and motivate the slightly modified version of the relational model that will be used in this investigation. It is assumed here that the reader is familiar with the fundamental concepts of the relational model [29], [39]. Since the focus here is different than that of most investigations of this model, the reader is warned that we shall use some symbols here in a manner different than found elsewhere.

Speaking informally, a fundamental premise of our investigation is that the structure of relations is determined primarily by two things: the number of "columns" that a given relation has, and the sets of possible values which can appear in each of these columns. For example, if a column is intended to contain salary data, then we would expect that only positive integers are permitted as entries in that column, whereas in a column for person-names we would expect only names (or perhaps, alphabetic strings).[6] To formally capture these ideas we establish a set of "basic types" (or domain designators), along with a fixed domain of possible values associated with each basic type.

---

[5] This notion highlights the fact that the current investigation is concerned with "pure" database access and transformation languages, i.e., those which focus primarily on the data structures provided by the database model.

[6] R. Fagin studied this notion using "domain dependencies" [16]. Also, R. Reiter has studied this using the logic-based formalism of "typed" databases [36], [37].

Notation. Let $\mathfrak{B}$ be a fixed countable set of *basic types* with a fixed, unspecified total ordering. Let the function Dom be defined on $\mathfrak{B}$ such that

a. Dom (B), the *domain* of B, is a countably infinite set of abstract symbols for each $B \in \mathfrak{B}$; and

b. Dom $(B) \cap$ Dom $(C) = \emptyset$ whenever $B \neq C$.

The set **DOM** of all *domain elements* is the set $\bigcup_{B \in \mathfrak{B}}$ Dom (B).

In many cases, two or more columns of a relation may have exactly the same domain (e.g., START-DATE and END-DATE). For this reason, relations are specified using (finite) sets of basic types, where each basic type may occur more than once. To formalize this, we first review the notion of multiset.

DEFINITION. A *multiset over* a set X is a total function $M: X \to \mathbf{N}$ (the natural numbers). A multiset M is *finite* if $\{x \in X | M(x) > 0\}$ is finite. If $x \in X$ then x is an *element* of M, denoted $x \in M$, if $M(x) > 0$. The *cardinality* of a multiset M is $|M| = \sum_{x \in X} M(x)$. Finally, if L and M are multisets then their *union*, $L \cup M$, is the multiset K such that $K(x) = M(x) + L(x)$ for each $x \in X$.

We now have:

DEFINITION. A *nonkeyed (relation) specifier* is a finite multiset over $\mathfrak{B}$. The *support* of a nonkeyed specifier R is the set supp $(R) = \{B \in \mathfrak{B} | R(B) > 0\}$.

While not all possible real-world relations can be modeled within the framework that is being developed here, the results obtained in this limited context are of sufficient interest to warrant investigation; furthermore, in this new area it is important to resolve simple problems before tackling the more complicated ones.

We generally denote a nonkeyed specifier by listing its elements, with multiple occurrences where appropriate. For instance, if R is a specifier with support $\{A, B, C\}$ and $R(A) = 2$, $R(B) = 1$, and $R(C) = 3$, we denote R by $AABCCC$ or A^2BC^3.

Formally, (relational) instances are associated with nonkeyed specifiers as follows.

DEFINITION. If R is a nonkeyed specifier, an *instance* of R is a finite subset of[7] $\times_{B \in \text{supp}(R)} (\times_{i=1}^{R(B)} (\text{Dom}(B)))$. The family of instances of R is denoted $\mathbf{I}(R)$.

We now extend our notation to include one key dependency per relation. Key dependencies, especially in the case of one key dependency per relation, are fundamental to many semantic data models, including the functional data model [23], [38] and the entity-relationship model [11]. Key dependencies are incorporated into our notation in the following convenient manner.

DEFINITION. A *keyed (relation) specifier* is an ordered pair (R, S) of multisets over $\mathfrak{B}$, usually written as $R:S$. The *support* of $R:S$, supp $(R:S)$, is[8] supp (RS). An *instance* of $R:S$ is a total function from an instance I of R to $\times_{B \in \text{supp}(S)} (\times_{i=1}^{S(B)} (\text{Dom}(B)))$. (We typically view such instances as if they are members of $\mathbf{I}(RS)$, i.e., as finite sets of ordered tuples.) The family of instances of a keyed specifier $R:S$ is denoted $\mathbf{I}(R:S)$.

Note that a nonkeyed specifier R can be viewed as the keyed specifier $R:\emptyset$.

Speaking informally, a relation scheme consists of one or more relations, some of which may share the same underlying (column) structure. For this reason, we formally define a relation schema to be a multiset of relation specifiers.

---

[7] We view Cartesian products such as this one as "flattened", that is, we view this product as a single product of subsets of **DOM** rather than as a product of products of subsets of **DOM**. Furthermore, we view the columns of this product to be ordered by the context of the discussion. If no order is specified by that context, then the underlying ordering on $\mathfrak{B}$ is used.

[8] Following relational tradition, if R and S are nonkeyed specifiers, we denote their union $R \cup S$ by RS.

DEFINITION.[9] A (*relational*) *schema* is a finite multiset[10] $\mathbf{P} = P_1, P_2, \cdots, P_n$ of relation specifiers. The *support* of $\mathbf{P}$ is the set $\text{supp}(\mathbf{P}) = \bigcup_{j=1}^{n} \text{supp}(P_j)$. An *instance* of $\mathbf{P}$ is an element of[11] $\mathbf{I}(\mathbf{P}) = \times_{j=1}^{n} \mathbf{I}(P_j)$. If P_j is nonkeyed for $1 \leqq j \leqq n$, then $\mathbf{P}$ is a *nonkeyed* relational schema.

As an aside, we note that the family of nonkeyed relational schemata as defined here corresponds precisely, in the terminology of the Format Model [20], to the family of formats which are constructed using a composition of one or more subformats, each of which is a collection of a composition of one or more basic types (except that here we do not associate "tokens" with the various components of our relational schemata).

Finally, we mention the version of the relational calculus used here. We assume that the reader is familiar with the calculus as described in [14], [29], [39]). In the current investigation we use, in the terminology of [39], the *domain relational calculus* in the sense that the variables and constants in our calculus range over individual domain elements. (Results in [39] and elsewhere indicate that the calculi obtained by letting variables range over tuples or domain elements are equivalent.) In keeping with our definition of basic types and the fact that their associated domains are disjoint, we assume that each (domain-value) variable occurring in our calculus expressions is associated with a given basic type. (Formally, we assume that for each basic type B there is an infinite set V_B of *B-variables*, where $V_B \cap V_C = \varnothing$ whenever $B \neq C$.) As noted earlier, we assume here for each relation specifier R (or keyed specifier $R:S$) that a fixed ordering of the occurrences of the basic types in R (RS) is given by the context of the discussion (or that an unspecified default ordering is used), and similarly that for each relational schema $\mathbf{P}$ a fixed ordering of the occurrences of the specifiers in $\mathbf{P}$ is given. These fixed orderings are used in calculus expressions to specify the specific "relation" (i.e., specifier occurrence in a schema) and the specific "column" (i.e., basic type occurrence in the specifier) that a given variable refers to. In the terminology of [39], only *safe* calculus expressions will be used. (Intuitively, the definition of safe expressions prevents them from yielding an infinite relation when evaluated on an n-tuple of finite relations.) Also, since no ordering is associated with the domains of basic types, we do not permit the predicates $<$ or $>$ (or more correctly, the symbols for these predicates) to occur in our calculus expressions. Finally, given relational schemata $\mathbf{P} = P_1, \cdots, P_m$ and $\mathbf{Q} = Q_1, \cdots, Q_n$, a (relational) *calculus expression* from $\mathbf{P}$ to $\mathbf{Q}$ is an n-tuple $\xi = (\xi_1, \cdots, \xi_n)$ where ξ_j is a (conventional) calculus expression which takes as input an instance of $\mathbf{P}$ and yields as output an instance of Q_j $(1 \leqq j \leqq n)$. In this case we write $\xi: \mathbf{P} \to \mathbf{Q}$.

3. Four measures of relative information capacity. In this section the four measures of relative information capacity are introduced and motivated. As noted in the Introduction, the first of these, calculous dominance, has its roots in the notion of "query equivalence" as described in [13], and a variant of it has been formally studied in [2], [3]. As will be seen, this notion is equivalent to the notion of query-dominance described in the Introduction, but is easier to work with. Two other measures of relative information capacity, namely absolute and generic dominance, were originally

[9] We use the term "schema" here to distinquish it from the usual notion of a relational "scheme", where a set of attributes as opposed to a multiset of basic types is specified for each relation [29], [39].

[10] When listing the occurrences of elements in a schema we separate them by commas to avoid ambiguities. (For example, A^2, B^2 denotes a schema with two specifiers, while A^2B^2 denotes a single specifier (or a schema with one specifier in it).)

[11] Elements of $\mathbf{I}(\mathbf{P})$ are n-tuples (thus, we do not view this product as "flattened"); and as before the order of the coordinates in this product are given by the context of the discussion, or if no such order is determined, by some unspecified but fixed ordering.

introduced in the more general Format Model [20], and generic dominance was also studied in [34]. Finally, the notion of internal dominance is a new notion which is based solely on the intuition that natural database transformations do not "invent" or "construct" new domain elements from old ones. The section concludes with a result stating that the four measures of information capacity are progressively less restrictive.

To begin the formal discussion, we present a notion fundamental to our approach.

DEFINITION. Let $\mathbf{P}$ and $\mathbf{Q}$ be relational schemata. A (*schema*) *transformation from* $\mathbf{P}$ *to* $\mathbf{Q}$ is a map $\sigma: \mathbf{I}(\mathbf{P}) \to \mathbf{I}(\mathbf{Q})$. In this case we write $\sigma: \mathbf{P} \to \mathbf{Q}$.

Note that a calculus expression $\xi: \mathbf{P} \to \mathbf{Q}$ can be viewed as a transformation.

In the spirit of [2], [3], [13], [20], we define relative information capacity using a pair of transformations, the composition of which forms the identity on the dominated family of instances:

DEFINITION. Let $\mathbf{P}$ and $\mathbf{Q}$ be schemata, and let $\sigma: \mathbf{P} \to \mathbf{Q}$ and $\tau: \mathbf{Q} \to \mathbf{P}$. Then $\mathbf{Q}$ *dominates* $\mathbf{P}$ *via* (σ, τ), denoted $\mathbf{P} \leqq \mathbf{Q}$ via (σ, τ), if $\tau \circ \sigma$ (i.e., the composition of σ followed by τ) is the identity on $\mathbf{I}(\mathbf{P})$.

Suppose that $\mathbf{P} \leqq \mathbf{Q}$ via (σ, τ). This means that information structured according to $\mathbf{P}$ can be restructured (via σ) to "fit" into $\mathbf{Q}$, and restructured again (via τ) to "fit" into $\mathbf{P}$, in such a way that the result is the same as the original. This suggests that $\mathbf{Q}$ has at least as much capacity for storing information as does $\mathbf{P}$.

It should be noted that $\mathbf{Q}$ dominates $\mathbf{P}$ via some pair σ, τ iff there is an injection of $\mathbf{I}(\mathbf{P})$ into $\mathbf{I}(\mathbf{Q})$. Our definition is given in terms of both σ and τ, because we shall restrict both σ and τ to have certain properties (e.g., that both be calculous mappings).

The first of our measures is based on restricting the class of permissible database transformations to be calculus expressions. We note that the notion in [2] of one schema being *included* in a second one is the same as our notion here of calculous dominance, except that in their formal investigation the query language used includes only the operations of projection, selection, join and union, which does not have the full power of the relational calculus. (For example, set difference cannot be realized using these operators).

DEFINITION. Let $\mathbf{P}$ and $\mathbf{Q}$ be schemata. Then $\mathbf{Q}$ *dominates* $\mathbf{P}$ *calculously*, denoted $\mathbf{P} \leqq \mathbf{Q}(\text{calc})$, if there is a pair of calculus expressions $\xi: \mathbf{P} \to \mathbf{Q}$ and $\omega: \mathbf{Q} \to \mathbf{P}$ such that $\mathbf{P} \leqq \mathbf{Q}$ via (ξ, ω). $\mathbf{P}$ and $\mathbf{Q}$ are *calculously equivalent*, denoted $\mathbf{P} \sim \mathbf{Q}(\text{calc})$, if $\mathbf{P} \leqq \mathbf{Q}(\text{calc})$ and $\mathbf{Q} \leqq \mathbf{P}(\text{calc})$.

It is easily verified that calculous dominance is transitive and reflexive, and that calculous equivalence is an equivalence relation.

To compare the notion of calculous dominance as just defined with the notion of query-dominance described in the Introduction, we present a formal definition of query-dominance for the current setting.

DEFINITION. Let $\mathbf{P}$ and $\mathbf{Q}$ be schemata. Then $\mathbf{Q}$ *query-dominates* $\mathbf{P}$ if there is a calculus expression $\mu: \mathbf{P} \to \mathbf{Q}$ such that for each relation specifier R and each calculus expression $\alpha: \mathbf{P} \to R$, there is a calculus expression $\beta: \mathbf{Q} \to R$ such that $\alpha = \beta \circ \mu$.

The next result formally states the equivalence of calculous dominance and query-dominance. (The proof is omitted because the techniques of [3], [13] are easily modified to fit the current context.)

PROPOSITION 3.1 [3], [13]. *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be the schemata. Then* $\mathbf{Q}$ *query-dominates* $\mathbf{P}$ *iff* $\mathbf{P} \leqq \mathbf{Q}(\text{calc})$. □

There are two advantages to the definition here of calculous dominance over the definition of query-dominance. First, calculous dominance is easier to work with because it involves only two calculus expressions (as opposed to infinitely many).

Second, as will be seen shortly, the form of the definition of calculous dominance is easily generalized to provide a variety of techniques for studying it.

The second measure of relative information capacity, called "generic dominance," is somewhat more general than calculous dominance, and is useful in showing that one schema does not calculously dominate another (see Proposition 4.5 below). Generic dominance formally captures an intuitively natural restriction on database transformations, namely that any transformation used should "... treat data values as essentially uninterpreted objects ..." [1], [20]. (However, we do allow our transformations to use a bounded number of domain elements as "constants", which correspond intuitively to the constants occurring in calculus expressions.) As in [20], to define generic dominance we first formalize this property of genericity.

DEFINITION. Let $Z \subseteq \mathbf{DOM}$. A *Z-permutation* (*of* **DOM**) is a function $\pi: \mathbf{DOM} \to \mathbf{DOM}$ such that

(a) $\pi(z) = z$ for each $z \in Z$, and

(b) the restriction of π to $\mathrm{Dom}(B)$ is a 1:1 onto function from $\mathrm{Dom}(B)$ to $\mathrm{Dom}(B)$ for each $B \in \mathscr{B}$.

Permutations on **DOM** are extended to families of instances in the natural manner. A transformation $\sigma: \mathbf{P} \to \mathbf{Q}$ is *Z-generic* if for each Z-permutation π and each instance I of $\mathbf{P}$, $\pi \circ \sigma(I) = \sigma \circ \pi(I)$ (i.e., σ and π commute on $\mathbf{I}(\mathbf{P})$).

Speaking informally, a Z-permutation π leaves Z fixed, and the restriction of π to $\mathrm{Dom}(B)$ is a permutation of $\mathrm{Dom}(B)$ for each basic type B. And, again speaking informally, a transformation is Z-generic if for each $B \in \mathscr{B}$ it treats all elements of $\mathrm{Dom}(B) - Z$ as "equals".

Using an induction on subformulas, the following result is easily verified (proof omitted).

LEMMA 3.2. *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be schemata,* $\xi: \mathbf{P} \to \mathbf{Q}$ *be a calculus expression, and let* Z *be the set of constants occurring in* ξ. *Then* ξ *is a* Z*-generic transformation.* □

It is easily seen that an analogous result holds for any query language that preserves Z-permutations. For example, the result applies to the query language of [1], which includes a least fixed point operator.

Following [20], generic dominance is now defined by requiring that the transformations σ and τ which restructure data be generic. (While technically different than the original definition of generic dominance given in [20], it can be verified that the notion used here and the original notion are equivalent in the current context.)

DEFINITION. Let $\mathbf{P}$ and $\mathbf{Q}$ be schemata. Then $\mathbf{Q}$ *dominates* $\mathbf{P}$ *generically*, denoted $\mathbf{P} \leqq \mathbf{Q}(\mathrm{gen})$, if there is a finite $Z \subseteq \mathbf{DOM}$ and Z-generic transformations $\sigma: \mathbf{P} \to \mathbf{Q}$ and $\tau: \mathbf{Q} \to \mathbf{P}$ such that $\mathbf{P} \leqq \mathbf{Q}$ via (σ, τ). $\mathbf{P}$ and $\mathbf{Q}$ are *generically equivalent*, denoted $\mathbf{P} \sim \mathbf{Q}(\mathrm{gen})$, if $\mathbf{P} \leqq \mathbf{Q}(\mathrm{gen})$ and $\mathbf{Q} \leqq \mathbf{P}(\mathrm{gen})$.

As with calculous dominance and equivalence, it is easily verified that generic dominance is reflexive and transitive, and that generic equivalence is an equivalence relation.

Generic dominance is of particular importance because it is independent of any data-access language, but captures a property of all such languages discussed in the literature.[12] (For example, each query in the language consisting of the relational algebra plus the least-fixed point operator is generic although this language is strictly stronger than the relational algebra or calculus [1].) Thus, results stating that one schema is not generically dominated by another can be used to support an intuitive

[12] As noted earlier, this investigation is concerned only with "pure" database query or transformation languages which do not encompass numeric computation or string manipulation.

claim that the first schema is not query-dominated by the other, where any natural query language is being used.

Our third measure of relative information capacity is more general than generic dominance, and focuses on the natural property that database transformations (at least, those used for restructuring data sets) are not typically based on numerical computations or string manipulations, and thus do not typically "invent" data values. (For example, a mapping which encodes the pair (i, j) of integers into the single integer 2^i3^j is based on a computation, and in essence "invents" the value 2^i3^j.)

To formally capture this property of not inventing data elements, we first need:

DEFINITION. Let I be an instance of a relation schema. Then the set of *symbols* of I, denoted $\mathrm{Sym}(I)$, is the set of elements of **DOM** which occur in I.

In the following we allow each given transformation to "invent" a (finite) set of data elements, these corresponding intuitively to the set of constants that might occur in a calculus expression. (As noted in § 7, it would also be interesting to study transformations which do not "invent" any domain elements at all.)

DEFINITION. Let $Z \subseteq \mathbf{DOM}$. A transformation $\sigma: \mathbf{P} \to \mathbf{Q}$ is *Z-internal* if $\mathrm{Sym}(\sigma(I)) \subseteq \mathrm{Sym}(I) \cup Z$ for each $I \in \mathbf{I}(\mathbf{P})$.

Note that if σ is Z-internal for some finite Z and $\mathrm{Sym}(I) \supseteq Z$, then $\mathrm{Sym}(\sigma(I)) \subseteq \mathrm{Sym}(I)$. As implied by the following result, each Z-generic transformation is Z-internal. (And by Lemma 3.2, each calculus expression is Z-internal for some finite Z.)

LEMMA 3.3. *Let* **P** *and* **Q** *be schemata,* $Z \subseteq \mathbf{DOM}$, *and* $\sigma: \mathbf{P} \to \mathbf{Q}$ *be Z-generic. Then* σ *is Z-internal.*

Proof. Suppose that $\sigma: \mathbf{P} \to \mathbf{Q}$ is Z-generic, and let $I \in \mathbf{I}(\mathbf{P})$. Suppose further that $\mathrm{Sym}(\sigma(I)) \not\subseteq Z \cup \mathrm{Sym}(I)$. Let B be a basic type such that there is some $b \in \mathrm{Dom}(B)$ with $b \in \mathrm{Sym}(\sigma(I)) - (Z \cup \mathrm{Sym}(I))$, and let $c \in \mathrm{Dom}(B) - (Z \cup \mathrm{Sym}(I) \cup \mathrm{Sym}(\sigma(I)))$. (Such a c exists because $\mathrm{Dom}(B)$ is infinite while Z, $\mathrm{Sym}(I)$ and $\mathrm{Sym}(\sigma(I))$ are all finite.) Let π be the Z-permutation such that $\pi(b) = c$, $\pi(c) = b$, and π is the identity on all other elements of **DOM**. Then $\pi(I) = I$ (since b and c are not in $\mathrm{Sym}(I)$) but $\sigma(I) \neq \pi(\sigma(I))$ (since c occurs in $\pi(\sigma(I))$ but not in $\sigma(I)$). Therefore $\sigma(\pi(I)) = \sigma(I) \neq \pi(\sigma(I))$, contradicting the assumption that σ is Z-generic. □

We now have:

DEFINITION. Let **P** and **Q** be schemata. Then **Q** *dominates* **P** *internally*, denoted $\mathbf{P} \leqq \mathbf{Q}(\mathrm{int})$, if there is a finite $Z \subseteq \mathbf{DOM}$ and Z-internal transformations $\sigma: \mathbf{P} \to \mathbf{Q}$ and $\tau: \mathbf{Q} \to \mathbf{P}$ such that $\mathbf{P} \leqq \mathbf{Q}$ via (σ, τ). **P** and **Q** are *internally equivalent*, denoted $\mathbf{P} \sim \mathbf{Q}(\mathrm{int})$, if $\mathbf{P} \leqq \mathbf{Q}(\mathrm{int})$ and $\mathbf{Q} \leqq \mathbf{P}(\mathrm{int})$.

As before, internal dominance is reflexive and transitive, and internal equivalence is an equivalence relation.

Our final measure of relative information capacity does not have the form of the other three, and is more general than all of them. The primary advantage of this final measure is that it is easily characterized in terms of the cardinalities of certain families of instances (see Theorem 4.2), and is therefore relatively easy to work with.

To define this type of dominance we need the following.

Notation. Let **P** be a relation schema and $Y \subseteq \mathbf{DOM}$. Then $\mathbf{I}_Y(\mathbf{P}) = \{I \in \mathbf{I}(\mathbf{P}) \mid \mathrm{Sym}(I) \subseteq Y\}$.

Suppose now that $\sigma: \mathbf{P} \to \mathbf{Q}$ is Z-internal and $Y \supseteq Z$. Then for each $I \in \mathbf{I}_Y(\mathbf{P})$, $\sigma(I) \in \mathbf{I}_Y(\mathbf{Q})$. In other words,[13] $\sigma[\mathbf{I}_Y(\mathbf{P})] \subseteq \mathbf{I}_Y(\mathbf{Q})$. Finally, if $\mathbf{P} \leqq \mathbf{Q}$ via (σ, τ) where

[13] If $f: M \to N$ and $K \subseteq M$, then $f[K]$ denotes $\{f(k) \mid k \in K\}$.

σ and τ are Z-internal, then σ is 1-1 and so[14] $|\mathbf{I}_Y(\mathbf{P})| \leqq |\mathbf{I}_Y(\mathbf{Q})|$ for each $Y \supseteq Z$. This motivates:

DEFINITION. Let $\mathbf{P}$ and $\mathbf{Q}$ be schemata. Then $\mathbf{Q}$ *dominates* $\mathbf{P}$ *absolutely*, denoted $\mathbf{P} \leqq \mathbf{Q}$(abs), if there is a finite $Z \subseteq \mathbf{DOM}$ such that $|\mathbf{I}_Y(\mathbf{P})| \leqq |\mathbf{I}_Y(\mathbf{Q})|$ for each (finite) $Y \supseteq Z$. $\mathbf{P}$ and $\mathbf{Q}$ are *absolutely equivalent*, denoted $\mathbf{P} \sim \mathbf{Q}$(abs), if $\mathbf{P} \leqq \mathbf{Q}$(abs) and $\mathbf{Q} \leqq \mathbf{P}$(abs).

(While technically different than the original definition of absolute dominance given in [20], it is easily verified that the notion used here and the original notion are equivalent in the current context.)

We conclude the section by showing that each of the formal measures of relative information capacity introduced above are indeed progressively less restrictive, in the sense that if $\mathbf{P}$ is dominated by $\mathbf{Q}$ according to one of the measures, then $\mathbf{P}$ is dominated by $\mathbf{Q}$ according to each of the subsequent measures.

THEOREM 3.4. *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be schemata. Then* $\mathbf{P} \leqq \mathbf{Q}$(calc) *implies* $\mathbf{P} \leqq \mathbf{Q}$(gen); $\mathbf{P} \leqq \mathbf{Q}$(gen) *implies* $\mathbf{P} \leqq \mathbf{Q}$(int); *and* $\mathbf{P} \leqq \mathbf{Q}$(int) *implies* $\mathbf{P} \leqq \mathbf{Q}$(abs).

Proof. Lemma 3.2 yields the first implication, and Lemma 3.3 yields the second one. Finally, suppose that $\mathbf{P} \leqq \mathbf{Q}$(int). Then there is a finite $Z \subseteq \mathbf{DOM}$ and Z-internal transformations $\sigma: \mathbf{P} \to \mathbf{Q}$ and $\tau: \mathbf{Q} \to \mathbf{P}$ such that $\tau \circ \sigma$ is the identity on $\mathbf{I}(\mathbf{P})$. In particular, then, σ is 1-1 on $\mathbf{I}(\mathbf{P})$. If Y is finite with $Z \subseteq Y \subseteq \mathbf{DOM}$, then $\sigma(\mathbf{I}_Y(\mathbf{P})) \subseteq \mathbf{I}_Y(\mathbf{Q})$ since σ is Z-internal, and so $|\mathbf{I}_Y(\mathbf{P})| \leqq |\mathbf{I}_Y(\mathbf{Q})|$ since σ is 1-1. Thus, $\mathbf{P} \leqq \mathbf{Q}$(abs). □

As we shall see, the converse of each of these implications is also true for nonkeyed relational schemata $\mathbf{P}$ and $\mathbf{Q}$ where $\mathbf{Q}$ consists of only one relation specifier (Theorem 5.2), and for nonkeyed relational schemata involving only one basic type (Theorem 5.6). However, if $\mathbf{Q}$ contains more that one specifier, or if key dependencies are incorporated, then at least one of these converse implications fails. More specifically, Proposition 4.5 shows that in either of these situations there are $\mathbf{P}$, $\mathbf{Q}$ such that $\mathbf{P} \leqq \mathbf{Q}$(int, abs) but $\mathbf{P} \not\leqq \mathbf{Q}$(calc, gen). It remains open whether calculous and generic dominance can be separated, or whether internal and absolute dominance can be separated.

4. Some basic results. In this section we present several basic results concerning the notions of information capacity dominance defined above. The first result gives a characterization of absolute dominance in terms of certain functions. A simple application of this result is given to indicate how it can be used to show that calculous dominance does not hold, and this result is also used as the basis for Theorem 6.2. The second major result of the section (Theorem 4.4) gives a characterization of internal dominance. The third major result (Proposition 4.5) shows that absolute and internal dominance are different from generic and calculous dominance, and illustrates a technique for showing that generic dominance does not hold. The section concludes with a number of results giving sufficient conditions for dominance (of one sort or another) to hold. Most important of these is Theorem 4.7, which concerns schemata constructed from other schemata through "re-namings" of the basic types used.

The functions needed for the characterization of absolute dominance are now presented.

DEFINITION. Let R be a nonkeyed relation specifier and let $B_1, \cdots, B_n$ be an enumeration of basic types that includes all basic types in supp(R) and possibly including other basic types as well. Then the *cardinality expression* of R (relative to this enumeration) is the expression $f_R(x_1, \cdots, x_n) = f_R(\mathbf{x})$ for indeterminates $x_1, \cdots, x_n$

[14] If X is a set then $|X|$ denotes the cardinality of X.

defined by

$$f_R(\mathbf{x}) = \prod_{1 \leqq j \leqq n} (x_j^{R(B_j)}).$$

Now suppose that R is the keyed specifier $S:T$ and $\text{supp}(R) \subseteqq \{B_1, \cdots, B_n\}$. Then the *cardinality expression* of R is

$$f_R(\mathbf{x}) = \left[\prod_{1 \leqq j \leqq n} (x_j^{S(B_j)})\right] \cdot \left[\log_2\left(\left(\prod_{1 \leqq j \leqq n} x_j^{T(B_j)}\right)+1\right)\right]$$
$$= f_S(\mathbf{x}) \log_2 (f_T(\mathbf{x})+1).$$

Finally, let $\mathbf{P} = P_1, \cdots, P_m$ be a relational schema with $\text{supp}(\mathbf{P}) \subseteqq \{B_1, \cdots, B_n\}$. Then the *cardinality expression* of $\mathbf{P}$ is

$$f_{\mathbf{P}}(\mathbf{x}) = \sum_{1 \leqq i \leqq m} (f_{P_i}(\mathbf{x})).$$

The significance of the cardinality expressions is given by:

LEMMA 4.1. *Let* $\mathbf{P}$ *be a relational schema with support contained in* $B_1, \cdots, B_n$, *and let* $X \subseteqq \mathbf{DOM}$ *be a finite set with* $|X \cap \text{Dom}(B_j)| = x_j$, $1 \leqq j \leqq n$. *Then*

$$|\mathbf{I}_X(\mathbf{P})| = 2^{f_{\mathbf{P}}(\mathbf{x})}.$$

Proof. Suppose first that $\mathbf{P}$ consists simply of one occurrence of a nonkeyed specifier R, where $|R| = m$. The number of distinct m-tuples in $\bigtimes_{B \in \text{supp}(R)} (\bigtimes_{i=1}^{R(B)} (\text{Dom}(B)))$ with coordinate values taken from X is clearly $f_R(\mathbf{x}) = \prod_{1 \leqq j \leqq n} (x_j^{R(B_j)})$. Thus, the number of distinct instances of R with values taken from X is the number of possible sets of these tuples, namely $2^{f_R(\mathbf{x})}$ as desired.

Suppose now that $R = S:T$, where T is nonempty. Recall that an instance of R is a total function from an instance I of S into $\bigtimes_{B \in \text{supp}(T)} (\bigtimes_{i=1}^{T(B)} (\text{Dom}(B)))$. In the present situation, we are concerned exclusively with such functions where the domain and range involve only elements of X. Thus, we are concerned with the number of *partial* functions from

$$S_X = \bigtimes_{B \in \text{supp}(S)} \left(\bigtimes_{i=1}^{S(B)} (X \cap \text{Dom}(B))\right)$$

into the set

$$T_X = \bigtimes_{B \in \text{supp}(T)} \left(\bigtimes_{i=1}^{T(B)} (X \cap \text{Dom}(B))\right).$$

To count the number of such functions, note that there is a 1-1 correspondence between the collection of these (partial) functions, and the collection of total functions from S_X into $T_X \cup \{\text{"NOT\_THERE"}\}$. (Intuitively speaking, if K is in this new collection of functions and if a tuple u of S_X is given the value "NOT_THERE" in K, then the tuple u does not occur in the domain of the instance of $\mathbf{I}(\mathbf{R})$ which corresponds to K.) The number of functions in the latter collection is easily seen to be

$$[|T_X|+1]^{|S_X|} = [f_T(\mathbf{x})+1]^{f_S(\mathbf{x})}.$$

The logarithm of this expression is precisely $f_R(\mathbf{x})$ as desired. (Note that if T is empty, and if we define $\prod_{1 \leqq j \leqq 0} n_j = 1$, then the logarithm term of $f_R(\mathbf{x})$ is $\log_2(1+1) = 1$, so the result also holds in this special case.)

Finally, the extension of this result to schemata involving more than one specifier is straightforward. □

The following characterization of absolute dominance is now immediate (proof omitted).

THEOREM 4.2. *Let* **P** *and* **Q** *be relational schemata. Then* $\mathbf{P} \leqq \mathbf{Q}$(abs) *iff there is some* $t \geqq 0$ *such that* $f_{\mathbf{P}}(\mathbf{x}) \leqq f_{\mathbf{Q}}(\mathbf{x})$ *for each* $\mathbf{x}$ *with* $x_j \geqq t (1 \leqq j \leqq n)$. □

The above result provides an easy mechanism for showing that calculous dominance does not hold in many cases. For example, the following corollary yields the intuitive conclusion that a relation with two NAME columns and one NUMBER column is not query-dominated by a relation with one NAME column and two NUMBER columns.

COROLLARY 4.3. $AAB \not\leqq ABB$(abs), *and hence* $AAB \not\leqq ABB$(calc).

Proof. Suppose $AAB \leqq ABB$(abs), and let $\{A, B\}$ be enumerated A, B. By Theorem 4.2 there is some $t \geqq 0$ such that for each ordered pair (x,y) with $x \geqq t$ and $y \geqq t$, $f_{\mathbf{P}}(x, y) \leqq f_{\mathbf{Q}}(x, y)$, i.e., $x^2 y \leqq xy^2$. But this is false for $x = 2s$ and $y = s$, where $s > t$ (and hence, $s > 0$). Thus $AAB \not\leqq ABB$(abs) after all. Finally, $AAB \not\leqq ABB$(calc) follows from Theorem 3.4. □

It is clear that the technique of the above proof can be applied in many situations to infer that one schema is not calculously dominated by another one.

We now turn to the second major result of the section, namely a characterization of internal dominance. In particular, this result shows that internal dominance is equivalent to a cardinality condition which is similar in spirit to the definition of absolute dominance. To state the result we need:

Notation. Let **P** be a schema, and let X and Y be finite subsets of **DOM**. Then $\mathbf{P}(X, Y)$ denotes $\{I \in \mathbf{I}(\mathbf{P}) \mid Y \subseteq \text{Sym}(I) \subseteq X\}$.

Thus, instances in $\mathbf{P}(X, Y)$ involve all the symbols of Y, and no symbols outside of X. Note that for each **P** and finite $X \subseteq \mathbf{DOM}$, $\mathbf{P}(X, \emptyset) = \mathbf{I}_X(\mathbf{P})$. Also, it is easily verified that $\mathbf{P}(X, Y) \neq \emptyset$ iff $Y \subseteq X \cap \bigcup_{B \in \text{supp}(\mathbf{P})} \text{Dom}(B)$. Finally, speaking intuitively note that if $\sigma[\mathbf{P}(X, Y)] \subseteq \mathbf{Q}(X, Y)$, then σ in some sense "preserves" Y.

We now have:

THEOREM 4.4. *Let* **P** *and* **Q** *be schemata. Then* $\mathbf{P} \leqq \mathbf{Q}$(int) *iff there is some finite* $Z \subseteq \mathbf{DOM}$ *such that*[15] $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$ *for each finite* $Y \subseteq \mathbf{DOM} - Z$.

Proof. Suppose that $\mathbf{P} \leqq \mathbf{Q}$(int). Then there is a finite Z and Z-internal transformations $\sigma: \mathbf{P} \to \mathbf{Q}$ and $\tau: \mathbf{Q} \to \mathbf{P}$ such that $\tau \circ \sigma$ is the identity on $\mathbf{I}(\mathbf{P})$. Let $Y \subseteq \mathbf{DOM} - Z$ be finite. Since σ is 1-1, to show that $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$ it suffices to show that $\sigma[\mathbf{P}(YZ, Y)] \subseteq \mathbf{Q}(YZ, Y)$. Suppose $I \in \mathbf{P}(YZ, Y)$. Then $Y \subseteq \text{Sym}(I) \subseteq YZ$. Since σ is Z-internal and $YZ \supseteq Z$, $\text{Sym}(\sigma(I)) \subseteq YZ$. Suppose that $\text{Sym}(\sigma(I)) \not\supseteq Y$. Let $b \in Y - \text{Sym}(\sigma(I))$. Since $Y \subseteq \text{Sym}(I)$, $b \in \text{Sym}(I)$. But since τ is Z-internal, $b \notin \text{Sym}(\tau(\sigma(I))) = \text{Sym}(I)$, a contradiction. Thus $\sigma(I) \in \mathbf{Q}(YZ, Y)$, and more generally $\sigma[\mathbf{P}(YZ, Y)] \subseteq \mathbf{Q}(YZ, Y)$. With this we have shown that $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$ for each finite Y such that $Y \cap Z = \emptyset$.

Suppose now that $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$ for each finite Y where $Y \cap Z = \emptyset$. Note that $\{\mathbf{P}(YZ, Y) \mid Y \subseteq \bigcup_{B \in \text{supp}(\mathbf{P})} \text{Dom}(B) - Z \text{ is finite}\}$ forms a partition of $\mathbf{I}(\mathbf{P})$ and $\{\mathbf{Q}(YZ, Y) \mid Y \subseteq \bigcup_{B \in \text{supp}(\mathbf{Q})} \text{Dom}(B) - Z \text{ is finite}\}$ forms a partition of $\mathbf{I}(\mathbf{Q})$. Furthermore, note that $\text{supp}(\mathbf{P}) \subseteq \text{supp}(\mathbf{Q})$. (For suppose that $B \in \text{supp}(\mathbf{P}) - \text{supp}(\mathbf{Q})$, and let $\emptyset \neq Y \subseteq \text{Dom}(B) - Z$ be finite. Then $\mathbf{P}(YZ, Y) \neq \emptyset$, whence $0 < |\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$. It follows that $\mathbf{Q}(YZ, Y) \neq \emptyset$, and hence that $B \in \text{supp}(\mathbf{Q})$.) Therefore, $\{\mathbf{Q}(YZ, Y) \mid Y \subseteq \bigcup_{B \in \text{supp}(\mathbf{P})} \text{Dom}(B) - Z \text{ is finite}\}$ forms a partition of a subset of $\mathbf{I}(\mathbf{Q})$.

Let some finite $Y \subseteq \bigcup_{B \in \text{supp}(\mathbf{P})} \text{Dom}(B) - Z$. By assumption, $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$, so there is a 1-1 map $\sigma_Y: P(YZ, Y) \to Q(YZ, Y)$. Letting $\sigma =$

[15] For this discussion, if M and N are subsets of **DOM** we use MN to denote $M \cup N$, etc.

$\cup\{\sigma_Y | Y \subseteq \cup_{B \in \text{supp}(\mathbf{P})} \text{Dom}(B) - Z$ is finite$\}$ it follows that σ is 1-1 on $\mathbf{I}(\mathbf{P})$. Finally, letting τ be defined so that τ is σ^{-1} on $\sigma[\mathbf{I}(\mathbf{P})]$ and $\tau(J) = \emptyset$ for each J in $\mathbf{I}(\mathbf{Q}) - \sigma[\mathbf{I}(\mathbf{P})]$, it is easily verified that τ is Z-internal and that $\mathbf{P} \leqq \mathbf{Q}$ via (σ, τ). Thus $\mathbf{P} \leqq \mathbf{Q}$(int) as desired. □

We now turn to the third major result of the section, which shows that there are examples where absolute and internal dominance hold, but generic and hence calculous dominance do not. The general proof technique used to show that generic dominance does not hold is of interest, because it provides one of the few known methods for demonstrating that one schema is not calculously dominated by another one, even though absolute dominance holds.

PROPOSITION 4.5.

a. $AB \leqq AA, BB$(abs, int) but[16] $AB \not\leqq [AA]^n, [BB]^n$(gen, calc) *for each* $n > 0$; *and*

b. $A : AA \leqq AA$(abs, int) *but* $A : AA \not\leqq AA$(gen, calc).

Proof. We first show that internal dominance (and hence absolute dominance) holds in both cases. To show that $AB \leqq AA, BB$(int) we use Theorem 4.4 with $Z = \emptyset$. Let $\mathbf{P} = AB$ and $\mathbf{Q} = AA, BB$. Suppose $Y \subseteq \mathbf{DOM}$ is finite, and set $Y_A = Y \cap \text{Dom}(A)$ and $Y_B = Y \cap \text{Dom}(B)$. If $|Y_A| \geqq |Y_B|$ let $f : Y_B \to Y_A$ be a 1-1 function, and define $\sigma_Y : \mathbf{P}(Y, Y) \to \mathbf{Q}(Y, Y)$ so that an instance $I \in \mathbf{P}(Y, Y)$ is mapped into $(J, K) \in \mathbf{Q}(Y, Y)$ where $J = \{(a, f(b)) | (a, b) \in I\}$ and $K = \{(b, b) | b \in Y\}$. It is clear that σ_Y is 1-1 and that $\sigma_Y[\mathbf{P}(Y, Y)] \subseteq \mathbf{Q}(Y, Y)$. It follows that $|\mathbf{P}(Y, Y)| \leqq |\mathbf{Q}(Y, Y)|$ in this case. A similar argument yields the same inequality if $|Y_A| \leqq |Y_B|$. Theorem 4.4 now implies the result (using $Z = \emptyset$).

We next show that $A : AA \leqq AA$(int). Let $\mathbf{P} = A : AA$ and $\mathbf{Q} = AA$, and let $Z \subseteq \text{Dom}(A)$ be a set with 5 distinct elements. As above, we now show for each finite $Y \subseteq \text{Dom}(A) - Z$ that $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$, and then apply Theorem 4.4 to obtain the desired result. To begin, let $Y \subseteq \text{Dom}(A) - Z$ be finite. Let $\leqq$ be a fixed total ordering of Dom (A). Let[17] $f : (YZ \times YZ) \to 2^{YZ}$ be a 1-1 function which satisfies the following conditions: For each $a \in YZ$, $f(a, a) = \{a\}$; for each pair a, a' in YZ with $a < a'$, $f(a, a') = \{a, a'\}$; and finally, $f : \{(a, a') | a > a'\} \to 2^{YZ}$ is defined so that for each pair, $\{a, a'\} \subseteq f(a, a')$. (One strategy for accomplishing this is as follows: Suppose that $Z = \{a_1, \cdots, a_5\}$, where $a_i < a_j$ in the ordering of Dom (A) iff $i < j$. Now let $a, a' \in YZ$ with $a > a'$. If $\{a, a'\} \cap Z \subseteq \{a_2, \cdots, a_5\}$, then define $f(a, a')$ to be $\{a_1, a, a'\}$. If $\{a, a'\} \cap Z = \{a_1\}$, then define $f(a, a')$ to be $\{a_2, a_3, a, a'\}$. Finally, if $a_1 \in \{a, a'\} \subseteq Z$ then $(a, a') = (a_i, a_1)$ for some i, $2 \leqq i \leqq 5$. If $2 \leqq i \leqq 3$ set $f(a, a') = \{a_1, a_i, a_4, a_5\}$, and if $4 \leqq i \leqq 5$ set $f(a, a') = \{a_1, a_2, a_3, a_i\}$. It is easily verified that f has the desired properties.) Now, given $I \in \mathbf{P}(YZ, Y)$, let $\sigma_Y(I) = \{(\bar{a}, a'') |$ for some a, a' in YZ, $(\bar{a}, a, a') \in I$ and $a'' \in f(a, a')\}$. It is easily verified that σ_Y is 1-1 and that $\sigma_Y[\mathbf{P}(YZ, Y)] \subseteq \mathbf{Q}(YZ, Y)$. Theorem 4.4 now yields that $A : AA \leqq AA$(int) as desired.

We now show for each $n > 0$ that $AB \not\leqq [AA]^n, [BB]^n$(gen) (and hence, $AB \not\leqq [AA]^n, [BB]^n$(calc)). Let $n > 0$ be fixed, and suppose to the contrary that $AB \leqq [AA]^n, [BB]^n$(gen), and more specifically that $Z \subseteq \mathbf{DOM}$ is finite, that $\sigma : AB \to [AA]^n, [BB]^n$ and $\tau : [AA]^n, [BB]^n \to AB$ are Z-generic, and that $\tau \circ \sigma$ is the identity on $\mathbf{I}(AB)$. In particular, then, σ is 1-1 on $\mathbf{I}(AB)$. Note that the range of σ is $\mathbf{I}([AA]^n, [BB]^n)$, and that each instance of $[AA]^n, [BB]^n$ is a $2n$-tuple, where each coordinate is a set of ordered pairs from Dom (A) or a set of ordered pairs from Dom (B). Let $\sigma_1 : AB \to [AA]^n$ be the transformation such that for each $I \in \mathbf{I}(AB)$, $\sigma_1(I)$ is the n-tuple consisting of the first n coordinates of $\sigma(I)$, and let $\sigma_2 : AB \to [BB]^n$

[16] If R is a specifier and $k \geqq 0$, then $[R]^k$ denotes the schema with k occurrences of R.

[17] For a set X, 2^X denotes the power set of X.

be defined analogously, that is, such that for each $I \in \mathbf{I}(AB)$, $\sigma_2(I)$ is the n-tuple consisting of the latter n coordinates of $\sigma(I)$. Note that σ_1 and σ_2 are Z-generic. Now let a, a' be in $\text{Dom}(A) - Z$; let b, b' be in $\text{Dom}(B) - Z$; and set $I = \{(a, b), (a', b')\}$ and $J = \{(a, b'), (a', b)\}$. Also, let π_A be the Z-permutation such that $\pi_A(a) = a'$, $\pi_A(a') = a$, and π_A is the identity elsewhere. Let π_B be defined analogously, with $\pi_B(b) = b'$, $\pi_B(b') = b$, and π_B the identity elsewhere. Note that π_B is the identity on $\mathbf{I}([AA]^n)$ and π_A is the identity on $\mathbf{I}([BB]^n)$. We now claim that $\sigma(I) = \sigma(J)$. To see this, first note that $\sigma_1(J) = \sigma_1(\pi_B(I)) = \pi_B(\sigma_1(I))$ (since σ_1 is Z-generic) $= \sigma_1(I)$ (since π_B is the identity on $\mathbf{I}([AA]^n)$). Similarly, $\sigma_2(J) = \sigma_2(\pi_A(I)) = \pi_A(\sigma_2(I)) = \sigma_2(I)$. This implies that $\sigma(I) = \sigma(J)$, and hence that σ is not 1-1, a contradiction. Thus $AB \not\leqq [AA]^n, [BB]^n$(gen) as desired.

Finally, the proof that $A: AA \not\leqq AA$(gen) is moderately involved, and is presented in Appendix A. □

Speaking intuitively, the fact that $A: AA \not\leqq AA$(gen) indicates that two finite functions from a set to itself cannot be generically encoded into a binary relation over that set.

As noted earlier, it remains open whether absolute and internal dominance can be distinguished in the context of information capacity dominance between (keyed) relation schemas, or whether generic and calculous dominance can be distinguished in this context. With regard to the latter question, it is interesting to recall the result of [1] which states that the generic query operation of transitive closure (of a binary relation) is not realizable by any calculus expression. In other words, the notions of generic and calculous can be distinguished in the context of query operations.

Returning to general results, we now present several sufficient conditions for inferring dominance of one sort or another. The first result presents cases where the existence of one transformation (from **P** to **Q**) rather than two (both from **P** to **Q** and back) are needed. For this result we use terminology of [34].

Notation. Let **P** and **Q** be schemata. Then **P** is *internally embeddable* in **Q** if there is a finite set $Z \subseteq \mathbf{DOM}$ and a 1-1 Z-internal transformation $\sigma: \mathbf{P} \to \mathbf{Q}$. The notions of *generically embeddable* and *calculously embeddable* are defined analogously.

We now have:

THEOREM 4.6. *Let* **P** *and* **Q** *be schemata. Then*

(a) $\mathbf{P} \leqq \mathbf{Q}$(gen) *iff* **P** *is generically embeddable in* **Q**; *and*

(b) $\mathbf{P} \leqq \mathbf{Q}$(int) *iff* **P** *is internally embeddable in* **Q**.

Proof. We give the proof of part (a) here, and present the more involved proof of part (b) in Appendix B. For part (a), it is clear that if $\mathbf{P} \leqq \mathbf{Q}$(gen) then **P** is generically embeddable in **Q**. For the converse, suppose now that $\sigma: \mathbf{P} \to \mathbf{Q}$ is a 1-1 Z-generic transformation for some finite $Z \subseteq \mathbf{DOM}$. Define $\tau: \mathbf{Q} \to \mathbf{P}$ so that $\tau(J) = I$ for each $J \in \sigma[\mathbf{I}(\mathbf{P})]$ where $J = \sigma(I)$, and $\tau(J) = \emptyset$ for each $J \in \mathbf{I}(\mathbf{Q}) - \sigma[\mathbf{I}(\mathbf{P})]$. It is clear that $\tau \circ \sigma$ is the identity on $\mathbf{I}(\mathbf{P})$; it therefore suffices to show that τ is Z-generic.

Let π be a Z-permutation, and let $J \in \mathbf{I}(\mathbf{Q})$. Suppose that $J \notin \sigma[\mathbf{I}(\mathbf{P})]$. This implies that $\pi(J) \notin \sigma[\mathbf{I}(\mathbf{P})]$ as well. (Otherwise, $\pi(J)$ would equal $\sigma(I')$ for some $I' \in \mathbf{I}(\mathbf{P})$. But then $J = \pi^{-1}(\pi(J)) = \pi^{-1}(\sigma(I')) = \sigma(\pi^{-1}(I')) \in \sigma[\mathbf{I}(\mathbf{P})]$, a contradiction.) We now have $\pi(\tau(J)) = \pi(\emptyset) = \emptyset = \tau(\pi(J))$.

Now suppose that $J = \sigma(I)$ for some $I \in \mathbf{I}(\mathbf{P})$. Then $\tau(J) = I$. Also, $\pi(J) = \pi(\sigma(I)) = \sigma(\pi(I))$, and so $\tau(\pi(J)) = \pi(I)$. Thus, $\tau(\pi(J)) = \pi(\tau(J))$ as desired. □

The techniques used to prove the above theorem are quite general, and so it appears that these results also hold in more general database models. In particular, both parts of this theorem hold in the Format Model [20] (assuming that the definition of family of instances used there is modified in analogy to the definition used here). It remains open whether the theorem also holds for calculous dominance.

The next result examines the impact of changing the basic types occurring in schemata. Speaking informally, the result states that dominance is preserved by renamings of basic types (even if different basic types are identified by the renaming.) For this result we use:

DEFINITION. A *homomorphism* (*on* $\mathscr{B}$) is a function $h: \mathscr{B} \to \mathscr{B}$. If h is a homomorphism and R is a nonkeyed specifier, then $h(R)$ denotes the nonkeyed specifier U where $U(A) = \sum_{h(B)=A} R(B)$ for each basic type A. If R and S are nonkeyed specifiers then $h(R:S) = h(R):h(S)$, and if $\mathbf{P} = P_1, \cdots, P_n$ is a schema then $h(\mathbf{P}) = h(P_1), \cdots, h(P_n)$.

The proof of this theorem is straightforward but lengthy, and is presented in Appendix C.

THEOREM 4.7. *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be schemata, and* h *a homomorphism on* $\mathscr{B}$. *If* $\mathbf{P} \leqq \mathbf{Q}$(xxx) *then* $h(\mathbf{P}) \leqq h(\mathbf{Q})$(xxx), *where "xxx" ranges over* "calc", "gen", "int", *and* "abs".

As an application of this result, note that since $A:AA \not\leqq AA$(gen, calc) by Theorem 4.7, we also have $A:BB \not\leqq AB$(gen, calc). Also, the argument used to show that $A:AA \leqq AA$(abs, int) in the proof of that theorem can be modified to show that $A:BB \leqq AB$(abs, int).

The converse of Theorem 4.7 does not hold for any of the types of dominance, since $ABB \not\leqq AAB$(xxx) (by Corollary 4.3) but $AAA \leqq AAA$(xxx), where "xxx" ranges over each of the four types of dominance.

We conclude the section with three relatively simple results for inferring dominance between two schemata, given dominance by two other schemata. The first one provides for a kind of "additivity" among schemata. (The straightforward proof is omitted.)

PROPOSITION 4.8. *Let* $\mathbf{P}$, $\mathbf{Q}$, *and* $\mathbf{R}$ *be schemata. Then*

(a) $\mathbf{P} \leqq \mathbf{Q}$(xxx) $\Rightarrow \mathbf{RP} \leqq \mathbf{RQ}$(xxx), *where* "xxx" *ranges over* "calc", "gen", *and* "int"; *and*

(b) $\mathbf{P} \leqq \mathbf{Q}$(abs) $\Leftrightarrow \mathbf{RP} \leqq \mathbf{RQ}$(abs). □

It remains open whether the converse of part (a) in the above result holds for any of calculous, generic or internal dominance.

Two applications of the above result along with the transitivity of dominance yield (proof omitted):

COROLLARY 4.9. *Let* $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$, *and* $\mathbf{S}$ *be schemata, with* $\mathbf{P} \leqq \mathbf{Q}$(xxx) *and* $\mathbf{R} \leqq \mathbf{S}$(xxx). *Then* $\mathbf{PR} \leqq \mathbf{QS}$(xxx), *where* "xxx" *ranges over* "calc", "gen", "int", *and* "abs". □

The final result provides for a kind of additivity within single specifiers. (The straightforward proof is again omitted.) It remains open whether the converse of part (a) of this result holds.

PROPOSITION 4.10. *Let* R, S, U, V *and* T *be nonkeyed specifiers. Then*

(a) $R:S \leqq U:V$(xxx) $\Rightarrow TR:S \leqq TU:V$(xxx), *where* "xxx" *ranges over* "calc", "gen", *and* "int"; *and*

(b) $R:S \leqq U:V$(abs) $\Leftrightarrow TR:S \leqq TU:V$(abs). □

5. Calculous dominance vs. semantic correspondence. In this section we present results which indicate that the notion of calculous dominance (and hence, query-dominance) does not accurately reflect or measure the presence of "semantic correspondence" between schemata. Our first result, Theorem 5.2, characterizes calculous dominance between nonkeyed relational schemata, where the dominating schema consists of a single specifier. This result implies that calculous dominance holds in a variety of counter-intuitive situations (see Example 5.3). The section concludes with a result which implies that each of the types of dominance is the same in the context of nonkeyed relational schemata which involve only one basic type.

For the first result, we use the natural partial ordering of nonkeyed specifiers, considered simply as multisets.

Notation. For nonkeyed specifiers R and S, write $R \subseteqq S$ if $R(B) \leqq S(B)$ for each $B \in \mathcal{B}$. Write $R \subset S$ if $R \subseteqq S$ and $R(B) < S(B)$ for some $B \in \mathcal{B}$.

Using an argument based on cardinalities, it is easily verified that (proof omitted):

LEMMA 5.1. *Let R and S be nonkeyed specifiers. Then $R \leqq S$(abs) iff $R \subseteqq S$ and $R \sim S$(abs) iff $R = S$.* □

We now have:

THEOREM 5.2. *Let $\mathbf{P} = R_1, \cdots, R_n$ be a nonkeyed schema and S a nonkeyed specifier. Then the following are equivalent*:

(a) $\mathbf{P} \leqq S$(calc);

(b) $\mathbf{P} \leqq S$(gen);

(c) $\mathbf{P} \leqq S$(int);

(d) $\mathbf{P} \leqq S$(abs); *and*

(e) *either $n = 1$ and $R_1 \subseteqq S$, or $n > 1$ and $R_i \subset S$ for each j, $1 \leqq j \leqq n$.*

Before proving this result we present an example illustrating its significance.

Example 5.3. Assume that a (large) set of NAME-values and a (large) set of NUMBER-values is fixed (where there is no ordering or other predicate on either of these sets). Suppose further that a relation scheme **R** consists of 50 relations R_i, each of which has one column with NAME-values and two columns with NUMBER-values; and also 50 relations S_j, each of which has two columns with NAME-values and one column with NUMBER-values. Also, let **T** be a relation scheme with a single relation in it, where that relation has two columns for NAME-values and two columns for NUMBER-values. By Theorem 5.2, $\mathbf{R} \leqq \mathbf{T}$(calc) and so **R** is query-dominated by **T**. Since it appears that there is no intuitively appealing, semantically meaningful 1-1 mapping of instances of **R** to instances of **T**, this indicates that query-dominance does not accurately measure whether there is a "semantic" connection between pairs of schemata. □

We now consider the proof of Theorem 5.2. In view of Theorem 3.4, it suffices to show that (d)⇒(e) and that (e)⇒(a). The first of these is given by:

LEMMA 5.4. *In the statement of Theorem* 5.2, (d) *implies* (e).

Proof. Suppose that $\mathbf{P} \leqq S$(abs). If $n = 1$, then $R_1 \subseteqq S$ by Lemma 5.1. Suppose now that $n > 1$, and suppose further that $R_j \not\subset S$ for some j, $1 \leqq j \leqq n$. Without loss of generality we can assume that $j = 1$. If $R_1 \neq S$, then $R_1 \not\subseteqq S$, from which Lemma 5.1 implies that $R_1 \not\leqq S$(abs). Since $R_1 \leqq R_1, \cdots, R_n$(abs) (by Corollary 4.9, along with the fact that $\varnothing$ is dominated by every schema) and absolute dominance is transitive, this implies that $\mathbf{P} = R_1, \cdots, R_n \not\leqq S$(abs) in this case.

Finally, suppose that $R_1 = S$. It is then clear that $f_{\mathbf{P}}(\mathbf{x}) - f_S(\mathbf{x}) = f_{R_2,\cdots,R_n}(\mathbf{x}) > 0$ for each $\mathbf{x} = (x_1, \cdots, x_n)$ (with $x_j > 0$, $1 \leqq j \leqq n$). Thus, $\mathbf{P} \not\leqq S$(abs) also holds in this case. □

We now present a lemma which lies at the heart of the proof that (e)⇒(a) in Theorem 5.2. Speaking intuitively, this lemma examines the "worst case" values for **P** and S. The reader will note that in the lemma, each basic type occurs only once in the specifier S. This result will be used in connection with Theorem 4.7 to yield the general result.

LEMMA 5.5. *Let $S = B_1 \cdots B_m$ be a nonkeyed specifier* (*with $B_i \neq B_j$ for $i \neq j$*), *let $R_j = B_1 \cdots B_{j-1} B_{j+1} \cdots B_m$ for each j, $1 \leqq j \leqq m$, and let $\mathbf{P} = [R_1]^n, \cdots, [R_m]^n$ for some $n > 0$. Then $\mathbf{P} \leqq S$(calc).*

Proof. To establish this lemma, we describe a 1-1 function $\gamma: \mathbf{I}(\mathbf{P}) \to \mathbf{I}(S)$ which has the properties that (i) γ is realizable by a calculus expression, and (ii) the inverse of γ (on $\gamma[\mathbf{I}(\mathbf{P})]$) is realizable by a calculus expression. The function γ will be defined as $\beta \circ \alpha$, where α and β are defined below.

Let $\mathcal{K} = \bigtimes_{1 \leq j \leq m} [\text{Dom}(B_j) \cup \{1, \cdots, n\}]$, and define $\alpha : \mathbf{I}(\mathbf{P}) \to 2^{\mathcal{K}}$ by

$$\alpha(I_1^1, \cdots, I_1^n, \cdots, I_m^1, \cdots, I_m^n) = \{(b_1, \cdots, b_{j-1}, i, b_{j+1}, \cdots, b_n) | (b_1, \cdots, b_{j-1}, b_{j+1}, \cdots, b_n) \in I_j^i\}.$$

Intuitively, $\mathcal{K}$ is essentially a subset of $\mathbf{I}(S)$, except that each tuple of $\mathcal{K}$ includes one integer between 1 and n. Note that the function α is a straightforward encoding of instances of $\mathbf{P}$ into subsets of $\mathcal{K}$, and that α is 1-1.

Let $h > n \cdot m$, and for each j, $1 \leqq j \leqq m$, choose $Z_j = \{z_j^1, \cdots, z_j^h\} \subseteqq \text{Dom}(B_j)$. (Speaking intuitively for a moment, it appears that it is now sufficient to define $\beta : \alpha[\mathbf{I}(\mathbf{P})] \to \mathbf{I}(S)$ so that it maps J in $\alpha[\mathbf{I}(\mathbf{P})]$ to $\{(b_1, \cdots, b_{j-1}, z_j^i, b_{j+1}, \cdots, b_m) | (b_1, \cdots, b_{j-1}, i, b_{j+1}, \cdots, b_m) \in J\}$. In other words, β would encode each tuple $(b_1, \cdots, b_{j-1}, i, b_{j+1}, \cdots, b_m)$ in J by the tuple $(b_1, \cdots, b_{j-1}, z_j^i, b_{j+1}, \cdots, b_m)$ in $\beta(J)$. However, a problem arises if for some q, $b_q \in Z_q$. For example, suppose that $q > j$, and that $b_q = z_q^p$. In this case, it will be unclear whether the tuple $(b_1, \cdots, z_j^i, \cdots, z_q^p, \cdots, b_m)$ is an encoding of $(b_1, \cdots, i, \cdots, z_q^p, \cdots, b_m)$ or $(b_1, \cdots, z_j^i, \cdots, p, \cdots, b_m)$. For this reason, a more sophisticated encoding must be used.)

Let $U = \{B_1, \cdots, B_m\}$ $(= \text{supp}(S))$. For each set V satisfying $\varnothing \subsetneqq V \subseteqq U$, let

$$I_V = \left\{ w \in \bigtimes_{B_j \in V} [Z_j \cup \{1, \cdots, n\}] \middle| \text{ exactly one coordinate of } w \text{ is in } \{1, \cdots, n\} \right\};$$

and

$$J_V = \bigtimes_{B_j \in V} Z_j.$$

Note that $|I_V| = n \cdot |V| \cdot h^{|V|-1}$ and $|J_V| = h^{|V|}$. For each nonempty $V \subseteqq U$ we have $n \cdot |V| \leqq n \cdot m < h$, and hence $|I_V| \leqq |J_V|$. This implies for each such V that there is a 1-1, total function $\beta_V : I_V \to J_V$. Note that since I_V is finite for each nonempty $V \subseteqq U$, β_V is finite and thus, intuitively speaking, describable using first order predicate calculus.

We now define $\beta : \alpha[\mathbf{I}(\mathbf{P})] \to \mathbf{I}(S)$ by[18]

$$\beta(I) = \{w[U - V] \beta_V(w[V]) | w = (w_1, \cdots, w_m) \in I \text{ and } V = \{B_j | w_j \in Z_j \cup \{1, \cdots, n\}\}\}.$$

Note that for each $I \in \alpha[\mathbf{I}(\mathbf{P})]$, $|\beta(I)| = |I|$ and for each $\mathbf{I} = (I_1^1, \cdots, I_1^n, \cdots, I_m^1, \cdots, I_m^n) \in \mathbf{I}(\mathbf{P})$, $|\beta \circ \alpha(\mathbf{I})| = \sum |I_j^i|$. Clearly β is 1-1 and it is straightforward to verify that it is realizable by a first order predicate calculus expression. It follows that the mapping $\gamma = \beta \circ \alpha : \mathbf{I}(\mathbf{P}) \to \mathbf{I}(S)$ is 1-1 and realizable by a relational calculus expression. Finally, it is also clear that there is a calculus expression $\delta : \mathbf{I}(S) \to \mathbf{I}(\mathbf{P})$ such that for each $J \in \mathbf{I}(S)$, if $J \in \gamma[\mathbf{I}(\mathbf{P})]$ then $\delta(J)$ is the unique I such that $\gamma(I) = J$; and if $J \notin \gamma[\mathbf{I}(\mathbf{P})]$ then $\delta(J) = \varnothing$. This completes the proof. □

We now have:

Conclusion of proof of Theorem 5.2. As previously observed, Theorem 3.4 implies that (a)⇒(b), (d)⇒(c), and (c)⇒(d) in the statement of Theorem 5.2. By Lemma 5.4 we have (d)⇒(e), and so it now suffices to show that (e)⇒(a).

To this end, suppose that (e) holds. If $n = 1$ then clearly (a) holds. Now let $\mathbf{P} = R_1, \cdots, R_n$ be a nonkeyed relational schema with $n > 1$ satisfying (e), and let $S = A_1^{k_1} \cdots A_m^{k_m}$ be a nonkeyed relation specifier such that $R_j \subset S$ for each j, $1 \leqq j \leqq n$. Let $\{B_1, \cdots, B_t\}$ be a set of new basic types, where $t = |S| = \sum_{1 \leq i \leq m} k_i$, and let the

[18] If $x = (x_1, \cdots, x_m) \in \mathcal{K}$ and $V = \{B_{i_1}, \cdots, B_{i_{|V|}}\} \subseteqq \{B_1, \cdots, B_m\}$ (where $i_1 < i_2 < \cdots < i_{|V|}$), then $x|V|$ denotes the tuple $(x_{i_1}, \cdots, x_{i_{|V|}})$. Also, if $x = (x_1, \cdots, x_{|V|}) \in \bigtimes_{B \in V} \text{Dom}(B)$ and $y = (y_1, \cdots, y_{|U-V|}) \in \bigtimes_{B \in U-V} \text{Dom}(B)$, then xy denotes the (unique) tuple $z = (z_1, \cdots, z_m) \in \bigtimes_{B \in U} \text{Dom}(B)$ formed from x and y in the natural manner (i.e., where z_i is the unique element of $\text{Dom}(B_i) \cap \{x_1, \cdots, x_{|V|}, y_1, \cdots, y_{|U-V|}\}$; this is well defined, since each of the basic types of U occurs only once in S).

function $h:\{B_1, \cdots, B_t\} \to \{A_1, \cdots, A_n\}$ be defined so that $h(B_s) = A_i$ for each s, $\sum_{1 \leqq j < i} k_j < s \leqq \sum_{1 \leqq j \leqq i} k_j$. For each j, $1 \leqq j \leqq n$, let R'_j be a nonkeyed specifier such that $\text{supp}(R'_j) \subseteq \{B_1, \cdots, B_t\}$, $h(R'_j) = R_j$, and each basic type occurs at most once in R'_j. Also, let $\mathbf{P}' = R'_1, \cdots, R'_n$; and let $S' = B_1 \cdots B_t$. Note that $R'_j \subset S'$ for each j, $1 \leqq j \leqq n$. Also, $h(\mathbf{P}') = \mathbf{P}$ and $h(S') = S$. Theorem 4.7 now implies that in order to show that $\mathbf{P} \leqq S(\text{calc})$ it suffices to show that $\mathbf{P}' \leqq S'(\text{calc})$. To accomplish this, we shall "expand" $\mathbf{P}'$ into a new schema $\mathbf{P}'''$ which has the properties that (i) $\mathbf{P}' \leqq \mathbf{P}'''(\text{calc})$; and (ii) $\mathbf{P}'''$ has the form of the schema $\mathbf{P}$ in Lemma 5.5, whence $\mathbf{P}''' \leqq S'(\text{calc})$.

To begin, for each j, $1 \leqq j \leqq n$, let R''_j be chosen such that $R'_j \subseteq R''_j$; $R''_j \subset S'$; and $|R''_j| = |S'| - 1$. (This is accomplished by "adding" basic types from $\{B_1, \cdots, B_s\}$ to R_j until all but one have been used.) Letting $\mathbf{P}'' = R''_1, \cdots, R''_m$ it is clear that $\mathbf{P}' \leqq ''(\text{calc})$.

Now let $T_1, \cdots, T_t$ be an enumeration of all specifiers T such that $T \subset S'$ and $|T| = |S'| - 1$, and let $q \geqq 0$ be the maximum number of occurrences of any of these specifiers in $\mathbf{P}''$. Finally, let $\mathbf{P}''' = [T_1]^q, \cdots, [T_t]^q$. It is now clear that $\mathbf{P}'' \leqq \mathbf{P}'''(\text{calc})$. And since $\mathbf{P}'''$ and S' satisfy the conditions of Lemma 5.5, we have $\mathbf{P}''' \leqq S'(\text{calc})$. By two applications of transitivity, $\mathbf{P}' \leqq S'(\text{calc})$. Finally, Theorem 4.7 yields $\mathbf{P} \leqq S(\text{calc})$ as desired. □

We briefly consider a natural analog of Theorem 5.2 for keyed specifiers. Specifically, suppose that the statement of the theorem were changed to allow keyed specifiers, and that condition (e) were changed to read "either $n = 1$ and $P_1 \leqq R(\text{xxx})$, or $n > 1$ and both $P_i \leqq R(\text{xxx})$ and $P_i \neq R$ for each i, $1 \leqq i \leqq n$" (where "xxx" ranges over one (or more) of the four types of dominance). A counterexample to this modified version of the theorem is easily obtained. For example, it is easily verified that $A:B \leqq A:BB(\text{calc})$ (and hence for each of the types of dominance) but $[A:B]^3 \not\leqq A:BB(\text{abs})$ (and hence for none of them).

The final result of this section concerns dominance between nonkeyed schemata which involve only one basic type. (Note that many of the early investigations of the relational calculus were essentially based on relational schemata of this type.)

THEOREM 5.6. *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be nonkeyed relational schemata over a single basic type* B. *Then the following are equivalent*:

(a) $\mathbf{P} \leqq \mathbf{Q}(\text{calc})$;
(b) $\mathbf{P} \leqq \mathbf{Q}(\text{gen})$;
(c) $\mathbf{P} \leqq \mathbf{Q}(\text{int})$;
(d) $\mathbf{P} \leqq \mathbf{Q}(\text{abs})$; *and*
(e) *the*[19] *leading coefficient* (*as a polynomial*) *of* $f_{\mathbf{Q}}(x) - f_{\mathbf{P}}(x)$ *is a nonnegative integer.*

Proof. In light of Theorem 3.4, it suffices to show that (d)⇒(e) and (e)⇒(a). To see that (d)⇒(e), suppose that $\mathbf{P} \leqq \mathbf{Q}(\text{abs})$. Theorem 4.2 now implies that there is some $t \geqq 0$ such that for each $x \geqq t$, $f_{\mathbf{P}}(x) \leqq f_{\mathbf{Q}}(x)$. It follows that the leading coefficient of $f_{\mathbf{Q}} - f_{\mathbf{P}}$ is nonnegative, i.e., that (e) holds.

To complete the proof, assume now that (e) holds, and assume that $\mathbf{P}$ and $\mathbf{Q}$ are defined over the basic type B. If the leading coefficient of $f_{\mathbf{Q}} - f_{\mathbf{P}}$ is 0, then $f_{\mathbf{P}} = f_{\mathbf{Q}}$, whence $\mathbf{P} = \mathbf{Q}$. In this case, clearly $\mathbf{P} \leqq \mathbf{Q}(\text{calc})$. Suppose now that the leading coefficient of $f_{\mathbf{Q}} - f_{\mathbf{P}}$ is positive. Then there is some $n > 0$, some k with $0 < k \leqq n$, and some nonnegative integers p_i and q_i $(1 \leqq i \leqq n)$ such that $(\alpha) f_{\mathbf{P}}(x) = p_n x^n + \cdots + p_1 x$, $(\beta) f_{\mathbf{Q}}(x) = q_n x^n + \cdots + q_1 x$, $(\gamma) q_j = p_j$ for $k < j \leqq n$, and (δ) $q_k > p_k \geqq 0$. (Thus, $\mathbf{Q}$ has

[19] The *leading coefficient* of a polynomial of one variable is the coefficient of the term having the highest exponent. If the polynomial is identically 0, the leading coefficient is defined to be 0.

q_n occurrences of A^n, q_{n-1} occurrences of A^{n-1}, etc., and similarly for $\mathbf{P}$; and the coefficients of the terms for n down to $k+1$ agree.)

Let $\mathbf{P}_1 = [A^n]^{p_n}, [A^{n-1}]^{p_{n-1}}, \cdots, [A^{k+1}]^{p_{k+1}}$, let $\mathbf{P}_2 = [A^k]^{p_k}$, and let $\mathbf{P}_3 = \mathbf{P} - \mathbf{P}_1\mathbf{P}_2$. Also, let $\mathbf{Q}_1 = \mathbf{P}_1$, $\mathbf{Q}_2 = \mathbf{P}_2$, $\mathbf{Q}_2' = [A^k]^{q_k - p_k}$, and $\mathbf{Q}_3 = \mathbf{Q} - \mathbf{Q}_1\mathbf{Q}_2\mathbf{Q}_2'$. Since $\mathbf{P}_1 = \mathbf{Q}_1$ and $\mathbf{P}_2 = \mathbf{Q}_2$, $\mathbf{P}_1 \leqq \mathbf{Q}_1$(calc) and $\mathbf{P}_2 \leqq \mathbf{Q}_2$(calc). Since $q_k > p_k$, $\mathbf{Q}_2' \neq \varnothing$, and so Theorem 5.2 implies $\mathbf{P}_3 \leqq \mathbf{Q}_2'$(calc). Finally, $\varnothing \leqq \mathbf{Q}_3$(calc). Three applications of Corollary 4.9 now yield $\mathbf{P} = \mathbf{P}_1\mathbf{P}_2\mathbf{P}_3 \leqq \mathbf{Q}_1\mathbf{Q}_2\mathbf{Q}_2'$(calc), and furthermore, $\mathbf{Q}_1\mathbf{Q}_2\mathbf{Q}_2' \leqq \mathbf{Q}_1\mathbf{Q}_2\mathbf{Q}_2'\mathbf{Q}_3$(calc). This last term is equal to $\mathbf{Q}$, and so we have $\mathbf{P} \leqq \mathbf{Q}$(calc) as desired. □

It should be noted that the above result cannot be generalized to apply to pairs of relational schemata including keyed specifiers. In particular, Proposition 4.5(b) shows that $A: A^2 \leqq A^2$(abs, int) but $A: A^2 \not\leqq A^2$(gen, calc).

6. Equivalence implies equality. This section presents the result that if $\mathbf{P}$ and $\mathbf{Q}$ are nonkeyed schemata and they are equivalent under any of the notions of dominance, then $\mathbf{P} = \mathbf{Q}$ (up to re-ordering). A corollary of this result (presented at the end of the section) implies that for any "natural" motion of information capacity equivalence, a pair of nonkeyed relational schemata are equivalent if and only if they are equal. This supports the intuition that in the relational model (with no dependencies) there is essentially at most one way to represent a given data set. It is interesting to note that a result of [20] concerning the Format Model provides contrast with the results mentioned here. In particular, it is shown there that two formats may be generically equivalent and yet be unequal.[20]

To begin the development, we present some notation and a straightforward lemma concerning polynomials of several variables. Speaking informally, the lemma states that such a polynomial is ultimately zero iff all of its coefficients are zero.

Notation. If $\mathbf{x} = (x_1, \cdots, x_n)$ and $\mathbf{p} = (p_1, \cdots, p_n)$, then $\mathbf{x}^{\mathbf{p}}$ denotes $\prod_{i=1}^{n} x_i^{p_i}$.

LEMMA 6.1. *Let $n > 0$, $H \subseteq N^n$ be a finite set, and for each $\mathbf{p} \in H$ let $a_{\mathbf{p}}$ be an integer. Let $f: \mathbf{N}^n \to \mathbf{N}$ be defined by $f(\mathbf{x}) = \sum_{\mathbf{p} \in H} a_{\mathbf{p}} \mathbf{x}^{\mathbf{p}}$. If there is some $t \geqq 0$ such that $f(\mathbf{x}) = 0$ for each $\mathbf{x}$ with $x_i \geqq t$ $(1 \leqq i \leqq n)$, then $a_{\mathbf{p}} = 0$ for each $\mathbf{p} \in H$.*

Proof. Suppose that at least one of the terms $a_{\mathbf{p}}$ is nonzero. Define the ordering $<$ on $\mathbf{N}^n$ by $(y_1, \cdots, y_n) < (z_1, \cdots, z_n)$ if there is some k, $1 \leqq k \leqq n$, such that $y_i = z_i$ for $k < i \leq n$ and $y_k < z_k$. (Intuitively speaking, $<$ is the reverse lexicographic ordering on $\mathbf{N}^n$.) Let $\mathbf{q}$ be the greatest element $\mathbf{p}$ in H (under $<$) with the property that $a_{\mathbf{p}} \neq 0$. Without loss of generality, we may assume that $a_{\mathbf{q}} > 0$. Thus $a_{\mathbf{q}} \geqq 1$.

Let $y_0 = \max\{t, |H| \cdot \sum_{\mathbf{p} \in H} |a_{\mathbf{p}}|\}$ and $s > \max\{p_i | 1 \leqq i \leqq n \text{ and } \mathbf{p} \in H\}$. *For each i,* $1 \leqq i \leqq n$, set $y_i = y_{i-1}^{2s} = y_0^{(2s)^i}$, and let $\mathbf{y} = (y_1, \cdots, y_n)$. It is clear that $y_i \geqq t$ for each i, $1 \leqq i \leqq n$, and hence that $f(\mathbf{y}) = 0$.

A straightforward induction shows that for each i, $0 \leqq i < n$, $y_{i+1} > y_0 \cdot \prod_{j \leqq i} y_j^s$. (Clearly this is true for $i = 0$. Supposing it is true for $i-1$, then $y_{i+1} = y_i^{2s} \geqq y_i \cdot y_i^s > (y_0 \cdot \prod_{j \leqq i-1} y_j^s) \cdot y_i^s = y_0 \cdot \prod_{j \leqq i} y_j^s$.) Now suppose that $\mathbf{p}$ and $\mathbf{p}'$ are in H with $\mathbf{p} < \mathbf{p}'$. It follows that $y_0 \cdot \mathbf{y}^{\mathbf{p}} < \mathbf{y}^{\mathbf{p}'}$. (For suppose that $\mathbf{p} = (p_1, \cdots, p_n)$ and $\mathbf{p}' = (p_1', \cdots, p_k', p_{k+1}, \cdots, p_n)$ where $1 \leqq k \leqq n$ and $p_k' > p_k$. Then $y_0 \cdot \mathbf{y}^{\mathbf{p}} = y_0 \cdot \prod_{i \leqq k} y_i^{p_i} \cdot \prod_{i > k} y_i^{p_i} \leqq [y_0 \cdot \prod_{i < k} y_i^s] \cdot y_k^{p_k} \cdot \prod_{i > k} y_i^{p_i} < [y_k] \cdot y_k^{p_k} \cdot \prod_{i > k} y_i^{p_i} \leqq \prod_{i \leqq k} y_i^{p_i'} \cdot \prod_{i > k} y_i^{p_i} = \mathbf{y}^{\mathbf{p}'}$.)

Since $f(\mathbf{y}) = 0$ and by the choice of $\mathbf{q}$, $a_{\mathbf{q}}\mathbf{y}^{\mathbf{q}} = \sum_{\mathbf{p} \in H, \mathbf{p} < \mathbf{q}} (-a_{\mathbf{p}})\mathbf{y}^{\mathbf{p}}$. However, $a_{\mathbf{q}}\mathbf{y}^{\mathbf{q}} \geqq \mathbf{y}^{\mathbf{q}} > y_0 \cdot \max\{\mathbf{y}^{\mathbf{p}} | \mathbf{p} \in H, \ \mathbf{p} < \mathbf{q}\} \geqq |H| \cdot \sum_{\mathbf{p} \in H} |a_{\mathbf{p}}| \cdot \max\{\mathbf{y}^{\mathbf{p}} | \mathbf{p} \in H, \mathbf{p} < \mathbf{q}\} \geqq \sum_{\mathbf{p} \in H, \mathbf{p} < \mathbf{q}} |a_{\mathbf{p}}| \cdot \mathbf{y}^{\mathbf{p}} \geqq$

[20] Furthermore, in [20], it is shown that two formats are generically equivalent if and only if one can be formed from the other using a set of six natural, local, structural transformations (called "reductions") and their inverses.

$\sum_{\mathbf{p}\in H,\mathbf{p}<\mathbf{q}} (-a_{\mathbf{p}})\mathbf{y}^{\mathbf{p}}$. This is a contradiction, and implies that each of the terms $a_{\mathbf{p}}$ in H are zero as originally desired. □

We now have:

THEOREM 6.2. *Let* **P** *and* **Q** *be nonkeyed relational schemata. Then the following are equivalent:*

(a) $\mathbf{P} \sim \mathbf{Q}$(calc);
(b) $\mathbf{P} \sim \mathbf{Q}$(gen);
(c) $\mathbf{P} \sim \mathbf{Q}$(int);
(d) $\mathbf{P} \sim \mathbf{Q}$(abs);
(e) $\mathbf{P} = \mathbf{Q}$.

Proof. In view of Theorem 3.4, and the trivial implication (e)⇒(a), it suffices to prove that (d)⇒(e). To this end, let us assume that (d) holds, i.e., that $\mathbf{P} \sim \mathbf{Q}$(abs). Letting $\{B_1, \cdots, B_n\}$ be an enumeration of the basic types in **P** and **Q**, Theorem 4.2 implies that there is a number $t \geqq 0$ such that $f_{\mathbf{P}}(\mathbf{x}) = f_{\mathbf{Q}}(\mathbf{x})$ for each $\mathbf{x} = (x_1, \cdots, x_n)$ with $x_i \geqq t$, $1 \leqq i \leqq n$. From Lemma 6.1, each coefficient of $f(\mathbf{x}) = f_{\mathbf{P}}(\mathbf{x}) - f_{\mathbf{Q}}(\mathbf{x})$ is 0, and so $\mathbf{P} = \mathbf{Q}$ as desired. □

As noted in § 2, the family of nonkeyed relational schemata corresponds to the family for formats [20] which are constructed using a composition of one or more subformats, each of which is a collection of a composition of one or more basic types. An alternative proof of the above theorem can be developed using this correspondence and Theorem 4.6 of [20].

Also, while it appears that the above theorem can be generalized to keyed relational schemata, no proof is currently known.

The following corollary of Theorem 6.2 can be interpreted as a result concerning any "natural" notion of relative information capacity dominance in the following sense: The corollary considers all relative information partial relationships $\ll$ which lie in the "region" between equality (i.e., isomorphism) and absolute dominance. Since we would expect any "natural" measure of information capacity (which does not involve computation) to lie in this region, the corollary applies to all such "natural" measures.

COROLLARY 6.3. *Let* $\ll$ *be any binary relation on nonkeyed relational schemata such that for each pair* **P**, **Q** *of schemata,*

(a) $\mathbf{P} \ll \mathbf{Q} \Rightarrow \mathbf{P} \leqq \mathbf{Q}$(abs); *and*
(b) $\mathbf{P} = \mathbf{Q} \Rightarrow \mathbf{P} \ll \mathbf{Q}$.

Also, let $\asymp$ *be defined so that* $\mathbf{P} \asymp \mathbf{Q}$ *iff* $\mathbf{P} \ll \mathbf{Q}$ *and* $\mathbf{Q} \ll \mathbf{P}$. *Then for each pair* **P**, **Q** *of schemata,* $\mathbf{P} \asymp \mathbf{Q}$ *iff* $\mathbf{P} = \mathbf{Q}$. □

7. Concluding remarks. The model of relative information capacity introduced here can serve as part of the foundation for the theoretical study of data relativism as it arises in a variety of database areas. While several interesting results have been reported above, the investigation also raises a number of important questions. In this section some of these are briefly mentioned.

A major area demanding further investigation is to seek measures of relative information capacity other than the ones presented here. For example, the results of § 5 indicate that none of the types of dominance studied here correspond closely to the intuitive notion of "semantic correspondence". A more "natural" measure might be obtained by modifying calculous dominance (and the other types as well) so that constants are not allowed. Alternatively, other "abstract" properties of natural database transformations (such as being generic or internal) might be formalized and investi-

gated. For example, a transformation $\sigma: \mathbf{P} \to \mathbf{Q}$ can be called *additive* if for each I and I', we have[21] $\sigma(I \cup I') = \sigma(I) \cup \sigma(I')$.

Another direction is to consider a broader notion of information capacity which incorporates update capabilities. For example, suppose that some update language UL is fixed (e.g., one consisting of all compositions of the primitive operations of delete one tuple, insert one tuple, and update one tuple). Define $\mathbf{P} \leqq \mathbf{Q}(UL)$ if (i) $\mathbf{P} \leqq \mathbf{Q}$(calc) via calculus expressions ξ and ω, *and* if (ii) whenever $I \in \mathbf{I}(\mathbf{P})$ and μ is an update in UL of I then there is an update μ' in UL of $\xi(I)$ such that $\mu'(\xi(I)) = \xi(\mu(I))$. This approach may give insight into the "view-update" problem.

A third general area is to consider the various complexity issues raised by this investigation. For example, what is the complexity of deciding whether $\mathbf{P} \leqq \mathbf{Q}$(xxx)? If it is known, say, that $\mathbf{P} \leqq \mathbf{Q}$(calc), then how hard is it to find a pair ξ, ω of calculus expressions such that $\mathbf{P} \leqq \mathbf{Q}$ via (ξ, ω)? Finally, it is also important to examine the complexity of actually performing ξ and ω on instances of $\mathbf{P}$ and $\mathbf{Q}$, that is, to examine the ease of translating between these two schemata.

A variety of other issues can be studied using the model of information capacity presented here. For example, it would be interesting to examine relative information capacity between schemata taken from database models besides the relational one; and to examine relative information capacity between schemata $\mathbf{P}$ and $\mathbf{Q}$, where $\mathbf{P}$ is taken from one model and $\mathbf{Q}$ from another (similar to [27]). Within the relational model, it would be useful to examine the impact of null values and new dependencies.

Finally, a number of specific open questions are raised in this report. Most provocative, perhaps, is the question of whether the notions of calculous and generic dominance are actually co-extensive in the context of relational schemata as defined here, or at least in the context of nonkeyed relational schemata. The analogous questions for internal and absolute dominance are also of interest. Another question is whether Theorem 4.6, which states that generic and internal dominance are equivalent to generic and internal embedability (respectively), can be extended to include calculus dominance. Also, can Theorem 6.2 be extended to include keyed specifiers? Finally, simple characterizations of calculous, generic and internal dominance remain unknown. (A notable exception here is that Theorem 4.6(a), together with results of [34], provides a characterization of generic dominance in terms of group theoretic concepts.)

Appendix A. Conclusion of proof of Proposition 4.5. This appendix is devoted to a proof of part of Proposition 4.5, specifically that:

PROPOSITION A.1. $A: AA \not\leqq AA$(gen).

The result is proved by contradiction, so let us begin by supposing that $A: AA \leqq AA$(gen). By Theorem 4.6 this implies that $A: AA$ is generically embedable in AA, i.e., that there is some finite $Z \subseteq \mathrm{Dom}(A)$ and a 1-1 Z-generic transformation $\sigma: A: AA \to AA$. The contradiction is obtained through a series of lemmas which focus on a specific element of $\mathbf{I}(A: AA)$. In particular, we have the following.

Notation. Let a, b, c, d be four distinct elements of $\mathrm{Dom}(A) - Z$. Also, let $I = \{(a, b, d), (b, a, c), (c, d, b), (d, c, a)\}$.

As an intuitive aid, I may be thought of as a directed graph whose edges are labelled by a "1" or "2", as shown in Fig. A1.

In the course of the proof, we shall demonstrate that $\sigma(I)$ has certain properties, and ultimately find an instance $I' \neq I$ such that $\sigma(I') = \sigma(I)$. As such, the proof is

[21] By $\cup$ we mean coordinate-wise union. Speaking intuitively, we also view $I \cup I'$ as undefined if any of the coordinates violate one of the key dependencies built into the corresponding specifier.

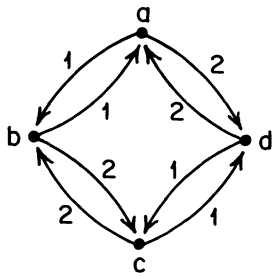


FIG. A1. *Representation of I as a labelled graph.*

reminiscent of, but more involved than, the proof that $AB \not\leq [AA]^n, [BB]^n$(gen) presented in § 4. The first step in that direction is to prove a general lemma which will be used repeatedly to infer properties of $\sigma(I)$.

LEMMA A.2. *Let μ be a Z-permutation such that $\mu(I) = I$. Then*

(a) *if $(x, y) \in \sigma(I)$ then $\mu(x, y) = (\mu(x), \mu(y)) \in I$; and*

(b) *if $x \in \text{Sym}(\sigma(I))$ then $\mu(x) \in \text{Sym}(\sigma(I))$.*

Proof. To see (a), suppose that $(x, y) \in \sigma(I)$. Then $\mu(x, y) \in \mu \circ \sigma(I) = \sigma \circ \mu(I) = \sigma(I)$ as desired. Part (b) is proved similarly. □

Next we have the following lemma which follows directly from Lemma 3.3.

LEMMA A.3. $\text{Sym}(\sigma(I)) \subseteq Z \cup \{a, b, c, d\}$.

We next define three specific Z-permutations.

Notation. Let α, β, and γ be the Z-permutations defined to be the identity on all elements of $\text{Dom}(A) - \{a, b, c, d\}$, and to map $\{a, b, c, d\}$ as shown in the following tables:

$e \mid \alpha(e)$	$e \mid \beta(e)$	$e \mid \gamma(e)$
$a \mid b$	$a \mid d$	$a \mid c$
$b \mid a$	$b \mid c$	$b \mid d$
$c \mid d$	$c \mid b$	$c \mid a$
$d \mid c$	$d \mid a$	$d \mid b$

Note that $\gamma = \beta \circ \alpha = \alpha \circ \beta$. It is easily verified (proof omitted) that:

LEMMA A.4. *$\alpha(I) = I$, $\beta(I) = I$, and $\gamma(I) = I$.* □

Speaking intuitively, the next five lemmas demonstrate that $\sigma(I)$ is very "uniform". This follows from the various symmetries occurring in I, as formally expressed by Lemma A.4 above.

LEMMA A.5. *Let $z \in Z$.*

(a) *If any of (z, a), (z, b), (z, c), or (z, d) are in $\sigma(I)$, then they all are; and*

(b) *if any of (a, z), (b, z), (c, z), or (d, z) are in $\sigma(I)$, then they all are.*

Proof. Suppose that $(z, a) \in \sigma(I)$. Because $(z, b) = \alpha(z, a)$, Lemmas A.2 and A.4 imply that $(z, b) \in \sigma(I)$. Similarly, $(z, c) = \gamma(z, a) \in \sigma(I)$ and $(z, d) = \beta(z, a) \in \sigma(I)$. Thus, each of (z, a), (z, b), (z, c), and (z, d) are in $\sigma(I)$. Similar arguments can be used to obtain the same result if it is assumed initially that one of (z, b), (z, c), or (z, d) is in $\sigma(I)$. Finally, the proof of part (b) is analogous. □

The proof of the next lemma is similar to the one just presented, and is omitted.

LEMMA A.6. *If one of (a, a), (b, b), (c, c), or (d, d) is in $\sigma(I)$, then they all are.*

LEMMA A.7.

(a) *If one of (a, b), (b, a), (c, d), or (d, c) is in $\sigma(I)$, then they all are;*

(b) *if one of (a, d), (d, a), (b, c), or (c, b) is in $\sigma(I)$, then they all are; and*

(c) *if one of (a, c), (c, a), (b, d), or (d, b) is in $\sigma(I)$, then they all are.*

Proof. Suppose that $(a, b) \in \sigma(I)$. Then $(b, a) = \alpha(a, b) \in \sigma(I)$ by Lemmas A.2 and A.4. Similarly, $(c, d) = \gamma(a, b)$ and $(d, c) = \beta(a, b)$ are in $\sigma(I)$. The other parts of case (a) are demonstrated analogously, as are cases (b) and (c). □

We now categorize the possible distinguishing properties of $\sigma(I)$ as follows.

Notation. The following six conditions are specified:

(1a) $(a, b) \in \sigma(I)$;	(1b) $(a, b) \notin \sigma(I)$;
(2a) $(a, d) \in \sigma(I)$;	(2b) $(a, d) \notin \sigma(I)$;
(3a) $(a, c) \in \sigma(I)$;	(3b) $(a, c) \notin \sigma(I)$;

We now have:

Conclusion of proof of Proposition A.1. Note that for each k, $1 \leqq k \leqq 3$, conditions (ka) and (kb) are exhaustive and mutually exclusive. As a result, the three pairs of conditions together partition the space of possibilities for $\sigma(I)$ into 8 exhaustive and mutually exclusive sets (namely (1a) and (2a) and (3a), (1a) and (2a) and (3b), etc.). We now demonstrate that in each of these 8 cases, there is an instance $I' \neq I$ such that $\sigma(I') = \sigma(I)$. This will provide the desired contradiction. We consider the eight cases in groups of two.

Case 1. ((1a) and (2a) and (3a); or (1b) and (2b) and (3b)). Let δ be a Z-permutation defined so that $\delta(b) = d$, $\delta(d) = b$, and δ is the identity on all other elements of Dom (A). Letting $I' = \delta(I)$, it is clear that $I' \neq I$. On the other hand, $\sigma(I') = \sigma \circ \delta(I) = \delta \circ \sigma(I)$. From Lemmas A.5, A.6 and A.7 it is easily seen that $\delta \circ \sigma(I) = \sigma(I)$, the desired contradiction.

Case 2. ((1a) and (2a) and (3b); or (1b) and (2b) and (3a)). In this case, the Z-permutation δ defined for Case 1 can again be used to derive the contradiction.

Case 3. ((1a) and (2b) and (3a); or (1b) and (2a) and (3b)). In this case, define δ so that $\delta(b) = c$, $\delta(c) = b$, and δ is the identity on all other elements of Dom (A). Letting $I' = \delta(I)$, it is now easily verified that $I' \neq I$, but $\sigma(I') = \sigma(I)$, yielding the desired contradiction.

Case 4. ((1b) and (2a) and (3a); or (1a) and (2b) and (3b)). In this case, define δ so that $\delta(c) = d$, $\delta(d) = c$, and δ is the identity on all other elements of Dom (A).

This completes the proof. □

Appendix B. Proof of Theorem 4.6(b). The purpose of this appendix is to prove part (b) of Theorem 4.6, which states that $\mathbf{P} \leqq \mathbf{Q}$(int) iff $\mathbf{P}$ is internally embeddable in $\mathbf{Q}$. The proof is nontrivial, and is of interest because it uses a very small number of assumptions about our data model, and thus appears to extend to most database models (see Remark B.6). To prove the result, it clearly suffices to prove that:

THEOREM B.1. *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be schemata, and suppose that* $\sigma: \mathbf{P} \to \mathbf{Q}$ *is a* 1-1, *Z-internal transformation for some finite* $Z \subseteqq \mathbf{DOM}$. *Then* $\mathbf{P} \leqq \mathbf{Q}$(int).

We begin with some motivating remarks. Suppose that $\sigma: \mathbf{P} \to \mathbf{Q}$ is 1-1 and Z-internal (for some finite $Z \subseteqq \mathbf{DOM}$). A naive attempt at proving that $\mathbf{P} \leqq \mathbf{Q}$(int) would be to define $\tau: \mathbf{Q} \to \mathbf{P}$ to be σ^{-1} (and to map all elements of $\mathbf{I}(\mathbf{Q}) - \sigma[\mathbf{I}(\mathbf{P})]$ to $\varnothing$). Recall from the proof of Theorem 4.4 that if σ and σ^{-1} are Z-internal and $\mathbf{P} \leqq \mathbf{Q}$ via (σ, σ^{-1}), then $\sigma[\mathbf{P}(YZ, Y)] \subseteqq \mathbf{Q}(YZ, Y)$ for each finite $Y \subseteqq \mathbf{DOM} - Z$. Since these inclusions are not necessarily true for an arbitrary 1-1 Z-internal $\sigma: \mathbf{P} \to \mathbf{Q}$, this naive attempt is not guaranteed to work. The underlying theme of the proof of Theorem B.1 presented here is to use σ to demonstrate for each finite $Y \subseteqq \mathbf{DOM} - Z$ that[22] $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$, and then use Theorem 4.4 to conclude that $\mathbf{P} \leqq \mathbf{Q}$(int).

[22] As in the statement and proof of Theorem 4.4, for this Appendix if M and N are subsets of **DOM** we use MN to denote $M \cup N$, etc. Also, if $b \in \mathbf{DOM}$ we write Mb to denote $M \cup \{b\}$, etc.

Before beginning the proof of Theorem B.1, we make a few observations concerning sets of the form $\mathbf{R}(V, W)$, where $\mathbf{R}$ is an arbitrary schema and V, W are finite. First (as noted before), $\mathbf{R}(V, \varnothing) = \mathbf{I}_V(\mathbf{R})$. Second, we have:

LEMMA B.2. *Let* $\mathbf{R}$ *be a schema, and let* V_1, V_2, W_1, W_2 *be finite subsets of* **DOM**. *Then* $\mathbf{R}(V_1, W_1) \cap \mathbf{R}(V_2, W_2) = \mathbf{R}(V_1 \cap V_2, W_1 \cup W_2)$.

Proof. If $I \in \mathbf{R}(V_1, W_1) \cap \mathbf{R}(V_2, W_2)$, then $W_i \subseteq \mathrm{Sym}(I) \subseteq V_i$ for $i = 1, 2$; that is, $W_1 \cup W_2 \subseteq \mathrm{Sym}(I) \subseteq V_1 \cap V_2$. Thus, $I \in \mathbf{R}(V_1 \cap V_2, W_1 \cup W_2)$. The converse is equally straightforward. □

(As an aside, note in the above lemma that if $W_1 \cup W_2 \not\subseteq V_1 \cap V_2$, then $\mathbf{R}(V_1, W_1) \cap \mathbf{R}(V_2, W_2) = \varnothing = \mathbf{R}(V_1 \cap V_2, W_1 \cup W_2)$.)

Our final observation about the sets $\mathbf{R}(V, W)$ is very general, and formally expresses a central aspect of our model of relational instances. Speaking roughly, the result states that the cardinality of a set of the form $\mathbf{R}(V, W)$ is essentially independent of the sets V and W, and depends only on the numbers $|V \cap \mathrm{Dom}(B)|$, $|W \cap \mathrm{Dom}(B)|$, and $|V \cap W \cap \mathrm{Dom}(B)|$ for each $B \in \mathrm{supp}(\mathbf{R})$. The straightforward proof of the result is omitted.

LEMMA B.3. *Let* $\mathbf{R}$ *be a schema, let* $V, W \subseteq$ **DOM** *be finite, and let* π *be a* $\varnothing$*-permutation of* **DOM**. *Then* $|\mathbf{R}(V, W)| = |\mathbf{R}(\pi(V), \pi(W))|$. □

We now begin the formal proof of Theorem B.1 by establishing the following.

Notation. Let $\mathbf{P}$ and $\mathbf{Q}$ be fixed schemata; let $Z \subseteq$ **DOM** be finite; and let $\sigma: \mathbf{P} \to \mathbf{Q}$ be 1-1 and Z-internal.

As mentioned above, the proof of the result consists in showing, for all finite $Y \subseteq$ **DOM** $- Z$, that $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$. By Lemma B.3, it suffices for each such Y to find a set X with the same "shape" as Y (i.e., such that $X \cap Z = \varnothing$ and $\pi(X) = Y$ for some Z-permutation π) with the property that $|\mathbf{P}(XZ, X)| \leqq |\mathbf{Q}(XZ, X)|$. In fact, an X will be exhibited such that $\sigma[\mathbf{P}(XZ, X)] \subseteq \mathbf{Q}(XZ, X)$; since σ is 1-1 this will yield the desired inequality. This is accomplished by Lemma B.5 below. As a preliminary step, Lemma B.4 essentially demonstrates this for sets Y satisfying $Y \subseteq \mathrm{Dom}(B) - Z$ for some basic type B.

The argument used for Lemma B.4 relies in part on "Ramsey's Theorem" for finite sets. We state here the relevant version of that theorem for the reader's convenience.

RAMSEY'S THEOREM [35], [18]. *Let* $0 < n \leqq k_1, k_2$ *be integers. Then there is a number* $r(n, k_1, k_2)$ *with the following property: For each set* U *such that* $|U| \geqq r(n, k_1, k_2)$ *and each function*[23] $f: [\![U]\!]^n \to \{1, 2\}$, *there is some* $i \in \{1, 2\}$ *and set* $V \subseteq U$ *with* $|V| = k_i$ *and* $f(V') = i$ *for each* $V' \in [\![V]\!]^n$.

We also use:

Notation. For $y \geqq 0$, let $q(y)$ denote $\max\{|\mathbf{I}_Y(\mathbf{Q})| \,|\, |Y| \leqq y\}$.

We now have:

LEMMA B.4. *For each* $n \geqq 0$ *there is a function* $\lambda_n: \mathbf{N} \to \mathbf{N}$ *with the following property: If* Y *is such that*

(i) $Y \subseteq$ **DOM** $- Z$, *and*

(ii) $|Y| \leqq y$,

and $B \in \mathrm{supp}(\mathbf{PQ})$ *and* $\bar{X}$ *are such that*

(iii) $\bar{X} \subseteq \mathrm{Dom}(B) - YZ$, *and*

(iv) $|\bar{X}| \geqq \lambda_n(y)$;

then there is some $X \subseteq \bar{X}$ *such that* $|X| = n$ *and* $\sigma[\mathbf{P}(XYZ, X)] \subseteq \mathbf{Q}(XYZ, X)$.

[23] For a set S and $m > 0$, $[\![S]\!]^m$ denotes the set $\{S' | S' \subseteq S \text{ and } |S'| = m\}$.

Proof. (Before beginning, we briefly mention the special case where the basic type B is in supp $(\mathbf{Q})$ − supp $(\mathbf{P})$. In this case, for each nonempty $X \subseteq \mathrm{Dom}(B)$ and each $Y \subseteq \mathbf{DOM}$, $\mathbf{P}(XYZ, X) = \emptyset$, and so $\sigma[\mathbf{P}(XYZ, X)] \subseteq \mathbf{Q}(XYZ, X)$ holds trivially.)

Now turning to the formal proof, we first note that λ_0 can be defined so that $\lambda_0(y) = 0$ for each y. This choice satisfies the conditions of the lemma because σ is Z-internal.

For the remaining cases, we perform an induction on n. To begin, define λ_1 so that $\lambda_1(y) = q(y+z)+1$. We now argue that this choice of λ_1 satisfies the statement of the lemma. Let $y \geqq 0$, and let Y, B and $\bar{X}$ satisfy conditions (i) through (iv). Suppose that the conclusion of the lemma is not satisfied. Then for each $b \in \bar{X}$ we have $\sigma[\mathbf{P}(YZb, b)] \not\subseteq \mathbf{Q}(YZb, b)$. Because $\sigma[\mathbf{P}(YZb, b)] \subseteq \mathbf{I}_{YZb}(\mathbf{Q}) = \mathbf{Q}(YZb, b) \cup \mathbf{Q}(YZ, \emptyset)$, this implies that $\sigma[\mathbf{P}(YZb, b)] \cap \mathbf{Q}(YZ, \emptyset) \neq \emptyset$ for each $b \in \bar{X}$. Also, for $b \neq b'$ we have $\mathbf{P}(YZb, b) \cap \mathbf{P}(YZb', b') = \emptyset$. For each $b \in \bar{X}$, let $I_b \in \mathbf{P}(YZb, b)$ be such that $\sigma(I_b) \in \mathbf{Q}(YZ, \emptyset)$. Note that $I_b \neq I_{b'}$ and $\sigma(I_b) \neq \sigma(I_{b'})$ for $b \neq b'$. From this we have $q(y+z)+1 = |\bar{X}| = |\{I_b | b \in \bar{X}\}| = |\{\sigma(I_b) | b \in \bar{X}\}| \leqq |\mathbf{Q}(YZ, \emptyset)| \leqq q(y+z)$, a contradiction. Therefore, there is some $b \in \bar{X}$ such that $\sigma[\mathbf{P}(YZb, b)] \subseteq \mathbf{Q}(YZb, b)$, and so the case for $n = 1$ is demonstrated. Note also that $\lambda_1(y) \geqq 1$ for each $y \geqq 0$.

Suppose now that the lemma is true for some $n > 0$ and that $\lambda_n(y) \geqq n$ for each $y \geqq 0$. The definition of $\lambda_{n+1}(y)$ involves an induction of length $q(n+y+z)+1$. Specifically, let $\psi_{q(n+y+z)+1} = n$, and for each i, $0 \leqq i \leqq q(n+y+z)$, let

$$\psi_i = r(n, \lambda_n(y+1), \psi_{i+1}) + 1.$$

(Note that $\psi_i \geqq n$ for each i, $0 \leqq i \leqq q(n+y+z)+1$.) Finally, set $\lambda_{n+1}(y) = \psi_0$. Clearly $\lambda_{n+1}(y) \geqq n+1$ for each $y \geqq 0$.

To complete the proof, it must be shown that λ_{n+1} has the property in the statement of the lemma. To this end, let $y \geqq 0$, and let Y and $\bar{X}$ satisfy conditions (i) through (iv). In particular, then, $|\bar{X}| \geqq \lambda_{n+1}(y) = \psi_0$. We now prove the following.

CLAIM. *There are some $b \in \bar{X}$ and $V \subseteq \bar{X} - \{b\}$ such that*

α. $|V| = \lambda_n(y+1)$, *and*

β. $\sigma[\mathbf{P}(XYZb, b)] \subseteq \mathbf{Q}(XYZb, b)$ *for each* $X \in [\![V]\!]^n$.

To prove the claim, suppose that it is false. A contradiction will be derived by inductively constructing a sequence $b_1, b_2, \cdots, b_{q(n+y+z)+1}$ of elements of $\bar{X}$ and a set $W_{q(n+y+z)+1}$ with certain properties.

To begin the induction, let $b_1 \in \bar{X}$ be fixed, and define $f_1 : [\![\bar{X} - \{b_1\}]\!]^n \to \{1, 2\}$ so that for each $X \in [\![\bar{X} - \{b_1\}]\!]^n$,

$$f_1(X) = \begin{cases} 1 & \text{if } \sigma[\mathbf{P}(XYZb_1, b_1)] \subseteq \mathbf{Q}(XYZb_1, b_1), \\ 2 & \text{if } \sigma[\mathbf{P}(XYZb_1, b_1)] \not\subseteq \mathbf{Q}(XYZb_1, b_1). \end{cases}$$

By assumption, there is no set $V \subseteq \bar{X} - \{b_1\}$ such that $|V| = \lambda_n(y+1)$ and $f_1(X) = 1$ for each $X \in [\![V]\!]^n$. Since

$$|\bar{X} - \{b_1\}| = \psi_0 - 1 = r(n, \lambda_n(y+1), \psi_1),$$

Ramsey's Theorem implies that there is a set $W_1 \subseteq \bar{X}$ such that $|W_1| = \psi_1$ and $f_1(X) = 2$ for each $X \in [\![W_1]\!]^n$. In particular, then, $\sigma[\mathbf{P}(XYZb_1, b_1)] \cap \mathbf{Q}(XYZ, \emptyset) \neq \emptyset$ for each $X \in [\![W_1]\!]^n$.

Suppose inductively that for some i, $1 \leqq i \leqq q(n+y+z)$, we have chosen $b_1, \cdots, b_i$ and $W_1, \cdots, W_i$ such that

(a) $|W_i| = \psi_i$;

(b) $W_i \subseteq \bar{X}$;

(c) $b_j \in \bar{X} - W_i$ for each j, $1 \leqq j \leqq i$; and

(d) $\sigma[\mathbf{P}(XYZb_j, b_j)] \cap \mathbf{Q}(XYZ, \varnothing) \neq \varnothing$ for each $X \in [\![W_i]\!]^n$ and each j, $1 \leqq j \leqq i$.

Let $b_{i+1} \in W_i$ be fixed, and define $f_{i+1} : [\![W_i - \{b_{i+1}\}]\!]^n \to \{1, 2\}$ so that for each $X \in [\![W_i - \{b_{i+1}\}]\!]^n$,

$$f_{i+1}(X) = \begin{cases} 1 & \text{if } \sigma[\mathbf{P}(XYZb_{i+1}, b_{i+1})] \subseteqq \mathbf{Q}(XYZb_{i+1}, b_{i+1}), \\ 2 & \text{if } \sigma[\mathbf{P}(XYZb_{i+1}, b_{i+1})] \not\subseteqq \mathbf{Q}(XYZb_{i+1}, b_{i+1}). \end{cases}$$

By assumption, there is no set $V \subseteqq \bar{X} - \{b_{i+1}\}$ such that $|V| = \lambda_n(y+1)$ and $f_{i+1}(X) = 1$ for each $X \in [\![V]\!]^n$. Ramsey's Theorem implies that there is some $W_{i+1} \subseteqq W_i$ such that $|W_{i+1}| = \psi_{i+1}$ and $f_{i+1}(X) = 2$ for each $X \in [\![W_{i+1}]\!]^n$. (Thus (a) and (b) hold for $i+1$.) In particular, $\sigma[\mathbf{P}(XYZb_{i+1}, b_{i+1})] \cap \mathbf{Q}(XYZ, \varnothing) \neq \varnothing$ for each $X \in [\![W_{i+1}]\!]^n$. Also, because conditions (c) and (d) hold for i and $W_{i+1} \subseteqq W_i$, we have $b_j \in \bar{X} - W_{i+1}$ and $\sigma[\mathbf{P}(XYZb_j, b_j)] \cap \mathbf{Q}(XYZ, \varnothing) \neq \varnothing$ for each $X \in [\![W_{i+1}]\!]^n$ and each j, $1 \leqq j \leqq i$. (Thus (c) and (d) hold for $i+1$.) With this the induction has been extended.

Let $t = q(n+y+z)+1$. By the above induction, there is a sequence $b_1, \cdots, b_t$, and a set W_t such that

(a′) $|W_t| = \psi_t = n$;

(b′) $W_t \subseteqq \bar{X}$;

(c′) $b_j \in \bar{X} - W_t$ for each j, $1 \leqq j \leqq t$; and

(d′) $\sigma[\mathbf{P}(XYZb_j, b_j)] \cap \mathbf{Q}(XYZ, \varnothing) \neq \varnothing$ for each $X \in [\![W_t]\!]^n$ and each j, $1 \leqq j \leqq t$.

In particular, (a′) and (d′) imply that

(d″) $\sigma[\mathbf{P}(W_tYZb_j, b_j)] \cap \mathbf{Q}(XYZ, \varnothing) \neq \varnothing$ for each j, $1 \leqq j \leqq t$.

Note that for $j \neq k$, $\mathbf{P}(W_tYZb_j, b_j) \cap \mathbf{P}(W_tYZb_k, b_k) = \varnothing$. For each j, $1 \leqq j \leqq t$, let $I_j \in \mathbf{P}(W_tYZb_j, b_j)$ be chosen so that $\sigma(I_j) \in \mathbf{Q}(W_tYZ, \varnothing)$. Note that if $j \neq k$ then $I_j \neq I_k$; and because σ is 1-1, $\sigma(I_j) \neq \sigma(I_k)$. Then

$$\begin{aligned} q(n+y+z)+1 &= t \\ &= |\{I_j | 1 \leqq j \leqq t\}| \\ &= |\{\sigma(I_j) | 1 \leqq j \leqq t\}| \\ &\leqq |\mathbf{Q}(W_tYZ, \varnothing)| \\ &\leqq q(n+y+z), \end{aligned}$$

a contradiction. This demonstrates the claim.

To complete the proof of the lemma, recall that we have fixed $y \geqq 0$ and sets Y and $\bar{X}$ which satisfy conditions (i) through (iv) of the statement of the lemma. By the claim, there is some $b \in \bar{X}$ and $V \subseteqq \bar{X} - \{b\}$ such that $|V| = \lambda_n(y+1)$ and $\sigma[\mathbf{P}(XYZb, b)] \subseteqq \mathbf{Q}(XYZb, b)$ for each $X \in [\![V]\!]^n$.

Because $|V| = \lambda_n(y+1)$ and $|Yb| = y+1$, the inductive assumption on n implies that there is a set $X \subseteqq V$ such that $|X| = n$ and $\sigma[\mathbf{P}(XYZb, X)] \subseteqq \mathbf{Q}(XYZb, X)$. For this X, we also have $\sigma[\mathbf{P}(XYZb, b)] \subseteqq \mathbf{Q}(XYZb, b)$. By Lemma B.2, $\mathbf{P}(XYZb, X) \cap \mathbf{P}(XYZb, b) = \mathbf{P}(XYZb, Xb)$, and similarly for $\mathbf{Q}$. Because σ is 1-1, $\sigma[\mathbf{P}(XYZb, X)] \cap \sigma[\mathbf{P}(XYZb, b)] = \sigma[\mathbf{P}(XYZb, Xb)]$. Therefore, $\sigma[\mathbf{P}(XYZb, Xb)] \subseteqq \mathbf{Q}[XYZb, Xb]$, and so the conclusion of the statement of the lemma is satisfied by using the set Xb. □

We can now prove the key lemma of the proof of Theorem B.1. For this result, we use:

Notation. Let $B_1, \cdots, B_s$ be an enumeration of the elements of supp $(\mathbf{PQ})$. Also, for each $X \subseteqq \bigcup_{1 \leqq i \leqq s} \text{Dom}(B_i)$, the *characteristic vector* of X, denoted char (X), is the vector $(|X \cap \text{Dom}(B_1)|, \cdots, |X \cap \text{Dom}(B_s)|)$.

Note that if X and Y are finite subsets of $\bigcup_{1 \leqq i \leqq s} \mathrm{Dom}(B_i)$ then $\mathrm{char}(X) = \mathrm{char}(Y)$ iff $Y = \pi(X)$ for some $\varnothing$-permutation π (or speaking intuitively, iff X and Y have the same "shape").

We now have the following lemma.

LEMMA B.5. *Let* $\mathbf{n} = (n_1, \cdots, n_s)$ *be a sequence of nonnegative integers. Then there is some* $X \subseteq \bigcup_{1 \leqq i \leqq s} \mathrm{Dom}(B_i)$ *such that*

(i) $X \subseteq \bigcup_{1 \leqq i \leqq s} \mathrm{Dom}(B_i) - Z$;

(ii) $\mathrm{char}(X) = \mathbf{n}$; *and*

(iii) $\sigma[\mathbf{P}(XZ, X)] \subseteq \mathbf{Q}(XZ, X)$.

Proof. To demonstrate this result, Lemma B.4 will be used repeatedly, once for each basic type in supp (**PQ**). We begin the argument by choosing a very large set Y, and then iteratively select subsets $W_i \subseteq Y \cap \mathrm{Dom}(B_i)$ such that $|W_i| = n_i$; ultimately the set $X = \bigcup_{1 \leqq i \leqq s} W_i$ will satisfy the conditions of the lemma.

To begin the proof we inductively define a sequence $(\kappa_1, \cdots, \kappa_s)$. Let $\kappa_s = \lambda_{n_s}(\sum_{1 \leqq j < s} n_j)$. Given κ_{i+1} for some i, $1 \leqq i < s$, let

$$\kappa_i = \lambda_{n_i}\left(\sum_{1 \leqq j < i} n_i + \sum_{i < j \leqq s} \kappa_j\right).$$

Finally, let $Y \subseteq \bigcup_{1 \leqq i \leqq s} \mathrm{Dom}(B_i) - Z$ be chosen so that $\mathrm{char}(Y) = (\kappa_1, \cdots, \kappa_s)$.

For each i, $1 \leqq i \leqq s$, let Y_i denote $\bigcup_{i < j \leqq s} (Y \cap \mathrm{Dom}(B_j))$. To choose the sets $W_1, \cdots, W_s$ we shall perform a moderately involved inductive construction. At the ith step, we will choose the set $W_i \subseteq Y \cap \mathrm{dom}(B_i)$ such that $|W_i| = n_i$, and shall use X_i to denote $\bigcup_{1 \leqq j \leqq i} W_i$. At the conclusion of the ith step, we will have shown that $\sigma[\mathbf{P}(X_i Y_i Z, X_i)] \subseteq \mathbf{Q}(X_i Y_i Z, X_i)$. Note that the successful completion of this induction will demonstrate the lemma, because $Y_s = \varnothing$ and $\mathrm{char}(X_s) = \mathbf{n}$.

To begin the induction, note that $Y_0 = Y$, and let $X_0 = \varnothing$. Then $\sigma[\mathbf{P}(X_0 Y_0 Z, X_0)] = \sigma[\mathbf{P}(YZ, \varnothing)] \subseteq \mathbf{Q}(YZ, \varnothing) = \mathbf{Q}(X_0 Y_0 Z, X_0)$ because σ is Z-internal.

Suppose now for some i, $0 \leqq i < s$, that sets $W_1, \cdots, W_i$ have been chosen such that $W_j \subseteq Y \cap \mathrm{Dom}(B_j)$ and $|W_j| = n_j$ for $1 \leqq j \leqq i$; and also such that, letting $X_i = \bigcup_{1 \leqq j \leqq i} W_j$, we have

(α) $\sigma[\mathbf{P}(X_i Y_i Z, X_i)] \subseteq \mathbf{Q}(X_i Y_i Z, X_i)$.

Note that $|X_i Y_{i+1}| = \sum_{1 \leqq j < i+1} n_i + \sum_{i+1 < j \leqq s} \kappa_j$ and that

$$\begin{aligned} |Y \cap \mathrm{Dom}(B_{i+1})| &= \kappa_{i+1} \\ &= \lambda_{n_{i+1}}\left(\sum_{1 \leqq j < i+1} n_i + \sum_{i+1 < j \leqq s} \kappa_j\right) \\ &= \lambda_{n_{i+1}}(|X_i Y_{i+1}|). \end{aligned}$$

Therefore, by Lemma B.4, there is a set $W_{i+1} \subseteq Y \cap \mathrm{Dom}(B_{i+1})$ such that $|W_{i+1}| = n_{i+1}$ and

(β) $\sigma[\mathbf{P}(X_i W_{i+1} Y_{i+1} Z, W_{i+1})] \subseteq \mathbf{Q}(X_i W_{i+1} Y_{i+1} Z, W_{i+1})$.

Let $X_{i+1} = X_i W_{i+1} = \bigcup_{1 \leqq j \leqq i+1} W_j$. By Lemma B.2, $\mathbf{P}(X_i Y_i Z, X_i) \cap \mathbf{P}(X_i W_{i+1} Y_{i+1} Z, W_{i+1}) = \mathbf{P}(X_i W_{i+1} Y_{i+1} Z, X_i W_{i+1}) = \mathbf{P}(X_{i+1} Y_{i+1} Z, X_{i+1})$; and the analogous equalities hold for **Q**. Since σ is 1-1, we also have $\sigma[\mathbf{P}(X_i Y_i Z, X_i)] \cap \sigma[\mathbf{P}(X_i W_{i+1} Y_{i+1} Z, W_{i+1})] = \sigma[\mathbf{P}(X_{i+1} Y_{i+1} Z, X_{i+1})]$. Now ($\alpha$) and ($\beta$) imply that $\sigma[\mathbf{P}(X_{i+1} Y_{i+1} Z, X_{i+1})] \subseteq \mathbf{Q}(X_{i+1} Y_{i+1} Z, X_{i+1})$, and the induction is extended.

As noted above, the set X_s obtained from the completed induction has properties (i), (ii) and (iii) in the statement of the lemma, and so the proof is complete. □

We now have:

Conclusion of proof of Theorem B.1. We have assumed that $\sigma: \mathbf{P} \to \mathbf{Q}$ is 1-1 and Z-internal for some finite $Z \subseteq \mathbf{DOM}$. To show that $\mathbf{P} \leqq \mathbf{Q}$(int) we shall apply Theorem 4.4. To this end, let $Y \subseteq \mathbf{DOM} - Z$ be finite. By Lemma B.5 there is some $X \subseteq \mathbf{DOM} - Z$ such that $\text{char}(X) = \text{char}(Y)$ and $\sigma[\mathbf{P}(XZ, X)] \subseteq \mathbf{Q}(XZ, X)$. Because σ is 1-1, this implies that $|\mathbf{P}(XZ, X)| \leqq |\mathbf{Q}(XZ, X)|$. By Lemma B.3, $|\mathbf{P}(YZ, Y)| \leqq |\mathbf{Q}(YZ, Y)|$. Because Y was arbitrary, Theorem 4.4 now implies that $\mathbf{P} \leqq \mathbf{Q}$(int) as desired. □

Remark B.6. The technique of the proof Theorem B.1 used here can also be applied to other database models in which the families of instances associated with schemata are defined using basic types in a manner analogous to the way they were defined here. In particular, two specific characteristics of the definition of families of instances used in the proof can be identified. Specifically, it was assumed throughout that if Y is a finite subset of **DOM** then for each schema $\mathbf{P}$, $\mathbf{I}_Y(\mathbf{P})$ is also finite. Second, the result Lemma B.3 was essential to the proof. It is easily verified that these were the only two characteristics of the definition of family of instances which were used in proving Theorem B.1. Thus, the result extends immediately to the Format Model of [20] and the IFO Model of [1a] (assuming that the definition of family of instances used there is modified in analogy to the definition used here).

Appendix C. Proof of Theorem 4.7. This appendix is devoted to the proof of Theorem 4.7, which states that (each type of) dominance is preserved by renaming of the basic types, even if different basic types are identified by the renaming. The result is restated here for the convenience of the reader.

THEOREM C.1. *Let* $\mathbf{P}$ *and* $\mathbf{Q}$ *be schemata, and* h *a homomorphism on* $\mathscr{B}$. *If* $\mathbf{P} \leqq \mathbf{Q}$(xxx) *then* $h(\mathbf{P}) \leqq h(\mathbf{Q})$(xxx), *where* "xxx" *ranges over* "calc", "gen", "int", *and* "abs".

Proof. Let $B_1, \cdots, B_n$ be an enumeration of the basic types of supp $(\mathbf{PQ})$, and let $A_1, \cdots, A_m$ be an enumeration of supp $(h(\mathbf{PQ}))$.

Now, suppose that $\mathbf{P} \leqq \mathbf{Q}$(abs). By Theorem 4.2 there is some $t \geqq 0$ such that for each $\mathbf{x}$ with $x_j \geqq t$, $1 \leqq j \leqq n$, $f_{\mathbf{P}}(\mathbf{x}) \leqq f_{\mathbf{Q}}(\mathbf{x})$. Suppose now that $h(B_j) = A_{i_j}$ for $1 \leqq j \leqq n$. Then $f_{h(\mathbf{P})}(x_1, \cdots, x_m) = f_{\mathbf{P}}(x_{i_1}, \cdots, x_{i_n})$ and $f_{h(\mathbf{Q})}(x_1, \cdots, x_m) = f_{\mathbf{Q}}(x_{i_1}, \cdots, x_{i_n})$. The result now follows for absolute dominance.

For the other types of dominance we define a family of mappings associated with the homomorphism h. First, for each basic type B, define

$$\mu_B: \text{Dom}(B) \to \text{Dom}(h(B))$$

such that it is 1-1 and onto. Note that μ_B is invertible. (If $h(B) = B$ we do not require that μ_B is the identity.) (We use here the assumption that all domains are countably infinite. If finite domains were permitted, then the definition of homomorphism would have to be modified so that for each $B \in \mathscr{B}$, $|\text{Dom}(h(B))| = |\text{Dom}(B)|$.) Also, define

$$\mu_{\mathbf{DOM}}: \mathbf{DOM} \to \mathbf{DOM}$$

such that $\mu_{\mathbf{DOM}} = \cup \{\mu_B | B \in \mathscr{B}\}$. Note that $\mu_{\mathbf{DOM}}$ need not be 1-1 or onto.

If $R = B_1, \cdots, B_k$ is a nonkeyed relational specifier then define[24]

$$\mu_R: \mathbf{I}(R) \to \mathbf{I}(h(R))$$

[24] For this discussion, we assume that if R is ordered $B_1, \cdots, B_k$ then $h(R)$ is ordered $h(B_1), \cdots, h(B_k)$, etc.

such that for each $I \in \mathbf{I}(R)$,

$$\mu_R(I) = \{(\mu_{B_1}(b_1), \cdots, \mu_{B_k}(b_k)) | (b_1, \cdots, b_k) \in I\}.$$

If $R = S : T$, then μ_R is defined analogously. It is easily verified that μ_R is 1-1 and onto for each (keyed or nonkeyed) specifier R, and hence invertible.

If $\mathbf{R} = R_1, \cdots, R_k$ is a schema, define

$$\mu_{\mathbf{R}} : \mathbf{I}(\mathbf{R}) \to \mathbf{I}(h(\mathbf{R}))$$

so that for each $I = (I_1, \cdots, I_k) \in \mathbf{I}(\mathbf{R})$, $\mu_{\mathbf{R}}(I) = (\mu_{R_1}(I), \cdots, \mu_{R_k}(I))$. Again, $\mu_{\mathbf{R}}$ is 1-1 and onto, and hence invertible.

Suppose now that $\mathbf{P} \leqq \mathbf{Q}$ via (σ, τ), where $\sigma : \mathbf{P} \to \mathbf{Q}$ and $\tau : \mathbf{Q} \to \mathbf{P}$ are arbitrary transformations. Letting $\sigma' = \mu_{\mathbf{Q}} \circ \sigma \circ \mu_{\mathbf{P}}^{-1}$ and $\tau' = \mu_{\mathbf{P}} \circ \tau \circ \mu_{\mathbf{Q}}^{-1}$, it is easily verified that $h(\mathbf{P}) \leqq h(\mathbf{Q})$ via (σ', τ'). To complete the proof, then, it suffices to show that if a transformation $\sigma : \mathbf{P} \to \mathbf{Q}$ is Z-internal (for some finite Z) then $\sigma' = \mu_{\mathbf{Q}} \circ \sigma \circ \mu_{\mathbf{P}}^{-1} : h(\mathbf{P}) \to h(\mathbf{Q})$ is Z'-internal (for some finite Z'); that if σ is Z-generic (for some finite Z) then σ' is Z'-generic (for some finite Z'); and that if σ is realized by a calculus expression then so is σ'. (By symmetry, if this is true for σ' then it is also true for τ'.)

Suppose now that $\sigma : \mathbf{P} \to \mathbf{Q}$ is Z-internal, and that $\sigma' = \mu_{\mathbf{Q}} \circ \sigma \circ \mu_{\mathbf{P}}^{-1}$. Letting $Z' = \mu_{\mathbf{DOM}}[Z]$, we now show that σ' is Z'-internal. Let $I \in \mathbf{I}(h(\mathbf{P}))$. Note that $\mu_{\mathbf{DOM}}[\text{Sym}\,(\mu_{\mathbf{P}}^{-1}(I))] \subseteqq \text{Sym}\,(I)$. (For suppose that $a \in \mu_{\mathbf{DOM}}[\text{Sym}\,(\mu_{\mathbf{P}}^{-1}(I))]$. Then $a = \mu_{\mathbf{DOM}}(b) = \mu_B(b)$ for some $B \in \mathcal{B}$ and $b \in \text{Dom}\,(B)$, where $b \in \text{Sym}\,(\mu_{\mathbf{P}}^{-1}(I))$. This implies that b occurs in some B-column of $\mu_{\mathbf{P}}^{-1}(I)$. From the definition of $\mu_{\mathbf{P}}$ it follows that $a = \mu_B(b)$ occurs in some $h(B)$-column of I. Thus $a \in \text{Sym}\,(I)$ as desired.) We now have:

$$\begin{aligned}
\text{Sym}\,(\sigma'(I)) &= \text{Sym}\,(\mu_{\mathbf{Q}} \circ \sigma \circ \mu_{\mathbf{P}}^{-1}(I)) \\
&= \mu_{\mathbf{DOM}}[\text{Sym}\,(\sigma(\mu_{\mathbf{P}}^{-1}(I)))] \\
&\subseteqq \mu_{\mathbf{DOM}}[\text{Sym}\,(\mu_{\mathbf{P}}^{-1}(I)) \cup Z] \qquad \text{(since } \sigma \text{ is } Z\text{-internal)} \\
&= \mu_{\mathbf{DOM}}[\text{Sym}\,(\mu_{\mathbf{P}}^{-1}(I))] \cup Z' \\
&\subseteqq \text{Sym}\,(I) \cup Z'.
\end{aligned}$$

Thus σ' is Z'-internal as desired.

Suppose now that $\sigma : \mathbf{P} \to \mathbf{Q}$ is Z-generic, and again set $Z' = \mu_{\mathbf{DOM}}[Z]$. Let α be a Z'-permutation and $I \in \mathbf{I}(h(\mathbf{P}))$. It now suffices to show that $\alpha \circ \sigma'(I) = \sigma' \circ \alpha(I)$. To this end, for each $B \in \mathcal{B}$, define

$$\bar{\alpha}_B : \text{Dom}\,(B) \to \text{Dom}\,(B)$$

so that $\bar{\alpha}_B(b) = \mu_B^{-1} \circ \alpha \circ \mu_B(b)$. Note that $\bar{\alpha}_B$ is a 1-1, onto mapping. Also, note that if $b \in Z$ then $\mu_B(b) = \mu_{\mathbf{DOM}}(b) \in \mu_{\mathbf{DOM}}[Z] = Z'$. Thus $\bar{\alpha}_B(b) = \mu_B^{-1} \circ \alpha(\mu_B(b)) = \mu_B^{-1}(\mu_B(b)) = b$. This implies that $\bar{\alpha}_B$ is the identity on $Z \cap \text{Dom}\,(B)$. Now define

$$\bar{\alpha}_{\mathbf{DOM}} : \mathbf{DOM} \to \mathbf{DOM}$$

so that $\bar{\alpha}_{\mathbf{DOM}} = \cup \{\bar{\alpha}_B | B \in \mathcal{B}\}$. It follows that $\bar{\alpha}_{\mathbf{DOM}}$ is a Z-permutation. Also, the map $\bar{\alpha}_{\mathbf{DOM}}$ can be applied to instances as well as domain elements.

We now argue for each schema $\mathbf{R}$ and each instance $J \in \mathbf{I}(\mathbf{R})$ that $\mu_{\mathbf{R}}^{-1} \circ \alpha \circ \mu_{\mathbf{R}}(J) = \bar{\alpha}_{\mathbf{DOM}}(J)$. To see this, let $\mathbf{R} = R_1, \cdots, R_s$ and let $J = (J_1, \cdots, J_s) \in \mathbf{I}(\mathbf{R})$. Then $\bar{\alpha}_{\mathbf{DOM}}(J) = (\bar{\alpha}_{\mathbf{DOM}}(J_1), \cdots, \bar{\alpha}_{\mathbf{DOM}}(J_s))$. So it suffices to show that if R is a specifier and $J \in \mathbf{I}(R)$ then $\bar{\alpha}_{\mathbf{DOM}}(J) = \mu_R^{-1} \circ \alpha \circ \mu_R(J)$. Suppose that $R = B_1, \cdots, B_t$ (or

$B_1, \cdots, B_k : B_{k+1}, \cdots, B_t)$, let $J \in \mathbf{I}(R)$, and let $(b_1, \cdots, b_t) \in J$. Then

$$\mu_R^{-1} \circ \alpha \circ \mu_R(b_1, \cdots, b_t) = (\mu_{B_1}^{-1} \circ \alpha \circ \mu_{B_1}(b_1), \cdots, \mu_{B_t}^{-1} \circ \alpha \circ \mu_{B_t}(b_t))$$
$$= (\bar{\alpha}_{\mathbf{DOM}}(b_1), \cdots, \bar{\alpha}_{\mathbf{DOM}}(b_t)) = \bar{\alpha}_{\mathbf{DOM}}(b_1, \cdots, b_t).$$

More generally we have $\mu_R^{-1} \circ \alpha \circ \mu_R(J) = \bar{\alpha}_{\mathbf{DOM}}(J)$, and the argument is complete.

Recalling that I is an arbitrary element of $\mathbf{I}(h(\mathbf{P}))$, we are now prepared to show that $\sigma' \circ \alpha(I) = \alpha \circ \sigma'(I)$. Specifically,

$$\begin{aligned}
\sigma' \circ \alpha(I) &= (\mu_{\mathbf{Q}} \circ \sigma \circ \mu_{\mathbf{P}}^{-1}) \circ \alpha \circ (\mu_{\mathbf{P}} \circ \mu_{\mathbf{P}}^{-1})(I) \\
&= \mu_{\mathbf{Q}} \circ (\sigma \circ (\mu_{\mathbf{P}}^{-1} \circ \alpha \circ \mu_{\mathbf{P}}))(\mu_{\mathbf{P}}^{-1}(I)) \\
&= \mu_{\mathbf{Q}} \circ (\sigma \circ \bar{\alpha}_{\mathbf{DOM}})(\mu_{\mathbf{P}}^{-1}(I)) && (\text{since } \mu_{\mathbf{P}}^{-1} \circ \alpha \circ \mu_{\mathbf{P}} = \bar{\alpha}_{\mathbf{DOM}} \text{ on } \mathbf{I}(\mathbf{P})) \\
&= \mu_{\mathbf{Q}} \circ (\bar{\alpha}_{\mathbf{DOM}} \circ \sigma)(\mu_{\mathbf{P}}^{-1}(I)) && (\text{since } \bar{\alpha}_{\mathbf{DOM}} \text{ is a } Z\text{-permutation}) \\
&= \mu_{\mathbf{Q}} \circ ((\mu_{\mathbf{Q}}^{-1} \circ \alpha \circ \mu_{\mathbf{Q}}) \circ \sigma)(\mu_{\mathbf{P}}^{-1}(I)) && (\text{since } \mu_{\mathbf{Q}}^{-1} \circ \alpha \circ \mu_{\mathbf{Q}} = \bar{\alpha}_{\mathbf{DOM}} \text{ on } \mathbf{I}(\mathbf{Q})) \\
&= (\mu_{\mathbf{Q}} \circ \mu_{\mathbf{Q}}^{-1}) \circ \alpha \circ (\mu_{\mathbf{Q}} \circ \sigma \circ \mu_{\mathbf{P}}^{-1})(I) \\
&= \alpha \circ \sigma'(I).
\end{aligned}$$

With this we have shown that σ' is Z'-generic on $\mathbf{I}(h(\mathbf{P}))$ as desired.

Finally, suppose that $\xi : \mathbf{P} \to \mathbf{Q}$ is a calculus expression. Recall that in our formalism, each variable occurring in ξ is associated with some basic type B, and in particular is taken from a set V_B of variables. For each basic type $B \in \mathscr{B}$, let V'_B be the set of variables occurring in ξ associated with B. Also, let $W_B \subseteq V_{h(B)}$ be chosen for each $B \in \mathscr{B}$ such that $|W_B| = |V'_B|$ and such that $W_B \cap W_C = \varnothing$ whenever $B \neq C$ (even if $h(B) = h(C)$). Also, let $\eta : \bigcup_{B \in \mathscr{B}} V'_B \to \bigcup_{B \in \mathscr{B}} W_B$ be a 1-1 function such that $\eta[V'_B] \subseteq W_B$ for each B. Finally, let ξ' be constructed from ξ by replacing each constant $b \in \mathrm{Dom}(B)$ occurring in ξ by $\mu_B(b)$, and replacing each variable v occurring in ξ by $\eta(v)$. It is easily verified that ξ' is a (safe) calculus expression mapping $h(\mathbf{P})$ to $h(\mathbf{Q})$.

We now argue that $\xi'(I) = \mu_{\mathbf{Q}} \circ \xi \circ \mu_{\mathbf{P}}^{-1}(I)$ for each fixed $I \in \mathbf{I}(\mathbf{P})$. Let $\mathbf{Q} = R_1, \cdots, R_n$. Then $\xi = (\xi_1, \cdots, \xi_n)$ where $\xi_j : \mathbf{P} \to R_j$. Let $\xi' = (\xi'_1, \cdots, \xi'_n)$. It suffices to show for some fixed $I \in \mathbf{I}(\mathbf{P})$ that $\xi'_j(I) = \mu_{R_j} \circ \xi_j \circ \mu_{\mathbf{P}}^{-1}(I)$. Let $R_j = B_1 \cdots B_t$ (or $B_1 \cdots B_k : B_{k+1} \cdots B_t$), let $\xi_j = \{(v_1, \cdots, v_t) | \psi\}$ and $\xi'_j = \{(\eta(v_1), \cdots, \eta(v_t)) | \psi'\}$. Suppose that α is a variable assignment for $\bigcup_{B \in \mathscr{B}} V'_B$ (i.e., $\alpha : \bigcup_{B \in \mathscr{B}} V'_B \to \mathbf{DOM}$ such that $\alpha(v) \in \mathrm{Dom}(B)$ for each $v \in V'_B \subseteq V_B$). Using an induction on subformulas, it is easily seen that

$$\psi(\alpha) \text{ is true of } \mu_{\mathbf{P}}^{-1}(I) \text{ iff } \psi'(\mu_{\mathbf{DOM}} \circ \alpha \circ \eta^{-1}) \text{ is true of } I.$$

Since the transformation $\alpha \mapsto \mu_{\mathbf{DOM}} \circ \alpha \circ \eta^{-1}$ maps the collection of variable assignments of $\bigcup_{B \in \mathscr{B}} V'_B$ *onto* the collection of variable assignments of $\bigcup_{B \in \mathscr{B}} W_B$, we see that

$$\begin{aligned}
\mu_{R_j} \circ \xi_j \circ \mu_{\mathbf{P}}^{-1}(I) &= \mu_{R_j}(\{\alpha(v_1, \cdots, v_t) | \psi(\alpha) \text{ is true of } \mu_{\mathbf{P}}^{-1}(I)\}) \\
&= \mu_{R_j}(\{(\alpha(v_1), \cdots, \alpha(v_t)) | \psi(\alpha) \text{ is true of } \mu_{\mathbf{P}}^{-1}(I)\}) \\
&= \{(\mu_{\mathbf{DOM}} \circ \alpha(v_1), \cdots, \mu_{\mathbf{DOM}} \circ \alpha(v_t)) | \psi(\alpha) \text{ is true of } \mu_{\mathbf{P}}^{-1}(I)\}) \\
&= \{\mu_{\mathbf{DOM}} \circ \alpha \circ \eta^{-1}(\eta(v_1), \cdots, \eta(v_t)) | \psi(\alpha) \text{ is true of } \mu_{\mathbf{P}}^{-1}(I)\}) \\
&= \{\mu_{\mathbf{DOM}} \circ \alpha \circ \eta^{-1}(\eta(v_1), \cdots, \eta(v_t)) | \psi'(\mu_{\mathbf{DOM}} \circ \alpha \circ \eta^{-1}) \text{ is true of } I\}) \\
&= \{\alpha'(\eta(v_1), \cdots, \eta(v_n)) | \psi'(\alpha') \text{ is true of } I\} \\
&= \xi'_j(I).
\end{aligned}$$

Thus, $\xi'_j(I) = \mu_{R_j} \circ \xi \circ \mu_{\mathbf{P}}^{-1}(I)$, and more generally, $\xi'(I) = \mu_{\mathbf{Q}} \circ \xi \circ \mu_{\mathbf{P}}^{-1}(I)$. □

Acknowledgments. The author expresses his gratitude to the participants of the USC Seminar on the Theory of Databases, and to Serge Abiteboul, for discussions which lead to several improvements of this paper. He also thanks two anonymous referees for careful readings of the paper and numerous helpful suggestions.

REFERENCES

[1] A. V. AHO AND J. D. ULLMAN, *Universality of data retrieval languages*, Symposium on Principles of Programming Languages, 1979, pp. 110-120.

[1a] S. ABITEBOUL AND R. HULL, IFO: *A formal semantic database model*, Technical Report TR-84-304, Dept. Computer Science, Univ. Southern California, Los Angeles, April 1984. Preliminary version appears in Proc. Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, April 1984, pp. 119-132.

[2] P. ATZENI, G. AUSIELLO, C. BATINI AND M. MOSCARINI, *Inclusion and equivalence between relational database schemata*, Theor. Comput. Sci., 19 (1982), pp. 267-285.

[3] G. AUSIELLO, C. BATINI AND M. MOSCARINI, *On the equivalence among database schemata*, Proc. International Conference on Data Bases, Aberdeen, 1980.

[4] C. BEERI, P. A. BERNSTEIN AND N. GOODMAN, *A sophisticate's introduction to database normalization theory*, Proc. 4th International Conference on Very Large Data Bases, 1978, pp. 113-124.

[5] C. BEERI, A. O. MENDELZON, Y. SAGIV AND J. D. ULLMAN, *Equivalence of relational database schemes*, this Journal, 10, (1981), pp. 352-370.

[6] P. A. BERNSTEIN, *Synthesizing third normal form relations from functional dependencies*, ACM Trans. Database Systems, 1 (1976), pp. 277-298.

[7] S. A. BORKIN, *Data Models: A Semantic Approach for Database Systems*, MIT Press, Cambridge, MA, 1980.

[8] R. BROWN AND D. S. PARKER, LAURA: *A formal data model and her logical design methodology*, VLDB, 1983, pp. 206-218.

[9] M. A. CASANOVA AND V. M. P. VIDAL, *Towards a sound view integration methodology*, Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1983, pp. 36-47.

[10] D. D. CHAMBERLIN, J. N. GRAY AND I. L. TRAIGER, *Views, authorization and locking in a relational database system*, Proc. AFIPS NCC 44, 1975, pp. 425-430.

[11] P. P. CHEN, *The entity-relationship model—toward a unified view of data*, ACM Trans. Database Systems, 1, (1976), pp. 9-36.

[12] W. CHU AND V. T. TO, *A hierarchical conceptual data model for data translation in a heterogeneous database system*, in Entity-Relationship Approach to Systems Analysis and Design, P. P. Chen, ed., North-Holland, Amsterdam, 1980.

[13] E. F. CODD, *Further normalization of the data base relational model*, in Data Base Systems, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 33-64.

[14] ———, *Relational completeness of database sublanguages*, Data Base Systems, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65-98.

[15] T. CONNORS, *Equivalence of projection-join views by query capacity*, Ph.D. thesis, Computer Science Dept., Univ. Southern California, Los Angeles, 1984; extended abstract appears in Proc. Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 1985, pp. 143-148.

[16] R. FAGIN, *A normal form for relational databases that is based on domains and keys*, ACM Trans. Database Systems, 6 (1981), pp. 387-415.

[17] M. HAMMER AND D. MCLEOD, *Database description with* SDM: *A semantic database model*, ACM Trans. Database Systems, 6 (1981), pp. 351-386.

[18] M. A. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, Reading, MA, 1978.

[19] D. M. HEIMBIGNER AND D. MCLEOD, A federated architecture for information management, ACM Trans. on Office Information Systems, 3 (1985), pp. 253-278.

[20] R. HULL AND C. K. YAP, *The format model: A theory of database organization*, J. ACM, 31 (1984), pp. 518-537.

[21] B. E. JACOBS, *Applications of database logic to the view update problem*, Univ. Maryland, College Park, MD, 1980.

[22]———, *On database logic*, J. ACM, 29 (1982), pp. 310-332.

[23] L. KERSCHBERG AND J. E. S. PACHECO, *A functional data base model*, Pontificia Universidade Catolica do Rio de Janeiro, Rio de Janeiro, Brazil, Feb., 1976.

[24] R. KING AND D. MCLEOD, *A database design methodology and tool for information systems*, ACM Trans. on Office Information Systems, 3 (1985), pp. 2-21.

[25] A. KLUG, *Entity-relationship views over uninterpreted enterprise schemas*, in Entity-Relationship Approach to Systems Analysis and Design, P. P. Chen, ed., North-Holland, Amsterdam, 1980, pp. 52-72.

[26] Y. E. LIEN, *On the semantics of the entity-relationship data model*, in Entity-Relationship Approach to Systems Analysis and Design, P. P. Chen, ed., North-Holland, Amsterdam, 1980, pp. 131-146.

[27] ———, *On the equivalence of database models*, J. ACM, 29 (1982), pp. 333-362.

[28] T.-W. LING, F. W. TOMPA AND T. KAMEDA, *An improved third normal form for relational databases*, ACM Trans. Database Systems, 6 (1981), pp. 329-346.

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

[30] D. MAIER, A. O. MENDELZON, F. SADRI AND J. D. ULLMAN, *Adequacy of decompositions of relational databases*, J. Comput. Systems Sci., 21 (1980), pp. 368-379.

[31] A. MOTRO, *Construction and interrogation of virtual databases*, TR-83-211, Univ. Southern California, Los Angeles, April, 1983.

[32] A. MOTRO AND P. BUNEMAN, *Constructing superviews*, Proc. ACM SIGMOD International Conference on the Management of Data, 1981, pp. 56-64.

[33] J. MYLOPOULOS, P. A. BERNSTEIN AND H. K. T. WONG, *A language facility for designing database-intensive applications*, ACM Trans. Database Systems, 5 (1980), pp. 185-207.

[34] C. Ó'DÚNLAING AND C. K. YAP, *Generic transformation of data structures*, Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982, pp. 186-195.

[35] F. P. RAMSEY, *On a problem in formal logic*, Proc. London Math. Soc. Ser 2, 30 (1930), pp. 264-286.

[36] R. REITER, *Equality and domain closure in first-order databases*, J. ACM, 27 (1980), pp. 235-249.

[37]———, *On the integrity of typed first order databases*, in Advances in Database Theory, H. Gallair, J. Minker and J.-M. Nicolas, eds., Plenum Press, New York, 1981, pp. 137-157.

[38] D. SHIPMAN, *The functional data model and the data language* DAPLEX, ACM Trans. Database Systems, 6 (1981), pp. 140-173.

[39] J. D. ULLMAN, *Principles of Database Systems*, 2nd ed., Computer Science Press, Potomac, MD, 1982.

[40] S. B. YAO, V. WADDLE AND B. C. HOUSEL, *View modeling and integration using the functional data model*, IEEE Trans. Soft. Engng, SE-8, 6 (1982), pp. 544-553.

[41] C. ZANIOLO, *A new normal form for the design of relational database schemata*, ACM Trans. Database Systems, 7 (1982), pp. 489-499.

SIAM J. COMPUT.
Vol. 15, No. 3, August 1986
© 1986 Society for Industrial and Applied Mathematics
017

ANALYSIS OF HENRIKSEN'S ALGORITHM FOR THE SIMULATION EVENT SET*

JEFFREY H. KINGSTON†

Abstract. An algorithm for the scheduling of events in a discrete-event simulation system, due to J. O. Henriksen, is presented. An $O(n^{1/2})$ upper bound on the average cost of an insertion during steady-state operation is derived. Although asymptotically inferior to the logarithmic performance of heaps, the bound is robust: it has a small constant of proportionality and is independent of the distribution of insertions.

Key words. analysis of algorithms, simulation, priority queues

AMS(MOS) subject classifications. 68C25, 68J05

1. Introduction. At the heart of a general-purpose simulation system (such as SIMULA, SIMSCRIPT etc.) lies the event scheduler. Its task is to maintain a set of *event notices*, each containing a key called the *event time*, and to implement two basic operations:

(i) *extract_min*: the notice with the smallest key is removed; and

(ii) *insert*(p): event notice p is inserted into its place in the set.

Other operations (*cancel*, for example) are usually implemented as well.

Since these operations define a priority queue, many well-known data structures may be used to hold the events. But most of them are rendered unsuitable by the peculiar demands of the application, which requires fast operation over an enormous range of operating conditions.

Vaucher and Duval [16] showed that the simple linked list in common use is often very slow, and suggested two lines of research. The first, indexing, begins an insertion by using the event time as an index into a table of pointers to sublists [16], [4], [6], [15]. Although very fast under optimal settings, indexing is flawed by an excessive sensitivity to its operating parameters [5]. Some recent papers [1], [14] reject indexing as a general-purpose solution, for this reason.

The second approach consists of algorithms based entirely on comparisons. The binary search tree [13] and its relative, the p-tree [10], are highly variable in performance, as shown empirically by Vaucher and Duval [16] and recently confirmed by the author's theoretical work [11]. Balanced tree schemes (AVL, 2-3 trees, etc. [13]) are $O(\log(n))$ but have excessive overhead [10]. The heap [7] is efficient, but may be unsuitable in some applications (for example, applications requiring successors) [12]. There is also an algorithm by Wyman [18].

Henriksen's "binary search indexed list" algorithm [8] is an improvement on Wyman's. It has been incorporated into the simulation languages GPSS/H and SLAM [9]. In this paper we study its average-case performance, and derive a robust $O(n^{1/2})$ upper bound.

2. The hold model. Before proceeding to Henriksen's algorithm, we present the theoretical model upon which its analysis is based.

Let the current notice be the event notice returned by the most recent *extract_min* operation (the current notice is not a member of the event set). The current time is the event time of the current notice, when first extracted. Simulated time always flows

* Received by the editors April 24, 1984, and in final revised form August 19, 1985.

† Department of Computer Science, University of Iowa, Iowa City, Iowa 52242.

forwards, so the event time of every notice must equal or exceed the current time. As the simulation proceeds, the event times increase without bound, but we discount this by measuring all times relative to the current time.

In a real simulation, the event set is shaped by many repetitions of the operations given above. No tractable model could encompass the range of possible simulations that results. But there is a large subset that can be usefully modelled.

```
procedure hold(X: real);
var current: notice;
begin
  current := extract_min;
  current↑.evtime := current↑.evtime + X;
  insert(current)
end;
```

FIG. 1. *Pascal implementation of the operation hold* (X).

Let the event set contain n notices, and consider the operation *hold* (X), which extracts the minimum notice, adds X to its event time, and re-inserts it (Fig. 1). We form a sequence of operations *hold* (X_1), *hold* $(X_2), \cdots$ by letting $X_1, X_2, \cdots$ be identically distributed random variables with probability density function $f(x)$. This is the hold model of a simulation. In this paper we require $f(x)$ to be continuous; we let the cumulative density function of the X_i be

$$F(x) = \int_0^x f(x)\, dx$$

and let

$$a = E(X_i) = \int_0^\infty xf(x)\, dx.$$

Vaucher [17] has shown using renewal theory [3], that under the hold model the event set will approach a steady state, and that, immediately after the current notice is extracted, the relative event time of each of the $n-1$ remaining notices is a random variable distributed as

$$g(x) = \frac{1 - F(x)}{a},$$

independently of the other notices. Kingston [11] has shown that any simulation that reaches a steady state in which *extract_min* operations are much more common than deletions of other notices, behaves as if operating under the hold model for some pdf $f(x)$. Therefore, the results developed here are applicable to this wide class of simulations.

Five distributions $f(x)$ will be used as examples (Table 1). They come mostly from [16] and were chosen to test the algorithms studied there under a wide variety of conditions.

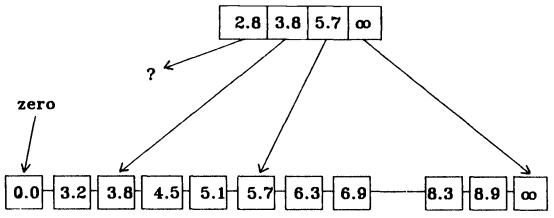
3. Henriksen's algorithm. Henriksen's algorithm [8] is a clever combination of binary search and list insertion (Fig. 2). The list of event notices is an ordered, doubly-linked structure with a sentinel at each end. Above it lies a binary search vector. Each vector entry consists of an event time and a pointer. If this event time is strictly

TABLE 1
The example probability distributions (all have $a = 1$).

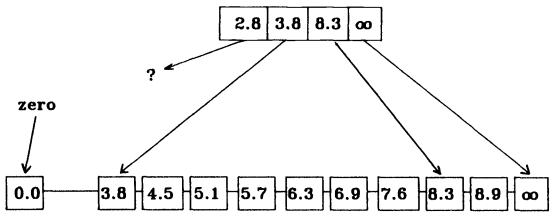
Distribution	$f(x)$		$g(x)$	
1. Exponential	e^{-x}	$x \geqq 0$	e^{-x}	$x \geqq 0$
2. Uniform 0-2	$\frac{1}{2}$	$0 \leqq x \leqq 2$	$1-\frac{x}{2}$	$0 \leqq x \leqq 2$
	0	$x > 2$	0	$x > 2$
3. Uniform 0.9-1.1	0	$0 \leqq x \leqq 0.9$	1	$0 \leqq x \leqq 0.9$
	$\frac{1}{0.2}$	$0.9 < x \leqq 1.1$	$\frac{1.1-x}{0.2}$	$0.9 < x \leqq 1.1$
	0	$x > 1.1$	0	$x > 1.1$
4. Bimodal, $q = 2/21$	$\frac{0.9}{q}$	$x \leqq q$	$1-\frac{0.9x}{q}$	$x \leqq q$
	0	$q < x \leqq 100q$	0.1	$q < x \leqq 100q$
	$\frac{0.1}{q}$	$100q < x \leqq 101q$	$10.1-\frac{0.1x}{q}$	$100q < x \leqq 101q$
	0	$x > 101q$	0	$x > 101q$
5. Triangular	$\frac{8x}{9}$	$x \leqq 1.5$	$1-\frac{4x^2}{9}$	$x \leqq 1.5$
	0	$x > 1.5$	0	$x > 1.5$

greater than the current time, the pointer leads down to an event notice with that time. Otherwise the pointer is undefined.

The minimum notice is found as *zero*↑.*rlink*, and may be extracted in constant time without altering the search vector. An insertion at time t is made by a binary search of the vector to find the first entry with time strictly greater than t, then a linear search to the left along the associated sublist.



(a) A typical configuration of the structure.



(b) The structure of (a) after the operation *hold* (4.4). One pull operation occurred.

FIG. 2. *Henriksen's algorithm in operation. Here vecsize* = 4, $m = 2$.

If the structure is to be efficient, the sublists must be kept short. Henriksen's balancing scheme is a beautiful compromise between cost and effectiveness. As a sublist is searched during an insertion, a count is kept of the number of notices scanned. If this count reaches a certain fixed number m (usually four), one vector entry is altered to point to the notice currently being scanned (Fig. 2b); Henriksen calls this a "pull operation". The search is then resumed with the count reset to 0.

During an insertion, pull operations at the low end of the vector may cause it to overflow. If this occurs, the vector size is doubled (for convenience this size is a power of two). In Henriksen's implementation this necessitates a re-initialization of the vector, an expensive (though rare) operation. The author's implementation (Fig. 3), by keeping the vector at the top end of available storage, reduces the cost to the resetting of a few vector indices.

Conversely, during an *extract_min* operation it may be possible to halve the vector size. This is probably good practice, but it is implemented here more for the theoretical reasons given in the next section.

In Fig. 3, the arrays *ptrvec* and *timevec* hold the vector entries. Procedure *setvec* (*newsize*) sets the vector size to *newsize* (unless *newsize* is too large, in which case the run is aborted), and *search* (t) returns the index of the first vector entry with time strictly greater than t. Also given are the two basic operations *extract_min* and *insert* (p). The structure may be initialised with a vector of length one, containing only the sentinel.

4. An upper bound on vector scans. As described above, an insertion begins with a binary search of the pointer vector. In this section we find an upper bound on the expected number of scans made during this search.

Let $c = search\ (current\uparrow.evtime)$ be the vector index of the left-most defined pointer, where *current* is defined in Fig. 1. In conformity with the relative origin used by the hold model, let I_i be the half-open time interval

$$I_i = [timevec\,[c+i-1] - current\uparrow.evtime,\ timevec\,[c+i] - current\uparrow.evtime).$$

Thus $X \in I_i$ if and only if $search\ (current\uparrow.evtime + X) = c+i$; we will also refer to pointer i, meaning $ptrvec\,[c+i]$. The leftmost interval, I_0, is called the current interval and contains 0. The rightmost interval we label I_{v-1}, so that there are v intervals altogether; it is of the form $[x, \infty)$ for some x.

When $p\uparrow.evtime - current\uparrow.evtime \in I_i$ we will say that event notice p lies in I_i. Since the intervals I_i partition the positive real numbers, every notice lies in exactly one of them. Let S_i be the number of notices in interval I_i, and define a Henriksen structure to be the v-tuple $(S_0, \cdots, S_{v-1})$. As the simulation proceeds, the S_i are constantly changing; in the steady state, we may treat them as random variables and define

$$p_{k_i} = \text{Prob}\ (S_i = k_i).$$

It is immediate that $S_i \leqq n-1$ for all i; there is also a useful lower bound:

LEMMA 1. *$S_i \geqq m$ for all $i > 0$.*

Proof. By induction. We assume that this condition holds when the simulation begins (for example, with an empty list, or when $v = 1$). It remains to prove that it is preserved by the basic operations.

First, deletion of the minimum notice cannot violate the condition, since it resides in the current interval, I_0, about which no statement is made.

Second, an insertion into interval I_i will either increase S_i by one (if there is no pull operation) or reduce it to m (if a pull occurs); in either case $S_i \geqq m$ afterwards.

```
const maxsize = 1024; m = 4;

type  notice = ↑record evtime: real;
                       llink, rlink: notice;
               end;

var   ptrvec:  array[0..maxsize-1] of notice; zero: notice;
      timevec: array[0..maxsize-1] of real;
      vecsize, leftlim, starti, startj: integer;

procedure setvec(newsize: integer); { set vecsize to newsize }
begin
  if newsize > maxsize then abort;
  vecsize := newsize; leftlim := maxsize - vecsize;
  starti := maxsize - vecsize/2 - 1; startj := vecsize/4;
end;

function search(t: real): integer;
var i, j: integer;
begin
  i := starti; j := startj;
  while j > 0 do begin
    if timevec[i] <= t then i := i+j else i := i-j;
    j := j/2
  end;
  if timevec[i] <= t then search := i+1 else search := i
end;

function extract_min: notice;
begin
  if (timevec[starti] <= zero↑.rlink↑.evtime) and (vecsize > 2)
    then setvec(vecsize/2);
  extract_min := zero↑.rlink;
  zero↑.rlink := zero↑.rlink↑.rlink; zero↑.rlink↑.llink := zero
end;

procedure insert(p: notice);
var i, count: integer; q: notice;
begin
  i := search(p↑.evtime); q := ptrvec[i]↑.llink; count := 0;
  while p↑.evtime < q↑.evtime do begin
    count := count+1;
    if count = m then begin { pull operation }
      if i <= leftlim then setvec(vecsize * 2);
      i := i-1; count := 0;
      ptrvec[i] := q; timevec[i] := q↑.evtime;
    end;
    q := q↑.llink
  end;
  p↑.llink := q; p↑.rlink := q↑.rlink;
  p↑.rlink↑.llink := p; q↑.rlink := p;
end;
```

FIG. 3. *Pascal implementation of Henriksen's algorithm.*

If there is a pull, S_{i-1} will be affected, either increasing by $S_i - m + 1$ or (in the case of a second pull) reducing to m. Hence $S_{i-1} \geqq m$ afterwards, and so on through any other affected intervals, thus completing the proof.

Let $C_1(X)$ be the number of scans made by procedure *search* during the operation *hold* (X). A simple induction shows that

$$C_1(X) = \log_2 (vecsize) \quad \text{for all } X.$$

We know that $vecsize \geqq v$, and that *vecsize* is a power of 2. But the implementation described in this paper, by halving *vecsize* whenever possible during an *extract_min* operation, also guarantees that $v > vecsize/2$, so we must have

$$E[C_1(X)] = \log_2 (vecsize) = \lceil \log_2 v \rceil. \tag{1}$$

Now by Lemma 1, at least m notices lie in each of $I_1, \cdots, I_{v-1}$, making at least $m(v-1)$ notices altogether; since there are $n-1$ notices in the list, we must have $m(v-1) \leqq n-1$, and

$$v \leqq \frac{n-1}{m} + 1,$$

so by (1),

$$E[C_1(X)] \leqq \left\lceil \log_2 \left(\frac{n-1}{m} + 1 \right) \right\rceil \simeq \lceil \log_2 (n-1) - \log_2 (m) \rceil \tag{2}$$

and this is the required bound on vector scans.

Informally, when insertions are spread evenly through the list, there is little opportunity for large numbers of notices to accumulate in any interval, since insertions there will cause pull operations which redistribute the notices. In such cases, there are many intervals and the bound is often achieved (see Fig. 4).

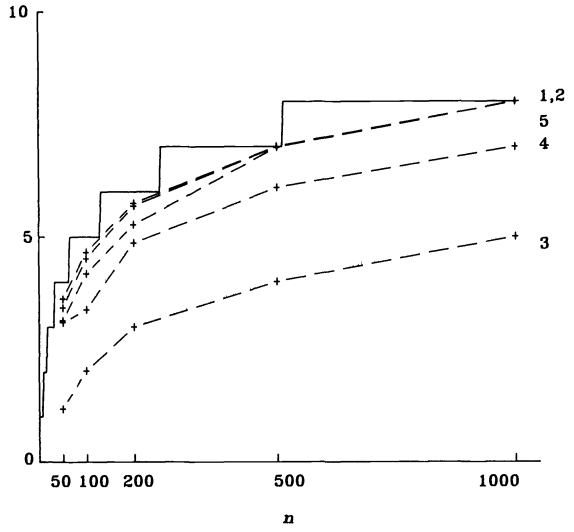


FIG. 4. *The bound on* $E[C_1(X)]$ *given by* (2), *as a function of* n, *with observations of* $E[C_1(X)]$ *for the five example distributions. The observations are averages over* 1,000,000 *hold operations*; $m = 4$.

On the other hand, when insertions are absent from a significant portion of the list, large numbers of notices may gather in one interval, thereby reducing the total number of intervals. For example, with distribution 3, about 90% of the notices lie in

I_0 (which contains [0, 0.9]), because, in the total absence of insertions into this range, there are no pull operations to break up this accumulation. We could ignore these notices and replace $(n-1)$ by $(n-1)/10$ in (2) in this case.

5. An upper bound on sublist scans. The analysis of sublist scans is considerably more complex than that of the vector scans presented in the previous section. Here, we must look in detail at the way notices are distributed among the intervals, and at the way insertions are performed.

In doing so, we will make the following reasonable assumptions. It is clear that an insertion at time X (relative to the current time) has a purely local effect on the structure, confined to an unknown (but generally small) number of intervals near X, in which pull operations occur. Thus, over this neighbourhood of X affected by the insertion, we may assume that, for sufficiently large n,

(i) $f(x)$ and $g(x)$ are constant; and

(ii) the random variables S_i are identically distributed.

The first is by the continuity of $f(x)$ (and hence $g(x)$); the second is by symmetry: there is nothing to distinguish neighbouring intervals I_i from each other.

Let $C_2(X)$ be the expected number of scans performed by the operation *hold* (X) during this second, or sublist phase of the insertion. Since X is distributed as $f(x)$, the expected number of sublist scans under the hold model is

$$E[C_2(X)] = \int_0^\infty C_2(x) f(x)\, dx. \tag{3}$$

Our aim is to find an upper bound to $C_2(X)$, which by substitution in (3) will bound $E[C_2(X)]$. We begin by relating $C_2(X)$ to the S_i.

LEMMA 2. *Let X lie in the neighbourhood of I_h. Then*

$$C_2(X) = \frac{E[S_h^2]/E[S_h]+1}{2}. \tag{4}$$

Proof. Suppose there are K intervals in the neighbourhood of I_h. By assumption (ii), there are about $KE[S_h]$ notices altogether in this neighbourhood, and hence $KE[S_h]$ "slots" available for X to lie in. Since Kp_j intervals contain exactly j notices, there are jKp_j notices (and hence slots) lying in intervals I_k such that $S_k = j$; so, assuming the slots are of equal width,

$$\text{Prob}\,(X \in I_k \text{ s.t. } S_k = j) = \frac{jKp_j}{KE[S_h]} = \frac{jp_j}{E[S_h]}.$$

An insertion into an interval I_k that contains j notices, will be made with equal probability into any one of the j available slots; on average, $(j+1)/2$ scans will be required. Summing over j,

$$\begin{aligned} C_2(X) &= \sum_{j=m}^{\infty} \frac{j+1}{2} \text{Prob}\,(X \in I_k \text{ s.t. } S_k = j) \\ &= \sum_{j=m}^{\infty} \frac{j+1}{2} \cdot \frac{jp_j}{E[S_h]} = \frac{E[S_h^2]/E[S_h]+1}{2}, \end{aligned}$$

thus completing the proof.

5.1. The main theorem. We now proceed to the main theorem, in which we seek to combine an algebraic description of the way *hold* (X) operations are made, with our knowledge that, over time, these operations act to maintain the steady state.

Let $S=(S_0, \cdots, S_{v-1})$ be a Henriksen structure, and let X be a random variable with pdf $f(x)$, representing a *hold* (X) operation. The pair (S, X) is our model of a random *hold* (X) operation into a Henriksen structure.

When this operation (S, X) occurs, some vector pointers near the insertion point may be moved back through the list by pull operations. We are interested in the effect of the insertion at X upon a typical pointer, say pointer h. Let $N_h(S, X)$ be the number of notices that pointer h is moved across (if any) during an insertion at time X (see Fig. 5).

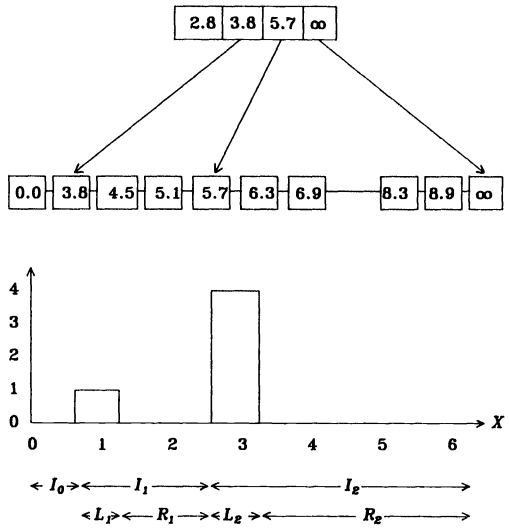


FIG. 5. *Example of* L_i, R_i *and* $N_h(S, X)$. *This structure is from Fig.* 2a *after the operation extract_min, so the current time is* 3.2. *As before, vecsize* $=4$, $m=2$; *also* $v=3$, $S=(0, 3, 5)$. *The graph is* $N_h(S, X)$ *for* $h=0$. *It indicates that an insertion in the absolute range* $5.7 \leqq X < 6.3$ *will cause pointer* 0, *now pointing to the notice with time* 3.8, *to be moved four notices to the right; while an insertion in the absolute range* $3.8 \leqq X < 4.5$ *will cause pointer* 0 *to move one notice to the right. All other insertions leave pointer* 0 *unaffected.*

LEMMA 3. *Let* x_h *lie in the neighbourhood of* I_h. *Then*

$$E[N_h(S, X)] = ag(x_h). \tag{5}$$

Proof. As simulated time advances, the endpoints of I_h gradually decrease (recall that all times are measured relative to the current time). This effect is offset by occasional pull operations on pointers h and $h-1$, which cause the endpoints of I_h to increase suddenly. The existence of a steady state implies that this fluctuation of I_h will be about some steady-state value. We have assumed that $g(x)$ is constant over this range of fluctuation, say with value $g(x_h)$.

The probability of an insertion occurring to the right of I_h is therefore $1-F(x_h) = ag(x_h)$. If a large number, say K, of *hold* (X) operations are performed, about $Kag\ (x_h)$ notices will be inserted to the right of I_h. At the same time, pointer h will be moved to the right across $KE[N_h(S, X)]$ notices, where E denotes expectation over S and X. But in order to maintain the steady state, these must be equal, thus proving (5).

This lemma expresses our knowledge of the structure being in a steady state. We now make a detailed study of the insertion process, in order to determine $E[N_h(S, X)]$ in another way.

First, we construct a pdf for (S, X). S and X are independent, but in order to make the analysis tractable we assume in addition that the S_i are mutually independent

also. This leads immediately to the joint, mixed pdf

$$\text{Prob}\,((S, X) = ((k_0, \cdots, k_{v-1}), x)) = p_{k_0} \cdots p_{k_{v-1}} f(x)\,dx$$

and thus

$$(6)\quad E[N_h(S, X)] = \sum_{k_0=1}^{\infty} \cdots \sum_{k_{v-1}=1}^{\infty} \int_{x=0}^{\infty} N_h((k_0, \cdots, k_{v-1}), x) f(x)\,dx \cdot p_{k_0} \cdots p_{k_{v-1}}.$$

Next we investigate $N_h((k_0, \cdots, k_{v-1}), x)$. Suppose $x \in I_i$ for some i. If $i \leqq h$, pointer h is unaffected by an insertion at x, so $N_h((k_0, \cdots, k_{v-1}), x) = 0$. For $i > h$ the case is more complicated. I_i can be divided into two subintervals L_i and R_i, defined as follows: R_i, the right-hand part of the interval I_i, contains those insertion points x where an insertion will cause less than $i-h$ pull operations, therefore leaving pointer h unaffected; hence $N_h((k_0, \cdots, k_{v-1}), x) = 0$ for $x \in R_i$. In L_i, the remainder of the interval, insertions are sufficiently expensive to cause at least $i-h$ pull operations (see Fig. 5). For all $x \in L_i$, an insertion at x will cause pointer h to be moved across

$$N_h((k_0, \cdots, k_{v-1}), x) = \sum_{j=h+1}^{i} (k_j - m) \text{ notices} \quad (x \in L_i).$$

This is true because pointer h moves across exactly those notices expelled from $I_{h+1}, \cdots, I_i$, and there are $k_{h+1} + \cdots + k_i$ notices in these intervals before the insertion, but only $m + \cdots + m$ afterwards. Now summing over the regions L_i (where $N_h((k_0, \cdots, k_{v-1}), x)$ is nonzero),

$$\begin{aligned}
&\int_{x=0}^{\infty} N_h((k_0, \cdots, k_{v-1}), x) f(x)\,dx \\
&\quad = \sum_{i=h+1}^{v} \int_{L_i} N_h((k_0, \cdots, k_{v-1}), x) f(x)\,dx \\
&\quad = \sum_{i=h+1}^{v} N_h((k_0, \cdots, k_{v-1}), x_i) \int_{L_i} f(x)\,dx \quad \text{for any } x_i \in L_i \qquad (7) \\
&\quad \simeq \sum_{i=h+1}^{v} N_h((k_0, \cdots, k_{v-1}), x_i) |L_i| f(x_i) \\
&\quad = \sum_{i=h+1}^{v} \sum_{j=h+1}^{i} (k_j - m) |L_i| f(x_i)
\end{aligned}$$

where $|L_i|$ is the length of the time interval L_i; this approximation is valid since $|L_i|$ is small enough to allow us to assume that $f(x)$ is constant over L_i.

Now L_i contains all those points $x_i \in I_i$ where an insertion causes at least $i-h$ pull operations, i.e. those points to the left of the rightmost $(i-h)m$ notices of I_i. There must, therefore, be $\max(k_i - (i-h)m, 0)$ notices in L_i, and since the density of notices near x_i is $(n-1)g(x_i)$ notices per time unit, the length of L_i is approximately

$$|L_i| = \frac{\max(k_i - (i-h)m, 0)}{(n-1)g(x_i)}.$$

Substituting this into (7) gives

$$\int_{x=0}^{\infty} N_h((k_0, \cdots, k_{v-1}), x) f(x)\,dx = \sum_{i=h+1}^{v} \sum_{j=h+1}^{i} (k_j - m) \frac{\max(k_i - (i-h)m, 0)}{(n-1)g(x_i)} f(x_i),$$

and now by (6)

$$E[N_h(S,X)] = \sum_{k_0=1}^{\infty} \cdots \sum_{k_{v-1}=1}^{\infty} \sum_{i=h+1}^{v} \sum_{j=h+1}^{i} (k_j - m) \frac{\max(k_i - (i-h)m, 0)}{(n-1)g(x_i)} f(x_i) p_{k_0} \cdots p_{k_{v-1}}$$

$$= \frac{f(x_h)}{(n-1)g(x_h)} \sum_{i=h+1}^{v} \sum_{j=h+1}^{i} \sum_{k_0=1}^{\infty} \cdots \sum_{k_{v-1}=1}^{\infty} (k_j - m) \max(k_i - (i-h)m, 0) p_{k_0} \cdots p_{k_{v-1}},$$

interchanging the order of summation and noting that x_h is in the neighbourhood of x_i. By Appendix 1, this simplifies to

$$E[N_h(S,X)] \geqq \frac{f(x_h)}{6m^2(n-1)g(x_h)} [E[S_h + 2m]E[(S_h - m)^3] + 3m^2 E[(S_h - m)^2] - m^2 E[S_h]E[S_h - m]]$$

and by (5) we have proven

THEOREM 1. *Let X lie in the neighbourhood of I_h. Then*

$$E[S_h + 2m]E[(S_h - m)^3] + 3m^2 E[(S_h - m)^2] - m^2 E[S_h]E[S_h - m] \leqq \frac{6m^2(n-1)ag^2(X)}{f(X)}.$$

This theorem is our main result, representing our knowledge of the inner workings of Henriksen's algorithm. The two low-order terms on the left-hand side could be omitted, except that there is no absolute guarantee that their sum will be positive. Also, by Appendix 1, the theorem could be expressed as an approximate equality; however, this is not necessary for our purposes.

5.2. Application of Theorem 1 to sublist scans. In order to use Theorem 1, we must relate the cube of S_h appearing there to the square appearing in (4).

LEMMA 4.

$$\frac{E[(S_h - m)^3]}{E[S_h - m]} \geqq \left[\frac{E[(S_h - m)^2]}{E[S_h - m]}\right]^2 \quad \text{and} \quad \frac{E[(S_h - m)^2]}{E[S_h - m]} \geqq E[S_h - m].$$

Proof. These inequalities are consequences of the Cauchy-Schwarz inequality [2], which states that, for any two sequences (u_i) and (v_i) of positive numbers,

$$\left[\sum_i u_i v_i\right]^2 \leqq \sum_i u_i^2 \sum_i v_i^2.$$

The first inequality is immediate from the substitution

$$u_i = \left[\frac{(i-m)^3 p_i}{\sum_i (i-m)p_i}\right]^{1/2}, \qquad v_i = \left[\frac{(i-m)p_i}{\sum_i (i-m)p_i}\right]^{1/2};$$

and the second from

$$u_i = [(i-m)^2 p_i]^{1/2}, \qquad v_i = p_i^{1/2}.$$

We are now able to bound $C_2(X)$. Dividing Theorem 1 by $E[S_h + 2m]E[S_h - m]$,

$$\frac{E[(S_h - m)^3]}{E[S_h - m]} + \frac{3m^2}{E[S_h + 2m]} \cdot \frac{E[(S_h - m)^2]}{E[S_h - m]} - \frac{m^2 E[S_h]}{E[S_h + 2m]} \leqq \frac{6m^2 a(n-1)g^2(X)}{E[S_h - m]E[S_h + 2m]f(X)},$$

$$\left[\frac{E[(S_h-m)^2]}{E[S_h-m]}\right]^2+\frac{3m^2}{E[S_h+2m]}[E[S_h-m]]-\frac{m^2E[S_h]}{E[S_h+2m]}$$
$$\leqq\frac{6m^2a(n-1)g^2(X)}{E[S_h-m]E[S_h+2m]f(X)}$$

by Lemma 4. Now letting $s=E[S_h]$ and rearranging,

$$\frac{E[(S_h-m)^2]}{s-m}\leqq\left[\frac{6m^2a(n-1)g^2(X)}{(s-m)(s+2m)f(X)}+\frac{m^2s}{s+2m}-\frac{3m^2(s-m)}{s+2m}\right]^{1/2};$$

$$\frac{E[S_h^2-2mS_h+m^2]}{s}\leqq\left[\frac{6m^2(s-m)}{s^2(s+2m)}\cdot\frac{a(n-1)g^2(X)}{f(X)}+\frac{m^2(s-m)^2}{s(s+2m)}-\frac{3m^2(s-m)^3}{s^2(s+2m)}\right]^{1/2};$$

$$\frac{E[S_h^2]}{s}-2m+\frac{m^2}{s}\leqq\left[\frac{6m^2(s-m)}{s^2(s+2m)}\cdot\frac{a(n-1)g^2(X)}{f(X)}\right]^{1/2}$$
$$+\left[\max\left(0,\frac{m^2(s-m)^2}{s(s+2m)}-\frac{3m^2(s-m)^3}{s^2(s+2m)}\right)\right]^{1/2}$$

using the inequalities $(a+b)^{1/2}\leqq a^{1/2}+b^{1/2}$ if $a, b\geqq 0$; and $(a+b)^{1/2}\leqq a^{1/2}$ if $a\geqq 0$ and $b<0$. Now $s\geqq m$ by Lemma 1, and given this condition we can show by elementary calculus that, for all s,

$$\left[\frac{6m^2(s-m)}{s^2(s+2m)}\right]^{1/2}\leqq 0.6267 \quad\text{and}\quad \left[\max\left(0,\frac{m^2(s-m)^2}{s(s+2m)}-\frac{3m^2(s-m)^3}{s^2(s+2m)}\right)\right]^{1/2}-\frac{m^2}{s}\leqq 0.$$

Therefore

$$\frac{E[S_h^2]}{E[S_h]}\leqq 0.6267\left[\frac{a(n-1)g^2(X)}{f(X)}\right]^{1/2}+2m,$$

and so by Lemma 2 we have established

THEOREM 2. *The expected number of sublist scans, $C_2(X)$, occurring during the operation hold (X) is bounded by*

$$C_2(X)\leqq 0.3134\left[\frac{a(n-1)g^2(X)}{f(X)}\right]^{1/2}+m+\tfrac{1}{2}.$$

Figure 6 shows some typical observations of $C_2(X)$ along with the corresponding bounds. It seems Theorem 2 is often quite close, although the inequalities used to prove Lemma 4 give no assurance that this will always be the case.

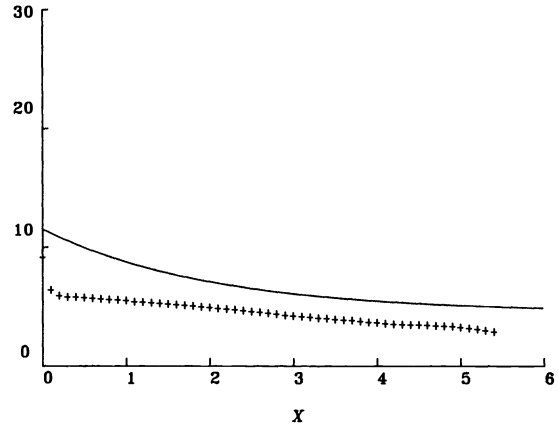
We have at once that the expected number of sublist scans is

$$E[C_2(X)]=\int_0^\infty C_2(x)f(x)\,dx$$
$$\leqq 0.3134[a(n-1)]^{1/2}\int_0^\infty g(x)f(x)^{1/2}\,dx+m+\tfrac{1}{2}. \tag{8}$$

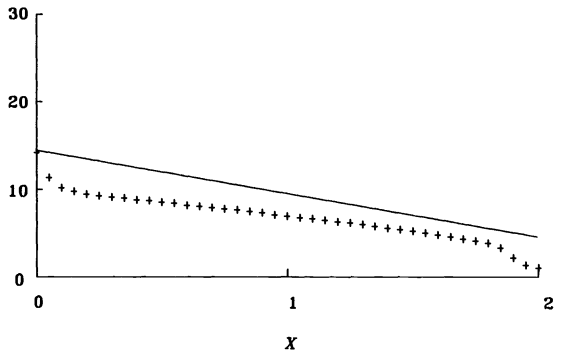
This inequality is of interest in its own right, and is tabulated for the various distributions in Table 2; but even more remarkable is the following distribution-independent bound.

THEOREM 3. *For any continuous* pdf $f(x)$,

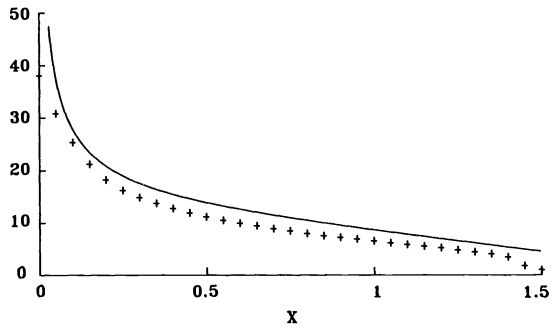
$$E[C_2(X)]\leqq 0.2216(n-1)^{1/2}+m+\tfrac{1}{2}.$$



(a) Exponential distribution



(b) Uniform 0-2 distribution



(c) Triangular distribution

FIG. 6. *The bound on $C_2(X)$ given by Theorem 2, with observations of $C_2(X)$, for the case $n = 500$, $m = 4$. Each data point is the average over 1,000,000 hold operations. Distributions 3 and 4 provide only a tiny range of useful data and have been omitted.*

TABLE 2
Upper bounds given by (8).

Distribution	Bound on $E[C_2(X)]$ given by (8)
1. Exponential	$0.2089(n-1)^{1/2}+m+1/2$
2. Uniform 0-2	$0.2216(n-1)^{1/2}+m+1/2$
3. Uniform 0.9-1.1	$0.0701(n-1)^{1/2}+m+1/2$
4. Bimodal	$0.0520(n-1)^{1/2}+m+1/2$
5. Triangular	$0.2068(n-1)^{1/2}+m+1/2$

Proof. By the Cauchy-Schwarz inequality for integrals,

$$\begin{aligned}\int_0^\infty g(x)f(x)^{1/2}\,dx &= \int_0^\infty g(x)^{1/2}[g(x)f(x)]^{1/2}\,dx\\ &\leqq \left[\int_0^\infty g(x)\,dx\right]^{1/2}\left[\int_0^\infty g(x)f(x)\,dx\right]^{1/2}\\ &= \left[\frac{-ag^2(x)}{2}\bigg|_0^\infty\right]^{1/2}\\ &= (2a)^{-1/2}.\end{aligned}$$

Substitution into (8) immediately yields the theorem.

This bound on the integral is achieved by distribution 2, and hence Theorem 3 is the strongest distribution-independent statement obtainable from Theorem 2.

6. Conclusion. Although the relative cost of a vector and sublist scan is implementation-dependent, it seems worthwhile to add (2) to Theorem 2 to obtain a bound on the expected cost of the operation *hold* (X):

$$\begin{aligned}C(X) &= C_1(X)+C_2(X)\\ &\leqq \lceil \log_2(n-1)-\log_2(m)\rceil + 0.3134\left[\frac{a(n-1)g^2(X)}{f(X)}\right]^{1/2}+m+\tfrac{1}{2};\end{aligned}$$

and thus

$$E[C(X)]\leqq \lceil \log_2(n-1)-\log_2(m)\rceil + 0.3134[a(n-1)]^{1/2}\int_0^\infty g(x)f(x)^{1/2}\,dx+m+\tfrac{1}{2};$$

and finally, from (2) and Theorem 3 the distribution-independent bound

$$E[C(X)]\leqq \lceil \log_2(n-1)-\log_2(m)\rceil + 0.2216(n-1)^{1/2}+m+\tfrac{1}{2}. \tag{9}$$

We have not mentioned the cost of pull operations, but they are not expensive, and, if $m\geqq 4$, will happen infrequently by comparison with sublist scans (the expected number of pull operations occurring during the operation *hold* (X) is less than $C_2(X)/m$).

The bound given by (9) is plotted in Fig. 7 as a function of n, together with observations of $E[C(X)]$ for the example distributions. Distributions 1, 2 and 5 come closest to the bound, which in these cases is quite good.

The methods of this paper have not proven applicable to the development of lower bounds, principally because Lemma 4 has no converse. However, the standard

argument involving entropy [13] can be adapted to produce a distribution-dependent, $O(\log(n))$ lower bound on $E[C(X)]$ [12].

Although the possibility of Henriksen's algorithm being $O(\log(n))$ remains open, the author conjectures that it is not. The low proportionality constants account sufficiently for the logarithmic appearance of test runs, and individual $O(n^{1/2})$ sequences of hold operations are not difficult to discover.

To conclude, we point to several advantages that Henriksen's algorithm has over other event list algorithms.

First, it is simple, not just because of its modest code length, but because there are no difficult dynamic adjustments to be made (as in the Indexed List algorithm [16]), and because the notice ordering is made explicit by the basic linked list underlying the structure.

Second, according to McCormack and Sargent's empirical study [14], Henriksen's algorithm is fast, even for large n, and by (9) this is true for a wide range of simulations. In addition, the algorithm will run very fast on some distributions (Fig. 7).

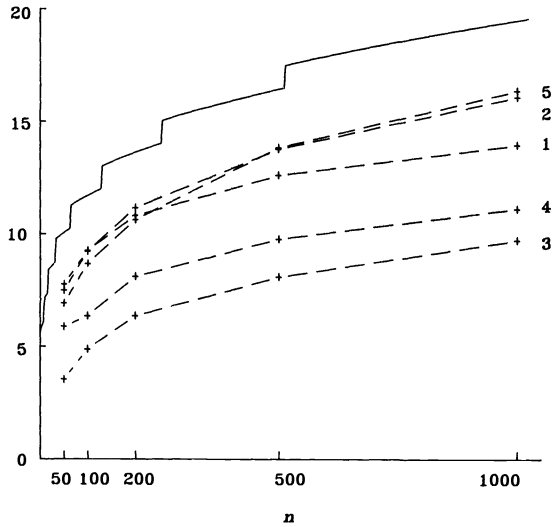


FIG. 7. *The distribution-independent bound on* $E[C(X)]$ *given by* (9) *as a function of* n, *with observations of* $E[C(X)]$ *for the five example distributions.* $m = 4$.

Appendix 1. We complete the proof of Theorem 1. Let

$$z = \sum_{i=h+1}^{v} \sum_{j=h+1}^{i} \sum_{k_0=1}^{\infty} \cdots \sum_{k_{v-1}=1}^{\infty} (k_j - m) \max(k_i - (i-h)m, 0) p_{k_0} \cdots p_{k_{v-1}}.$$

We are to prove

$$z \geqq \frac{1}{6m^2}[E[S_h + 2m]E[(S_h - m)^3] + 3m^2 E[(S_h - m)^2] - m^2 E[S_h]E[S_h - m]].$$

Divide the sum z into two parts z_1 (for the case $j = i$) and z_2 (for $j \neq i$):

$$z_1 = \sum_{i=h+1}^{v} \sum_{k_0=1}^{\infty} \cdots \sum_{k_{v-1}=1}^{\infty} (k_i - m) \max(k_i - (i-h)m, 0) p_{k_0} \cdots p_{k_{v-1}}$$

$$= \sum_{i=h+1}^{v} \sum_{k_i=1}^{\infty} (k_i - m) \max(k_i - (i-h)m, 0) p_{k_i},$$

$$z_2 = \sum_{i=h+1}^{v} \sum_{j=h+1}^{i-1} \sum_{k_0=1}^{\infty} \cdots \sum_{k_{v-1}=1}^{\infty} (k_j - m) \max (k_i - (i-h)m, 0) p_{k_0} \cdots p_{k_{v-1}}$$

$$= \sum_{i=h+1}^{v} \sum_{j=h+1}^{i-1} \sum_{k_i=1}^{\infty} \sum_{k_j=1}^{\infty} (k_j - m) \max (k_i - (i-h)m, 0) p_{k_i} p_{k_j}$$

$$= \sum_{i=h+1}^{v} \sum_{j=h+1}^{i-1} \sum_{k_i=1}^{\infty} E[S_j - m] \max (k_i - (i-h)m, 0) p_{k_i}$$

$$= \sum_{i=h+1}^{v} \sum_{k_i=1}^{\infty} (i-h-1) E[S_h - m] \max (k_i - (i-h)m, 0) p_{k_i}$$

since $E[S_j - m] = E[S_h - m]$ (I_j is in the neighbourhood of I_h). Now

$$z = z_1 + z_2 = \sum_{i=h+1}^{v} \sum_{k_i=1}^{\infty} [k_i - m + (i-h-1)E[S_h - m]] \max (k_i - (i-h)m, 0) p_{k_i}$$

$$= \sum_{i=1}^{v-h} \sum_{j=1}^{\infty} [j - m + (i-1)E[S_h - m]] \max (j - im, 0) p_j$$

since $\max (k_i - (i-h)m, 0)$ is nonzero only near I_h, and the S_i are identically distributed in this neighbourhood;

$$= \sum_{i=1}^{v-h} \sum_{j=im+1}^{\infty} [j - m + (i-1)E[S_h - m]](j - im) p_j$$

$$= \sum_{i=1}^{v-h} \sum_{j=1}^{\infty} [j + im - m + (i-1)E[S_h - m]] j p_{j+im}$$

$$\simeq \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} [j^2 + j(i-1)E[S_h]] p_{j+im};$$

now using the axis transformation $(i, j) \to (u, v, w)$, where $i = w$, $j = m(u-w) + v$,

$$= \sum_{u=1}^{\infty} \sum_{v=1}^{m} \sum_{w=1}^{u} [[m(u-w) + v]^2 + [m(u-w) + v][w-1]E[S_h]] p_{mu+v}.$$

With some effort we can show, letting $\alpha = mu + v$, that

$$\sum_{w=1}^{u} [m(u-w) + v]^2 \geqq \frac{1}{6m^2} [2m(\alpha - m)^3 + 3m^2(\alpha - m)^2],$$

and

$$\sum_{w=1}^{u} [m(u-w) + v](w-1)E[S_h] \geqq \frac{E[S_h]}{6m^2} [(\alpha - m)^3 - m^2(\alpha - m)]$$

(these inequalities are also approximate equalities). Therefore

$$z \geqq \frac{1}{6m^2} \sum_{u=1}^{\infty} \sum_{v=1}^{m} [2m(\alpha - m)^3 + 3m^2(\alpha - m)^2 + E[S_h][(\alpha - m)^3 - m^2(\alpha - m)]] p_{mu+v}$$

$$= \frac{1}{6m^2} \sum_{\alpha=m+1}^{\infty} [E[S_h + 2m](\alpha - m)^3 + 3m^2(\alpha - m)^2 - m^2 E[S_h](\alpha - m)] p_\alpha$$

$$= \frac{1}{6m^2} [E[S_h + 2m]E[(S_h - m)^3] + 3m^2 E[(S_h - m)^2] - m^2 E[S_h]E[S_h - m]].$$

Acknowledgments. Thanks go to Drs. Allan G. Bromley and Norman Y. Foo for many valuable discussions, and to the anonymous referee for many improvements in presentation.

REFERENCES

[1] JOHN H. BLACKSTONE, GARY L. HOGG AND DON T. PHILLIPS, *A two-list synchronizaton procedure for discrete event simulation*, Comm. ACM, 24 (1981), pp. 825-829.

[2] R. COURANT AND F. JOHN, *Introduction to Calculus and Analysis, Vol.* 2, John Wiley, New York, 1974.

[3] D. R. COX, *Renewal Theory*, Methuen, 1962.

[4] D. DAVEY AND J. G. VAUCHER, *Self-optimizing partitioned sequencing sets for discrete event simulation*, INFOR, 18 (1980), pp. 41-61.

[5] R. ENGELBRECHT-WIGGANS AND W. L. MAXWELL, *Analysis of the time indexed list procedure for synchronization of discrete event simulations*, Management Sci., 24 (1978), pp. 1417-1427.

[6] W. R. FRANTA AND K. MALY, *An efficient data structure for the simulation event set*, Comm. ACM, 20 (1977), pp. 596-602.

[7] G. H. GONNET, *Heaps applied to event driven mechanisms*, Comm. ACM, 19 (1976), pp. 417-418.

[8] J. O. HENRIKSEN, *An improved events list algorithm*, Proc. Winter Simulation Conference, December 1977, pp. 547-557.

[9] ———, *Event list management—a tutorial*, Proc. Winter Simulation Conference, 1983, pp. 543-551.

[10] ARNE JONASSEN AND OLE-JOHAN DAHL, *Analysis of an algorithm for priority queue administration*, BIT, 15 (1975), pp. 409-422.

[11] J. H. KINGSTON, *Analysis of tree algorithms for the simulation event list*, Acta Informatica, 22 (1985), pp. 15-33.

[12] ———, *Analysis of algorithms for the simulation event list*, Ph.D. thesis, Basser Dept. of Computer Science, University of Sydney, July 1984.

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

[14] W. M. MCCORMACK AND R. G. SARGENT, *Analysis of future event set algorithms for discrete event simulation*, Comm. ACM, 24 (1981), pp. 801-812.

[15] E. D. ULRICH, *Event manipulation for discrete simulations requiring large numbers of events*, Comm. ACM, 21 (1978), pp. 777-785.

[16] J. G. VAUCHER AND P. DUVAL, *A comparison of simulation event list algorithms*, Comm. ACM, 18 (1975), pp. 223-230.

[17] J. G. VAUCHER, *On the distribution of event times for the notices in a simulation event list*, INFOR, 15 (1977), pp. 171-182.

[18] F. P. WYMAN, *Improved event-scanning mechanisms for discrete event simulation*, Comm. ACM, 18 (1975), pp. 350-353.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986

© 1986 Society for Industrial and Applied Mathematics

THE RISCH DIFFERENTIAL EQUATION PROBLEM*

J. H. DAVENPORT†

Abstract. We propose a new algorithm, similar to Hermite's method for the integration of rational functions, for the resolution of Risch differential equations in closed form, or proving that they have no resolution. By requiring more of the presentation of our differential fields (in particular that the exponentials be weakly normalised), we can avoid the introduction of arbitrary constants which have to be solved for later.

We also define a class of fields known as exponentially reduced, and show that solutions of Risch differential equations which arise from integrating in these fields satisfy the "natural" degree constraints in their main variables, and we conjecture (after Risch and Norman) that this is true in all variables.

Key words. symbolic integration, structure theorems, differential algebra

AMS(MOS) subject classifications. Primary 68C20; secondary 12H20

1. Introduction. In his major paper on the integration of functions in closed form, Risch [1969] showed that the integration of functions involving exponentials was intimately linked with the resolution of ordinary differential equations of the following type:

$$y' + fy = g, \tag{1}$$

(henceforth called "Risch differential equations"), where f and g are known and lie in a given differential field F, and the problem is to determine y in that differential field satisfying the equation. For example, the well-known unintegrability of e^{-x^2} is equivalent to the insolubility of $y' - 2xy = 1$ for rational functions y, and the fact that $\int 2x e^{-x^2} = -e^{-x^2}$ is equivalent to the fact that -1 is a solution to $y' - 2xy = 2x$. We note that, if $\exp(-\int f)$ does not belong to F, there can be at most one solution to this equation. This condition is satisfied for the equations that arise from the integration of $g \exp(\int f)$, and one of our concerns will be to ensure that it remains true throughout the proof. Our proof, like Risch's [1969], will proceed by induction on the structure of F, which we allow to be a transcendental extension of the rational functions by a series of logarithms and exponentials. We note that the question of algebraic functions is treated, by a slight generalisation of Risch's argument, in Davenport [1984].

In order for the induction argument in Risch [1969] to work, Risch had to consider a more general form, viz.

$$y' + fy = \sum_{i=1}^{m} c_i g_i, \tag{2}$$

where the g_i are members of F and the c_i are unknown constants. We do not need this complication for our recursion, and so we will restrict ourselves to equations of the form (1). Should it be necessary to consider equations of the form (2) (as in Cherry [1983]), the mechanisms developed here can be generalised readily to such equations.

Our approach is similar to Risch's [1969], inasmuch as it is recursive in the structure of the differential field, regarding y as a rational function in its principal

* Received by the editors September 1, 1983, and in revised form June 12, 1985.

† School of Mathematics, University of Bath, Bath BA2 7AY, England.

indeterminate ϑ, whose coefficients are determined by recursion. The simplest way of expressing the difference between his methods and ours is to look at the various canonical representations for rational functions, where for definiteness we consider a rational function $f(x) \in F(x)$, the field of rational functions of x with coefficients in F. Risch [1969] chooses essentially the simplest

$$f(x) = \frac{f_n(x)}{f_d(x)},$$

where f_n and f_d are relatively prime members of $F[x]$, the ring of polynomials with coefficients in F. If one can determine y_d, then (1) is reduced to a purely polynomial problem, and f_n can be determined.

We choose one of the following two representations (3) and (4) (depending on the differential nature of x). The first representation is

$$f(x) = f_0 + f_{p+}(x) + \frac{f_n(x)}{f_d(x)}, \tag{3}$$

where f_0, (known as the "x-free"[1] part of f) belongs to F,

$$f_{p+}(x) = \sum_{i=1}^{m} f_i x^i$$

is the "polynomial part" of f, and f_n and f_d are relatively prime elements of $F[x]$ with the additional constraint that degree $(f_n) <$ degree (f_d). f_n/f_d is then a proper fraction, known as the "fractional part" of f, and sometimes written f_f. The second representation is similar, being

$$f(x) = f_0 + f_{p-}(x) + f_{p+}(x) + \frac{f_n(x)}{f_d(x)}, \tag{4}$$

where in addition to the definitions above, we have that

$$f_{p-}(x) = \sum_{i=-n}^{-1} f_i x^i$$

is the "negative polynomial part" of f, and that x does not divide f_d. We will refer to n as the degree of f_{p-}. f_n/f_d is still a proper fraction, known as the "fractional part" of f, and again may written f_f. We note that decompositions of type (3) add, in the sense that, if $a = b + c$, then $a_f = b_f + c_f$ etc., and the same is true for decompositions of type (4). It does not make sense to mix the two types.

Our overall strategy will be to express each of f, g and y in (1) in representation (3) (or representation (4) if x is an exponential), and then to determine:

a) the partial fraction decomposition of y_f by considering the partial fraction decompositions of f_f and g_f;

b) y_{p+} by considering f_{p+} and g_{p+};

c) (if x is exponential) y_{p-} by considering f_{p-} and g_{p-};

d) (if necessary) y_0 by recursion on f_0 and g_0.

[1] The word "constant" has a different meaning in differential algebra, and hence we use this, somewhat clumsy, locution.

Another way of looking at the difference between our method and Risch's is to observe that, if $f=0$, (1) reduces to the problem $y=\int g$. In the special case that F is just a field of rational functions, Risch's method reduces to solving a set of linear equations for the numerator of $\int g$, while ours reduces to Hermite's [1872] method of integration via partial fractions.

2. Degree bounds. One of the major problems of Risch differential equations is that of bounding the size of the solution. To see that this is nontrivial, consider the equation

$$y'+\left(1+\frac{5}{x}\right)y=1. \tag{5}$$

The solution to this equation is

$$y=\frac{x^5-5x^4+20x^3-60x^2+120x-120}{x^5}.$$

However, this equation is not a particularly useful one to solve, since it arises from the integration problem $\int e^{x+5\log x}$, which might be more clearly expressed as $\int x^5e^x$. Not only is the solution to (5) much larger than we would expect, it has a denominator even though the right-hand side of (5) does not.

In order to obviate these problems, we shall only consider a subset of the Risch differential equations—those in which f is weakly normalised. An element f of a differential field $D(\varphi)$ is said to be *weakly normalised* with respect to φ if:

a) it has an integral which is elementary over $D(\varphi)$;

b) no logarithm whose argument depends on φ occurs linearly in it with a positive integer coefficient, where the integral is written so that the argument of any logarithm not in $D(\varphi)$ is a square-free polynomial (with respect to φ, and with coefficients belonging to D);

c) the sum of the coefficients of the logarithms occurring linearly in the integral, weighted by the degrees in φ of their arguments, is not a negative integer.

Part (b) is weaker than Cherry's [1983] definition of normalised, which also rules out fractional coefficients greater than 1. Even our definition is in fact slightly stronger than we really need, but the precise condition is fairly messy to formulate. Part (c) is somewhat bizarre, but is equivalent to saying that, if the integral is written in terms of $\hat{\varphi}=1/\varphi$, then $\log\hat{\varphi}$ does not occur with a positive integer coefficient. This substitution is used in Davenport [1984] to give an algorithm for Risch differential equations in algebraic extensions of $K[z]$.

The implications of condition (c) are somewhat complex. Suppose that φ is a *primitive* over D, i.e. that $\varphi'\in D$. Given an element f of $D(\varphi)$, f_f is a proper fraction: $f_f=p/q\in D(\varphi)$, where $p=\sum_{j=0}^{m}p_j\varphi^j$ and $q=\sum_{j=0}^{n}q_j\varphi^j$. We write f_∞ for p_{n-1}/q_n, noting that this is zero if $m<n-1$. We observe that this too is additive: $(f+g)_\infty=f_\infty+g_\infty$. Furthermore, if f is a proper fraction, $(f')_\infty=(f_\infty)'$.

LEMMA 2.1. *If f is weakly normalised with respect to a primitive φ, then $\int f_\infty$ can not be written in the form $g+N\varphi$, where $g\in D$ and N is a negative integer.*

Proof. By Liouville's theorem (Risch [1969]), we can write

$$\int f=h_{p+}+h_0+\frac{h_n}{h_d}+\sum_{i=1}^{k}c_i\log p_i(\varphi)+\sum_{i=k+1}^{l}c_i\log p_i,$$

where the c_i are constants, the $p_i(\varphi)(1 \leqq i \leqq k)$ are monic polynomials of degree n_i in $D[\varphi]$, and the $p_i(k < i \leqq l)$ are elements of D. Differentiating, and decomposing according to (3),

$$f_f = (h_f)' + \sum_{i=1}^{k} \frac{c_i p_i'}{p_i}.$$

Now, since p_i is a monic polynomial in φ of degree n_i, $(p_i'/p_i)_\infty = n_i\varphi' + p'_{i,n_i-1}$, where $p_i = \varphi^{n_i} + \sum_{j=1}^{n_i-1} p_{i,j}\varphi^j$. Therefore

$$f_\infty = (h_\infty)' + \left(\sum_{i=1}^{k} c_i n_i\right)\varphi' + \left(\sum_{i=1}^{k} c_i p_{i,n_i-1}\right)',$$

i.e.

$$\int f_\infty = c + h_\infty + \sum_{i=1}^{k} c_i p_{i,n_i-1} + \left(\sum_{i=1}^{k} c_i n_i\right)\varphi,$$

where c is a constant of integration, h_∞ and $p_{i,n_i-1} \in D$, and condition (c) says that the term in parentheses is not a negative integer.

From the point of view of integrating in terms of elementary functions, there is no real problem with the restriction to weakly normalised f. A Risch differential equation (1) arises from the integration problem $\int g\vartheta^n$, where $\vartheta = e^h$, and n is an integer (which may be negative). This implies that $f = nh'$, thus automatically satisfying part (a) of the definition of weakly normalised. If part (b) is not satisfied, so that nh contains a summand of the form $k \log \eta$, then we rewrite the original integration problem as $\int g\eta^k\hat{\vartheta}$, where $\hat{\vartheta} = e^{nh - k\log\eta}$. If part (c) is not satisfied, life is not quite so simple, as we see from the example of $\exp(-\frac{1}{2}\log(x^2+1) + 1/x)$, where bringing the $-\frac{1}{2}\log(x^2+1)$ out of the exponential means that the integrand is no longer expressed in a purely transcendental fashion, since we have generated a $\sqrt{x^2+1}$. However, we can transform the exponential into

$$\frac{\exp(\frac{1}{2}\log(x^2+1) + 1/x)}{x^2+1},$$

which is weakly normalised and expressed purely in terms of transcendental functions.

3. Statement of results. This paper will prove results equivalent to part (b) of the Main Theorem of Risch [1969].

MAIN THEOREM. *Let $F = K(z, \vartheta_1, \cdots, \vartheta_n)$, where K is a field of constants which is finitely generated*[2] *over* $\mathbf{Q}$, *z is transcendental over K and a solution of $z' = 1$, each ϑ_i a monomial over $K(z, \vartheta_1, \cdots, \vartheta_{i-1})$.*

(a) *Let $f \in F$. Then one can determine in a finite number of steps whether there are $v_0 \in F$, $v_i \in \bar{K}F$, $i = 1, \cdots, m$ and $c_1, \cdots, c_m$ in $\bar{K}$ such that*

$$f = \left(v_0 + \sum_{i=1}^{m} c_i \log v_i\right)'.$$

If they do exist we can find them.

(b) *Let f, g_i $i = 1, \cdots, m$ be elements of F. Then one can find, in a finite number of steps, $h_1, \cdots, h_r$ in F and a simultaneous set S of linear algebraic equations in $m + r$*

[2] This is Risch's original wording. In fact one needs the stipulation "explicitly finitely generated" for the theorem to be true (Davenport and Trager [1981]).

variables, with coefficients in K, such that (2) *holds for* $y \in F$ *and* c_i *elements of K iff* $y = \sum_{i=1}^{r} y_i h_i$ *where* y_i *are elements of K and* $c_1, \cdots, c_m, y_1, \cdots, y_r$ *satisfy S.*

We refer to (a) as being able to solve the *integration problem* for F, and to the variant of (b) which results from using (1) instead of (2), viz. being able to determine if there is a y such that $y' + fy = g$, as being able to solve the *Risch o.d.e. problem* for F.

We shall prove the following three theorems, which correspond to the base case part (b) of the induction in the Main Theorem, and to the cases ϑ_n logarithmic and ϑ_n exponential in the inductive part (b) argument. First, however, we need some notation and conventions. Throughout this paper we assume that D is a differential field of characteristic 0, explicitly finitely generated over $\mathbf{Q}$. This implies that all the usual polynomial and rational function operations, including the taking of greatest common divisors, can be performed over finite extensions of D. Since D is explicitly finitely generated, factoring of polynomials can also be performed (Davenport and Trager [1981]). We always use $'$ for the differentiation operator of D. We assume that K is the field of constants of D, that is the set of elements k such that $k' = 0$. The elements of K are called *constants*. We say that ϑ is a *monomial* over D if ϑ is transcendental over D, $D(\vartheta)$ and D have the same field of constants, and either:

a) there is a nonzero f in D such that $\vartheta' = f'/f$ (informally, $\vartheta = \log f$, but this notation can lead to problems of choice of constant of integration etc.—see the discussion in Risch [1969]);

b) there is an f in D such that $\vartheta' = f'\vartheta$ (informally, $\vartheta = \exp f$, but this notation can equally lead to problems).

We shall often ask questions of the form "Is $\exp h$ a member of D?" or "What is the representation of h as a member of $D(\vartheta)$?". Such questions can be answered, if D is a field of elementary functions with K as its field of constants, as a consequence of the Structure Theorems (Rosenlicht [1976], Caviness [1977]).

THEOREM 1. *Let z be transcendental over K and a solution to* $z' = 1$, *f a weakly normalised nonzero element of* $K(z)$, *and g an element of* $K(z)$. *Then there is an algorithm to determine if there is a y in* $K(z)$ *such that* $y' + fy = g$. *If such a y exists, the algorithm finds it.*

This theorem says that we can solve the Risch o.d.e. problem for $K(z)$.

THEOREM 2. *Let D be a differential field such that there are algorithms for integration and the solution of Risch differential equations over D,* ϑ *a logarithmic monomial over D, f a weakly normalised nonzero element of* $D(\vartheta)$, *and g an element of* $D(\vartheta)$. *Then there is an algorithm to determine if there exists a y in* $D(\vartheta)$ *such that* $y' + fy = g$. *If such a y exists, the algorithm finds it.*

THEOREM 3. *Let D be a differential field such that there are algorithms for integration and the solution of Risch differential equations over D,* ϑ *an exponential monomial over D, f a weakly normalised nonzero element of* $D(\vartheta)$, *and g an element of* $D(\vartheta)$. *Then there is an algorithm to determine if there exists a y in* $D(\vartheta)$ *such that* $y' + fy = g$. *If such a y exists, the algorithm finds it.*

4. Proof of Theorem 1. Throughout this section, f, y and g are assumed to be decomposed according to (3). The proof proceeds via a series of reductions until we end up with an equation of the form (1) where we know that y has to be a constant. If f/g is not a constant, we can then deduce that the Risch differential equation was not soluble. Such an example occurs in $\int e^{x^2}$, which gives us the Risch equation $y' + 2xy = 1$, to be solved for $y \in K(x)$. We may also remark that taking traces shows that there is no advantage in working with an algebraic extension of K.

LEMMA 4.1. *Let p be an irreducible polynomial of $K[z]$. If p divides neither f_d nor g_d, then p does not divide y_d.*

Proof. Suppose[3] $p^n \| y_d$. It then follows that p^{n+1} divides the denominator of y', while at most p^n divides the denominator of fy. Hence p^{n+1} divides the denominator of $y'+fy$, contradicting the fact that p does not divide g_d.

The next result lets us deal with the "easy" part of g_d, that part which does not have a factor in common with f_d.

LEMMA 4.2. *For a Risch differential equation* (1) *over $K(z)$, either there is a substitution $y = \hat{y} + Y$ which reduces it to $\hat{y}' + f\hat{y} = \hat{g}$, where $\hat{g}_d$ contains no irreducible factor that does not divide f_d, or the equation is insoluble. Furthermore, given f and g, we can determine which and calculate Y if it exists.*

Proof. Write $g_d = pq$, where p is the product of all the irreducible factors of g_d that divide f_d, and q is the product of all the irreducible factors of g_d that do not. We can then perform a partial fraction decomposition, writing

$$\frac{g_n}{g_d} = \frac{a}{p} + \frac{b}{q}.$$

We proceed much as in Hermite integration of rational functions, performing a square-free decomposition of q,

$$q = \prod_{i=1}^{n} q_i^i$$

where the q_i are square-free and relatively prime, and a partial fraction decomposition:

$$\frac{b}{q} = \sum_{i=1}^{n} \sum_{j=1}^{i} \frac{b_{i,j}}{q_i^j}$$

where $\deg(b_{i,j}) < \deg(q_i)$.

We can now reduce the problem to one with a smaller value of n (if $n > 1$). The equation $Rq_n + Sq_n' = b_{n,n}$ is uniquely soluble for R, $S \in K[z]$ with $\deg(S) < \deg(q_n)$. If we write $Y = -S/(n-1)q_n^{n-1}$, we note that one of the terms from Y' cancels the $b_{n,n}/q_n^n$ term in g, that the rest of the terms from Y' have a denominator of q_n^{n-1}, and that the partial fraction decomposition of fY contains only the denominators f_d and q_n^{n-1}. Hence the g term in the equation resulting from the substitution $y = \hat{y} + Y$ has at most q_n^{n-1} in the denominator.

A sequence of such substitutions (which can be merged into one substitution) reduces the equation to one in which the irreducible factors of g_d which do not divide f_d occur only linearly in g_d. We now assert that, if any such factors are left, the equation is insoluble. But this is evident, since if y_d does not contain a factor of p (being an irreducible factor of g_d not dividing f_d), the denominator of $y'+fy$ does not contain such a factor, while if $p^n \| y_d$ then the denominator of $y'+fy$ has a factor of p^{n+1}. In neither case can a factor of multiplicity one occur.

LEMMA 4.3. *For a Risch differential equation* (1) *over $K(z)$ such that f is weakly normalised and every irreducible factor of g_d divides f_d, there is a substitution $y = \hat{y} + Y$ which reduces it to $\hat{y}' + f\hat{y} = \hat{g}$, where every irreducible factor of $\hat{g}_d$ occurs no more often in $\hat{g}_d$ than it does in f_d. Furthermore, given f and g, we can calculate Y.*

[3] This notation means that p^n divides y_d, but that p^{n+1} does not.

Proof. Let p be a square-free element of $K[z]$ such that $p^\beta \| f_d$, $p^\gamma \| g_d$ and such that f_d/p^β and g_d/p^γ are relatively prime to p. From the hypotheses of the lemma, we know that $\beta = 0$ implies $\gamma = 0$, and Lemma 4.1 then implies that p does not divide y_d. Suppose, therefore, that $\beta > 0$. If $\gamma \leqq \beta$, then the irreducible factors of p already occur no more often in g_d than they do in f_d, and no reduction is necessary as far as they are concerned. Hence we are faced with the problem of reducing the multiplicity γ of p in g_d until $\gamma \leqq \beta$. We will write α for the multiplicity of p in y_d.

CASE 1. $\beta > 1$. Construct partial fraction decompositions of f_f, g_f and y_f, such that the most negative terms with respect to p are f_β/p^β, g_γ/p^γ and y_α/p^α, where f_β, g_γ and y_α are elements of $K[z]$ of degree less than that of p. Furthermore, f_β is coprime to p by the choice of p. The partial fraction decomposition of $y' + fy$ contains many terms, but the ones with the greatest multiplicity of p in the denominator are $-\alpha y_\alpha p'/p^{\alpha+1}$ and $f_\beta y_\alpha/p^{\alpha+\beta}$. Since $\beta > 1$, the latter term dominates, and must be equal to the term with the greatest multiplicity of p in the denominator on the right-hand side of (1), viz. g_γ/p^γ.

Hence $\alpha + \beta = \gamma$ and $f_\beta y_\alpha \equiv g_\gamma$ (modulo p). This equation can be solved for y_α, and then the substitution $y = \hat{y} + y_\alpha/p^\alpha$ decreases the multiplicity of p in g_d.

CASE 2. $\beta = 1$. We perform the same partial fraction decompositions as in Case 1, but this time the principal terms arising from y' and fy have the same multiplicity, so that the principal term on the left-hand side of (1) appears to be $y_\alpha(-\alpha p' + f_1)/p^{\alpha+1}$. This will actually be the principal term, unless cancellation occurs. But cancellation can only occur if there is a common factor q between p and $-\alpha p' + f_1$.

Suppose, if possible, that this occurs, and write $p = qr$ (q and r are relatively prime since p is square-free, and can be assumed monic) and $f_1 = aq + br$ (for minimal a and b). We thus have $r | -\alpha(qr' + rq') + aq + br$, which implies that $r | q(-\alpha r' + a)$. Hence $r | -\alpha r' + a$, and the only polynomial of degree less than that of r divisible by r is zero. But the partial fraction decomposition of f includes the term $f_1/p = a/r + b/q$, so that $\int f$ includes the term $\int a/r = \int \alpha r'/r = \alpha \log r$. α has to be a positive integer, and this contradicts the assumption that f was weakly normalised, and hence cancellation cannot occur.

We can thus equate the principal terms on the two sides of (1) and deduce that $\alpha = \gamma - 1$ and that $g_\gamma \equiv (-\alpha p' + f_1) y_\alpha$ (modulo p). This equation can be solved for y_α, and, as in Case 1, we can reduce γ.

LEMMA 4.4. *If a Risch differential equation satisfying the conclusion of Lemma* 4.3 *has a solution, then that solution is a polynomial in* $K[z]$, *i.e.* $y_f = 0$.

Proof. Suppose $y_f \neq 0$. We know from Lemma 4.1 that the only factors of y_d are factors of f_d. If p is an irreducible polynomial such that $p^\alpha \| y_d$, $p^\beta \| f_d$, then the argument of Lemma 4.3 shows that $p^{\alpha+\beta} \| g_d$, which does not satisfy the conclusion of Lemma 4.3.

We note that equation (5), which satisfies all the conclusions of Lemma 4.3 except for f being weakly normalised, but whose solution is not a polynomial, shows that this last condition is necessary.

LEMMA 4.5. *If* $f_{p+} + f_0$ *is nonzero, then there is an algorithm for determining if any Risch differential equation over* $K(z)$ *satisfying the conclusions of Lemma* 4.3 *has a solution, and finding the solution, if any.*

Proof. Lemma 4.4 implies that the solution must be a polynomial. Clearly $y = 0$ if, and only if, $g = 0$, so we may suppose that both y and g are nonzero. Write α for the degree of y, if it exists, β for the degree of $f_{p+} + f_0$ and γ for the degree of $g_{p+} + g_0$ (-1 for the degree of 0). Write y_α, f_β and g_γ for the leading coefficients of these quantities. Then the leading term of y' is $\alpha y_\alpha z^{\alpha-1}$, that of the polynomial part of fy is $f_\beta y_\alpha z^{\alpha+\beta}$ and the leading term of g_{p+} is $g_\gamma z^\gamma$. If $\gamma < \beta$ then there is no solution, otherwise

the substitution $y = \hat{y} + (g_\gamma/f_\beta)z^{\gamma-\beta}$ reduces the degree of $g_{p+} + g_0$. The conclusions of Lemma 4.3 are still satisfied, so we continue this reduction until either $g = 0$, when we have a solution, or $\gamma < \beta$, when we have a proof of unsatisfiability.

LEMMA 4.6. *If* $f_{p+} + f_0 = 0$ *and* degree $(f_n) <$ degree $(f_d) - 1$, *then there is an algorithm for determining if any Risch differential equation over* $K(z)$ *satisfying the conclusions of Lemma* 4.3 *has a solution, and finding the solution, if any.*

Proof. Lemma 4.4 implies that the solution must be a polynomial. Suppose that $y_{p+} \neq 0$, and let its most significant term have degree α. Then the most significant term of $(y')_{p+}$ has degree $\alpha - 1$, and the most significant term of $(f_y)_{p+}$ has degree at most $\alpha - 2$, by the degree condition in the hypothesis. If the degree of g_{p+} is γ, this tells us that $\alpha = \gamma + 1$, and that $\alpha y_\alpha = g_\gamma$, where y_α and g_γ are the leading coefficients of y and g_{p+}. Repating this process allows us to find the whole of y_{p+}.

Having found y_{p+}, all that is possible is that y should be a constant, y_0. If g/f is a constant, this is then y_0, otherwise we deduce that the equation is insoluble.

LEMMA 4.7. *If* $f_{p+} + f_0 = 0$ *and* degree $(f_n) =$ degree $(f_d) - 1$, *then there is an algorithm for determining if any Risch differential equation satisfying the conclusions of Lemma* 4.3 *has a solution, and finding the solution, if any.*

Proof. The reasoning is similar to that of Lemma 4.6, except that now the most significant terms of $(y')_{p+}$ and $(fy)_{p+}$ both have degree $\alpha - 1$, and hence there is a possibility of cancellation between them. If the leading coefficient of y_{p+} is y_α, then cancellation means that $\alpha y_\alpha + f_\infty y_\alpha = 0$, i.e. that $f_\infty = -\alpha$. By Lemma 2.1, though, f_∞ cannot be a negative integer, and hence cancellation cannot occur. Hence $\alpha = \gamma - 1$ and $(\alpha + f_\infty)y_\alpha = g_\gamma$. As above, we thus find the whole of y_p, and then $y_0 = g/f$, provided that is a constant.

The proof of Theorem 1 is then completed by observing that Lemmas 4.2 and 4.3 give the fractional part of y (or prove that the equation is insoluble), and then the appropriate one of Lemmas 4.5, 4.6, 4.7 gives $y_{p+} + y_0$ (or proves that the equation is insoluble).

5. Proof of Theorem 2. In this section, we shall in fact prove a more general theorem, where we assume that ϑ is a *primitive monomial* over D, i.e. that ϑ is transcendental over D, that $D(\vartheta)$ and D have the same field of constants, and that $\vartheta' \in D$. This certainly includes logarithmic monomials as a special case, but we have to appeal to a more general Structure Theorem (Rothstein and Caviness [1979]) in order to answer questions about the representation of elements of $D(\vartheta)$, for which we will need to insist that $D(\vartheta)$ be *log-explicit* (see Rothstein and Caviness [1979] for details).

THEOREM 2'. *Let* D *be a differential field such that there are algorithms for integration and the solution of Risch differential equations over* D, ϑ *a primitive monomial over* D, f *a weakly normalised nonzero element of* $D(\vartheta)$, *and* g *an element of* $D(\vartheta)$. *Then there is an algorithm to determine if there exists a* y *in* $D(\vartheta)$ *such that* $y' + fy = g$. *If such a* y *exists, the algorithm finds it.*

Throughout this section, f, g and y, all of which are elements of $D(\vartheta)$, are assumed to be decomposed with respect to ϑ according to (3). The proof proceeds in a very similar manner to that of Theorem 1, aided by the following proposition.

PROPOSITION 1. *If* p *is a square-free polynomial in* $D[\vartheta]$, *then* p *and* p' *are relatively prime.*

A proof of this proposition can be found, for example, in Davenport [1983a, Proposition 2, p. 91] (the result is only stated for logarithmic monomials, but the

generalisation is immediate). We note, though, that this result is not the well-known characteristic of square-free polynomials, which says that p and $\partial p/\partial\vartheta$ are relatively prime.

The proofs of the following four lemmas are identical to the corresponding proofs in the previous section.

LEMMA 5.1. *Let p be an irreducible polynomial of $D[\vartheta]$. If p divides neither f_d nor g_d, then p does not divide y_d.*

LEMMA 5.2. *For a Risch differential equation* (1) *over $D(\vartheta)$, either there is a substitution $y=\hat{y}+Y$ which reduces it to $\hat{y}'+f\hat{y}=\hat{g}$, where $\hat{g}_d$ contains no irreducible factor that does not divide f_d, or the equation is insoluble. Furthermore, given f and g, we can determine which and calculate Y if it exists.*

LEMMA 5.3. *For a Risch differential equation* (1) *over $D(\vartheta)$ such that f is weakly normalised with respect to ϑ and every irreducible factor of g_d divides f_d, there is a substitution $y=\hat{y}+Y$ which reduces it to $\hat{y}'+f\hat{y}=\hat{g}$, where every irreducible factor of $\hat{g}_d$ occurs no more often in $\hat{g}_d$ than it does in f_d. Furthermore, given f and g, we can calculate Y.*

LEMMA 5.4. *If a Risch differential equation satisfying the conclusions of Lemma* 5.3 *has a solution, then that solution is a polynomial in $D[\vartheta]$, i.e. $y_f=0$.*

LEMMA 5.5. *If f_{p+} is nonzero, then there is an algorithm for determining if any Risch differential equation over $D(\vartheta)$ satisfying the conclusions of Lemma* 5.3 *has a solution, and finding the solution, if any.*

Proof. Lemma 5.4 implies that the solution must be a polynomial. Clearly $y=0$ if, and only if, $g=0$, so we may suppose that both y and g are nonzero. Write α for the degree of y, if it exists, β for the degree of f_{p+} and γ for the degree of $g_{p+}+g_0$ (-1 for the degree of 0). Write y_α, f_β and g_γ for the leading coefficients of these quantities. Then the leading term of y' is either $y'_\alpha\vartheta^\alpha$ or $\alpha y_\alpha\vartheta'\vartheta^{\alpha-1}$ (depending on whether or not y_α is a constant), that of the polynomial part of fy is $f_\beta y_\alpha\vartheta^{\alpha+\beta}$ and the leading term of g_{p+} is $g_\gamma\vartheta^\gamma$. If $\gamma<\beta$ then there is no solution, otherwise the substitution $y=\hat{y}+(g_\gamma/f_\beta)\vartheta^{\gamma-\beta}$ reduces the degree of $g_{p+}+g_0$. The conclusions of Lemma 5.3 are still satisfied, so we continue this reduction until either $g=0$, when we have a solution, or $\gamma<\beta$, when we have a proof of unsatisfiability.

LEMMA 5.6. *If $f_{p+}+f_0=0$, there is an algorithm for determining if any Risch differential equation satisfying the conclusions of Lemma* 5.3 *has a solution, and finding the solution, if any.*

Proof. Lemma 5.4 implies that the solution must be a polynomial, which we shall write as $\sum_{i=0}^{\alpha} y_i\vartheta^i$. We shall write γ for the degree of $g_{p+}+g_0$, and $g_{p+}+g_0=\sum_{i=0}^{\gamma} g_i\vartheta^i$. The degree of y' is either α or $\alpha-1$ (depending on whether or not y_α is a constant), while the degree of $(fy)_{p+}$ is at most $\alpha-1$. We must first demonstrate that cancellation does not occur between these two terms.

Suppose, therefore, that cancellation does occur, viz. that $\alpha-1>\gamma$. This implies that y_α is a constant. f is, by hypothesis, a proper fraction, which means that ϑf can be written as $f_\infty+P/Q$, where $f_\infty\in D$ and P/Q is a proper fraction in $D(\vartheta)$. Then the coefficient of $\vartheta^{\alpha-1}$ in $y'+fy$ is $\alpha y_\alpha\vartheta'+y'_{\alpha-1}+f_\infty y_\alpha$. If this is to vanish, we must have $y_{\alpha-1}=c-y_\alpha\int(\alpha\vartheta'+f_\infty)$, where c is a constant of integration. Lemma 2.1 implies that f_∞ cannot annihilate the $\alpha\vartheta'$ term, and this contradicts the requirement that $y_{\alpha-1}\in D$.

Hence $\alpha=\gamma$ or $\alpha=\gamma+1$. Comparing coefficients of ϑ_α leads to the equation $y'_\alpha=g_\alpha$. We assumed that an integration algorithm was available in D, so y_α is determined up to a constant of integration, say z_α. Comparing coefficients of $\vartheta_{\alpha-1}$, we

see that

$$y'_{\alpha-1}+\alpha\vartheta'\left(z_a+\int g_\alpha\right)+f_\infty\left(z_\alpha+\int g_\alpha\right)=g_{\alpha-1}.$$

Hence

$$y_{\alpha-1}=\int h-z_\alpha\int(\alpha\vartheta'+f_\infty),$$

where h is known. By Lemma 2.1, the second integral is possible, and contains ϑ with a nonzero coefficient. Since ϑ does not occur in $y_{\alpha-1}$, this requirement determines z_α. We note that, if the first integral is not possible, or generates new logarithms other than ϑ, then the Risch differential equation has no solution. We have now determined y_α completely, and $y_{\alpha-1}$ up to a constant of integration. Proceeding similarly, we determine the whole of y up to a constant of integration, so that $y=y_{\text{known}}+z$. z is determined by the requirement that $(y'+fy)_f=g_f$. Of course, if this does not yield a constant, then the equation is insoluble.

LEMMA 5.7. *If $f_{p+}=0$, but $f_0\neq 0$, then there is an algorithm for determining if any Risch differential equation satisfying the conclusions of Lemma* 5.3 *has a solution, and finding the solution, if any.*

Proof. Lemma 5.4 implies that the solution has to be a polynomial. Since f has an elementary integral, the analysis of Risch [1969, p. 182] shows that f_0 has an elementary integral, say h. The situation now depends on whether or not $\exp(h)\in D$.

CASE 1. $\exp(h)\notin D$. Suppose $y=\sum_{i=0}^{\alpha}y_i\vartheta^i$ and $g_{p+}+g_0=\sum_{i=0}^{\gamma}y_i\vartheta^i$. The leading term of the polynomial part of $y'+fy$ is (assuming no cancellation occurs), $(y'_\alpha+f_0y_\alpha)\vartheta^\alpha$. Cancellation implies that $y_\alpha=\exp(-\int f_0)$, which is impossible since $y_\alpha\in D$. Hence we do have the leading term, and so $\alpha=\gamma$ and

$$y'_\alpha+f_0y_\alpha=g_\alpha.$$

This is a Risch differential equation in D, and by hypothesis there is an algorithm for solving it (or proving it insoluble). Furthermore, that solution is unique. Having determined y_α, we can substitute $y=\hat{y}+y_\alpha\vartheta^\alpha$, and obtain an equation with the same f where the degree of $g_{p+}+g_0$ is smaller, and g_f still satisfies the conclusions of Lemma 5.3.

CASE 2. $\exp(h)\in D$. The analysis of the previous case no longer applies, not only because cancellation may occur, but also because the solutions to the Risch differential equations in D are no longer unique, and thus we have an undetermined multiple of $\exp(-h)$ in y_α, which we can only determine from the later equations. Fortunately, there is a trick that will get us out of this problem. Suppose that y is a solution to $y'+fy=g$. Substituting $y=\hat{y}\exp(-h)$ into this equation gives

$$\hat{y}'e^{-h}-h'\hat{y}e^{-h}+(f_0+f_f)\hat{y}e^{-h}=g.$$

But $h'=f_0$, so this simplifies to $\hat{y}'+f_f\hat{y}=g\exp(h)$. Conversely, any solution to this equation gives a solution to the original equation. Hence we make this transformation, apply Lemmas 5.1 to 5.3 and 5.6 to this equation to find a solution or prove that it does not have one, and then transform the solution back.

That these complications can arise is seen in the following example, where the Risch differential equation to be solved is

$$\frac{dy}{dx}+\left(-\frac{1}{x\log^2 x}-\frac{1}{x^2}\right)y=x(2\log^2 x-2)-\log^2 x+\frac{e^{-1/x}(2\log x-1)}{2x}+\log x+\frac{x}{\log x}.$$

Writing $\varphi = \log x$, this becomes

$$\frac{dy}{dx} + \left(-\frac{1}{x\varphi^2} - \frac{1}{x^2} \right) y = x(2\varphi^2 - 2) - \varphi^2 + \frac{e^{-1/x}(2\varphi - 1)}{2x} + \varphi + \frac{x}{\varphi}.$$

This equation arises from attempting to integrate

$$e^{1/\varphi + 1/x} \left(x(2\varphi^2 - 2) - \varphi^2 + \frac{e^{-1/x}(2\varphi - 1)}{2x} + \varphi + \frac{x}{\varphi} \right).$$

The equation already satisfies the conclusions of Lemma 5.3, and $f_{p+} = 0$, while $f_0 = -1/x^2$. D is the field $Q(x, e^{1/x})$, and, ignoring the fact that $e^{\int f_0} \in D$, we reduce our problem to the following:

$$\frac{dy_2}{dx} - \frac{y_2}{x^2} = 2x - 1.$$

The solution to this is

$$y_2 = c\, e^{-1/x} + x^2,$$

where c is a constant to be determined. The equation for y_1 enables us to determine that $c = \frac{1}{2}$, and the final solution is

$$y = \left(\frac{e^{-1/x}}{2} + x^2 \right) \log^2 x - x^2 \log (x).$$

Having proved these lemmas, we completed the proof of Theorem 2 by observing that Lemmas 5.2 and 5.3 give the fractional part of y (or prove that the equation is insoluble), and then the appropriate one of Lemmas 5.5, 5.6, 5.7 gives $y_{p+} + y_0$ (or proves that the equation is insoluble).

6. Proof of Theorem 3. Throughout this section, f, g and y, all of which are elements of $D(\vartheta)$, are assumed to be decomposed with respect to ϑ according to (4). We will find it useful to have a collective term for $y_{p+} + y_0 + y_{p-}$, and we will refer to this as a generalised polynomial. We shall write $\vartheta = \exp \eta$, where $\eta \in D$. The proof proceeds in a very similar manner to that of Theorem 2 except that the necessity of using decomposition (4) will force some additional complications, and again we are aided by the following proposition.

PROPOSITION 2. *If p is a square-free polynomial in $D[\vartheta]$ which is not divisible by ϑ, then p and p' are relatively prime.*

A proof of this proposition can be found, for example, in Davenport [1983a, Proposition 3, p. 96].

The proofs of the following four lemmas are similar to the corresponding proofs in the previous sections.

LEMMA 6.1. *Let p be an irreducible polynomial of $D[\vartheta]$, not divisible by ϑ. If p divides neither f_d nor g_d, then p does not divide y_d.*

LEMMA 6.2. *For a Risch differential equation (1) over $D(\vartheta)$, either there is a substitution $y = \hat{y} + Y$ which reduces it to $\hat{y}' + f\hat{y} = \hat{g}$, where $\hat{g}_d$ contains no irreducible factor that does not divide f_d, or the equation is insoluble. Furthermore, given f and g, we can determine which and calculate Y if it exists.*

LEMMA 6.3. *For a Risch differential equation (1) over $D(\vartheta)$ such that f is weakly normalised with respect to ϑ and every irreducible factor of g_d divides f_d, there is a*

substitution $y = \hat{y} + Y$ which reduces it to $\hat{y}' + f\hat{y} = \hat{g}$, where every irreducible factor of $\hat{g}_d$ occurs no more often in $\hat{g}_d$ than it does in f_d. Furthermore, given f and g, we can calculate Y.

Proof. Almost identical to Lemma 4.3. The difference is in Case 2, where we argued from $r|-\alpha r' + a$ that, as r' and a were of lower degree than r, then $a = \alpha r'$. That argument no longer holds, since r' is of the same degree as r, say n. Since the leading term of r is ϑ^n, and that of $-\alpha r' + a$ is $-n\alpha\eta'\vartheta^n$, we deduce that $-n\alpha\eta' r = -\alpha r' + a$. As in Lemma 4.3, $\int f$ includes the term

$$\int \frac{a}{r} = \int \frac{\alpha r' - n\alpha\eta' r}{r} = \alpha \log r - n\alpha\eta.$$

This contradicts the assumption that f was weakly normalised, and the rest of the proof proceeds identically.

LEMMA 6.4. *If a Risch differential equation satisfying the conclusions of Lemma* 6.3 *has a solution, then that solution is a generalised polynomial in $D[\vartheta, \vartheta^{-1}]$, i.e. $y_f = 0$.*

LEMMA 6.5. *For a Risch differential equation satisfying the conclusions of Lemma* 6.3, *such that either $f_0 = 0$ or f_0 has an elementary integral and there is no integer N such that $\exp(\int f_0 - N\eta) \in D$, there is an algorithm for determining whether or not there is a solution, and finding it.*

Proof. Write $y_p = \sum_{i=-\alpha_1}^{\alpha_2} y_i\vartheta^i$, $f_p = \sum_{i=-\beta_1}^{\beta_2} f_i\vartheta^i$ and $g_p = \sum_{i=-\gamma_1}^{\gamma_2} g_i\vartheta^i$, where the upper limits on the summations are all nonnegative integers, and the lower limits are all nonpositive integers, and the notation is chosen such that $y_{\alpha_2} \neq 0$, unless possibly $\alpha_2 = 0$ etc. The terms of most positive degree in ϑ in $y' + fy$ are $y'_{\alpha_2}\vartheta^{\alpha_2}$, $\alpha_2\eta' y_{\alpha_2}\vartheta^{\alpha_2}$ and $y_{\alpha_2} f_{\beta_2}\vartheta^{\alpha_2+\beta_2}$.

Hence, if $\beta_2 \neq 0$, the leading term of $y' + fy$ is $y_{\alpha_2} f_{\beta_2}\vartheta^{\alpha_2+\beta_2}$, and $\alpha_2 = \gamma_2 - \beta_2$, with $y_{\alpha_2} = g_{\gamma_2}/f_{\beta_2}$. Hence the substitution $y = \hat{y} + y_{\alpha_2}\vartheta^{\alpha_2}$ reduces this equation to a similar one with smaller γ_2, and repeating this process finds the whole of $y_{p+} + y_0$.

If $\beta_2 = 0$, there is cancellation if, and only if, $y'_{\alpha_2} + (\alpha_2\eta' + f_0)y_{\alpha_2} = 0$. The conditions on f_0 imply that this is impossible for $y_{\alpha_2} \in D$ if $\alpha_2 \neq 0$. Hence $\alpha_2 = \gamma_2$, and $y_{\alpha_2} (\alpha_2 > 0)$ is the unique solution to

$$y' + (\alpha_2\eta' + f_0)y = g_{\gamma_2}, \tag{6}$$

which is a Risch differential equation in D, and therefore soluble by hypothesis. Hence the substitution $y = \hat{y} + y_{\alpha_2}\vartheta^{\alpha_2}$ reduces this equation to a similar one with smaller γ_2, and repeating this process finds the whole of y_{p+}.

Identical methods enable us to find y_{p-}, and so we are left with the problem of finding y_0 (in the case $\beta_1 = \beta_2 = 0$). y_0 satisfies $y_0' + f_0 y_0 = g_0$. If $f_0 \neq 0$, then this equation has at most one solution, by the hypothesis on f_0 with $N = 0$, which we can determine by solving this Risch differential equation in D. We then check that g_f is correctly determined, and we have a solution to the whole problem. If $f_0 = 0$, then $y_0 = c + \int g_0$, where c is a constant of integration, which we determine from the condition $(c + \int g_0) f_f = g_f$.

In this lemma, we have shown that $\alpha_2 = \gamma_2 - \beta_2$. This degree condition is not true without the hypothesis that $\exp(\int f_0 - N\eta) \notin D$, as the example[4] of

$$\int \exp\left(\frac{1}{e^x + 1} - 10x\right) \frac{2{,}581{,}284{,}541e^x + 1{,}757{,}211{,}400}{39{,}916{,}800e^{3x} + 119{,}750{,}400e^{2x} + 119{,}750{,}400e^x + 39{,}916{,}800}$$

[4] I am grateful to Prof. Rothstein for pointing out the construction of this example to me.

shows. The integral is, in fact,

$$\exp\left(\frac{1}{e^x+1}-10x\right)\frac{39{,}916{,}800e^{11x}+19{,}958{,}400e^{9x}-26{,}611{,}200e^{8x}+\cdots-175{,}721{,}140}{39{,}916{,}800e^{x}+39{,}916{,}800}.$$

This implies that the, admittedly somewhat obscure, condition on f_0 is necessary. Another example of this problem can be found in the following generalisation of an example of Davenport [1983b]:

$$\int (x^n-n)\exp\left(xe^{1/x^n}+\frac{n+1}{x^n}\right)dx=(-1)^n n!\sum_{i=0}^{n}\frac{(-x)^i\exp\left(xe^{1/x^n}+\dfrac{i}{x^n}\right)}{i!}.$$

Proof of Theorem 3. After applying the reductions of Lemmas 6.2 and 6.3, we know by Lemma 6.4 that y has to be a generalised polynomial. The only problem is to ensure that Lemma 6.5 is applicable. We know that f is integrable, and we can decompose its integral, say h, into a sum of logarithms and a decomposition according to (4):

$$\int f=h_{p+}+h_0+h_{p-}+h_f+\sum_{i=1}^{k}c_i\log p_i(\vartheta)+\sum_{i=k+1}^{l}c_i\log p_i, \tag{7}$$

where the $p_i(\vartheta)(1\leqq i\leqq k)$ are monic polynomials of degree n_i in $D[\vartheta]$ and the $p_i(k<i\leqq l)$ belong to D. Differentiating and equating parts of decomposition (4) is a little tricky, since $(\log p_i(\vartheta))'$ is not a proper rational function, but $n_i\eta'$ plus a proper fraction. Applying this to the derivative of (7) shows that

$$f_0=h_0'+\left(\sum_{i=1}^{k}c_in_i\right)\eta'+\sum_{i=k+1}^{l}\frac{c_ip_i'}{p_i},$$

and the right-hand side of this clearly has an elementary integral.

Since $\vartheta=\exp\eta$, the assertion that $\exp(\int f_0-N\eta)\in D$ is equivalent to $\vartheta^{-N}\exp(\int f_0)\in D$.

Case 1. $\exp\int f_0\notin D(\vartheta)$. We can apply Lemma 6.5 directly.

Case 2. $\exp\int f_0\in D(\vartheta)$. Similarly, unless the expression for $\exp\int f_0$ as a member of $D(\vartheta)$ is of the form $\varphi\vartheta^{-N}$ with $\varphi\in D$ and N an integer, Lemma 6.5 is applicable.

Case 3. $\exp\int f_0=\varphi\vartheta^{-N}$. Then $f_0=-N\eta'+\varphi'/\varphi$, thus the equation to solve has the form

$$y'+\left(f_f-N\eta'+\frac{\varphi'}{\varphi}\right)y=g. \tag{8}$$

If we make the substitution $y=\hat{y}\vartheta^N/\varphi$, this becomes

$$\frac{\hat{y}'\vartheta^N}{\varphi}+\frac{f_f\hat{y}\vartheta^N}{\varphi}=g,$$

which can be rewritten as

$$\hat{y}'+f_f\hat{y}=g\varphi\vartheta^{-N}.$$

Lemma 6.5 can be applied to this equation, and, if it has a solution, the substitution reversed to yield a solution to (8).

7. Exponentially reduced fields. The algorithms defined in the previous sections have occasionally performed substitutions in order to carry out their tasks. Since such substitutions complicate the programming and cause the integrals to appear to have

very different degrees than the integrands would lead one to suspect, we present here a definition of a class of fields in which such substitutions are unnecessary.

An element η of a field $K(z, \vartheta_1, \cdots, \vartheta_n)$ is said to be *exponentially reduced* if the appropriate one of the next two conditions is satisfied and η_0 is exponentially reduced as a member of $K(z, \vartheta_1, \cdots, \vartheta_{n-1})$.

i) ϑ_n *is logarithmic.* If η is decomposed according to (3) with respect to ϑ_n and $\eta_0 \neq 0$, then $\exp(\eta_0) \notin \overline{K(z, \vartheta_1, \cdots, \vartheta_{n-1})}$ (equivalently, η_0 is not a linear combination, with rational coefficients, of the logarithms and arguments of exponentials in $\vartheta_1, \cdots, \vartheta_{n-1}$).

ii) ϑ *is exponential.* If η is decomposed according to (4) with respect to ϑ_n and $\eta_0 \neq 0$, then $\exp(\eta_0) \notin \overline{K(z, \vartheta_1, \cdots, \vartheta_n)}$ (equivalently, η_0 is not a linear combination, with rational coefficients, of the logarithms and arguments of exponentials in $\vartheta_1, \cdots, \vartheta_n$).

We notice that a rational multiple of an exponentially reduced element is exponentially reduced (which would not be true if we only tested membership in $K(z, \vartheta_1, \cdots, \vartheta_n)$, rather than in $\overline{K(z, \vartheta_1, \cdots, \vartheta_n)}$).

A field $K(z, \vartheta_1, \cdots, \vartheta_n)$ is said to be an *exponentially reduced* field if z is transcendental over K with $z' = 1$, K is the field of constants of $K(z, \vartheta_1, \cdots, \vartheta_n)$, and each ϑ_i is transcendental over $K(z, \vartheta_1, \cdots, \vartheta_{i-1})$ such that either:

a) there is a nonzero η in $K(z, \vartheta_1, \cdots, \vartheta_{i-1})$ such that $\vartheta_i' = \eta'/\eta$;

b) there is an η in $K(z, \vartheta_1, \cdots, \vartheta_{i-1})$ such that $\vartheta_i' = \eta'\vartheta_i$, such that no logarithm occurs linearly in η with a rational coefficient and such that η is exponentially reduced as a member of $K(z, \vartheta_1, \cdots, \vartheta_{i-1})$.

We should note that not all fields generated over $K(z)$ by monomials can be expressed in an exponentially reduced way. In fact, there are four possibilities.

i) The presentation of such a field may be exponentially reduced. An example of this is $K(z, \vartheta_1, \vartheta_2)$ where $\vartheta_1' = \vartheta_1$ ($\vartheta_1 = \exp(x)$) and $\vartheta_2' = 2\vartheta_1^2\vartheta_2$ ($\vartheta_2 = \exp(\exp(2x))$).

ii) The presentation given may not be exponentially reduced, but there may be another presentation of the field which is. An example of this is $K(z, \varphi_1, \varphi_2)$ where $\varphi_1' = \varphi_1$ ($\varphi_1 = \exp(x)$) and $\varphi_2' = (1+2\varphi_1^2)\varphi_2$ ($\varphi_2 = \exp(x+\exp(2x))$). This corresponds to the same field as the previous example, with $\vartheta_1 = \varphi_1$ and $\vartheta_2 = \varphi_2/\varphi_1$.

iii) The field may not have an exponentially reduced presentation, but it may be a subfield of a field with an exponentially reduced presentation. An example of this is $K(z, \varphi_1, \varphi_2)$ where $\varphi_1' = 2\varphi_1$ ($\varphi_1 = \exp(2x)$) and $\varphi_2' = (1+2\varphi_1)\varphi_2$ ($\varphi_2 = \exp(x+\exp(2x))$). This field does not have an exponentially reduced presentation, but it is a subfield of the field of the first example, with $\varphi_1 = \vartheta_1^2$ and $\varphi_2 = \vartheta_1\vartheta_2$. The inclusion is strict, since ϑ_1 is in the larger field, but not the smaller.

iv) There may be no exponentially reduced field containing the given field, as in $K(z, \varphi_1, \varphi_2)$ where $\varphi_1' = 4x^3/(x^4+1)$ ($\varphi_1 = \log(x^4+1)$) and $\varphi_2' = (1+2x^3/(x^4+1))\varphi_2$ ($\varphi_2 = \exp(x+\frac{1}{2}\log(x^4+1))$). In order to ensure that the argument to the exponentials contains no logarithms with rational coefficients, we have to introduce $\sqrt{x^4+1}$, and, as is well known, this cannot be rationalised.

We now claim that the process of applying the Risch integration algorithm (part (a)) and the algorithm proposed in §§ 4–6 of this paper to an element of an exponentially reduced field does not require any substitutions. The Risch part (a) process does not, and the process of §§ 4–6 requires substitutions if the equation is not weakly normalised, or in Case 2 of Lemma 5.7 and Case 3 of Theorem 3. In each Risch differential equation that part (a) requires us to solve, f is of the form $N\eta'$, where N is an integer and η the argument of one of the exponential ϑ_i, since it comes from the integration of $g\vartheta_i^N$. Since no logarithm occurs linearly in η with a rational coefficient, $N\eta'$ is weakly

normalised with respect to all the variables occurring in it. Similarly, the requirement that η be exponentially reduced means that Case 2 of Lemma 5.7 does not happen, nor Case 3 of Theorem 3.

We still need to verify that the equations generated recursively satisfy these constraints. Recursion occurs in Case 1 of Lemma 5.7 and in the last paragraph of the proof of Lemma 6.5. In both cases, the equations generated are of the form $y'+f_0y=g$. We have already seen that f_0 is always integrable in these cases, and the condition that η contains no logarithms with rational coefficients implies that f_0 is weakly normalised with respect to its main variable. Since the definition of "exponentially reduced" contains a recursive clause, $\int f_0$ is also exponentially reduced. The only other instance of recursion is in (6). Here the new equation contains $n\eta'+f_0$ for some integer n, where η is the argument of an exponential ϑ_i. This is certainly still weakly normalised, but, in general, the sum of two exponentially reduced elements is not exponentially reduced. If $\int(n\eta'+f_0)$ is not exponentially reduced, we can write

$$\int(n\eta'+f_0)=\sum c_i\varphi_i,$$

where the c_i are rational numbers and the φ_i are the logarithmic ϑ_i and the arguments of the exponential ϑ_i. But η is a summand on the right, so we can write $\int f_0$ as such a linear combination, so that f_0, and hence f, are not exponentially reduced. Thus this recursion does not introduce any problems.

Another way of expressing the claim we have made is the following result.

THEOREM 4. *If $K(z, \vartheta_1, \cdots, \vartheta_n)$ is an exponentially reduced field, then the solutions of each Risch differential equation encountered during this integration process satisfy the "natural" degree bounds*:

a) $y_d \mid gcd(g'_d, g_d)$;

b) degree $(y_{p+}) \leqq$ degree $(g_{p+})+1$ (*if the main variable is z or logarithmic*);

c) degree $(y_{p+}) \leqq$ degree (g_{p+}) *and* degree $(y_{p-}) \leqq$ degree (g_{p-}) (*if the main variable is exponential*).

This result is far from being a complete degree bound on integrands, but it does indicate that a class of anomalies cannot take place in exponentially reduced fields.

8. Conclusions. We have seen that it is possible to solve a Risch differential equation problem in a manner similar to Hermite's method for integrating rational functions, and that this method gives one more information about the subproblems arising than does the method of solution proposed by Risch. If the field in which the original integrand that gave rise to the Risch differential equation problem lies was exponentially reduced, then this solution can be found without any changes of variable and without the introduction of any arbitrary constants which have to be solved for later, except for the constant of integration introduced in Lemma 5.6. This is very similar to the treatment in part (a) of the Main Theorem (Risch [1969]), where a constant of integration appears in the integration of polynomials in a logarithmic main variable.

Our method of solution is somewhat different from that of Rothstein [1976], who proceeds by clearing denominators and reducing the equation to a purely polynomial one. It is unclear which algorithm will prove most efficient in practice.

We have also arrived at the characterisation of "exponentially reduced" fields, which would seem to be important from the point of view of Risch differential equation problems, especially in view of Theorem 4. We generalise Theorem 4 into the following

CONJECTURE (after Risch and Norman, see Davenport [1982]). Let $D=K(z, \vartheta_1, \cdots, \vartheta_n)$ be an exponentially reduced field, and f an element of D with an

integral that is elementary over D, so that

$$\int f = \frac{p}{q} + \sum_{i=1}^{k} c_i \log p_i,$$

where p, q, $p_i \in \bar{K}[z, \vartheta_1, \cdots, \vartheta_n]$, and $c_i \in \bar{K}$. Then:

a) Each p_i is a factor of the denominator of f, or a factor of the arguments of the existing logarithms among the ϑ_i;

b) Each irreducible factor of q which is not an exponential monomial occurs at most $n-1$ times in q, where n is its multiplicity in the denominator of f;

c) Each irreducible factor of q which is an exponential monomial occurs at most n times in q, where n is its multiplicity in the denominator of f;

d) The degree of each of $z, \vartheta_1, \cdots, \vartheta_n$ in p/q is at most one more than its degree in f, where the degree of ϑ_i in a rational function is defined as the difference of its degree in the numerator and denominator (but never negative).

Acknowledgments. This research was largely performed at the Computer and Information Sciences Department of the University of Delaware, and the author is grateful to the System Development Foundation for its support under grant 301, and to the University for its hospitality. At that time the author was a Research Fellow of Emmanuel College Cambridge, and he is grateful to the College for its support. Dr. B. M. Trager posed the original question, and Prof. B. F. Caviness and Dr. G. W. Cherry have read many drafts of this paper. Mr. J. A. Abbott, Prof. M. Rothstein, Prof. M. F. Singer and Dr. Trager also made many useful comments.

REFERENCES

B. F. CAVINESS, *Methods for symbolic computation with transcendental functions*, Proc. 4th International Colloquium on Advanced Computing Methods on Theoretical Physics, A. Visconti, ed., Marseilles, 1977, pp. 16-43.

G. W. CHERRY, *Algorithms for integrating elementary functions in terms of logarithmic integrals and error functions*, Ph.D. thesis, Univ. Delaware, Newark, DE, August 1983.

J. H. DAVENPORT, *On the parallel Risch algorithm* (I), Proc. EUROCAM '82, J. Calmet, ed., Lecture Notes in Computer Science 144, Springer, Berlin, 1982, pp. 144-157.

———, *Intégration formelle*, Rapport de Recherche 375, IMAG, Grenoble, May 1983a.

———, *Integration—what do we want from the theory?*, Proc. EUROCAL '83, J. A. van Hulzen, ed., Lecture Notes in Computer Science 162, Springer, Berlin, 1983b, pp. 2-11 (also Appendix 3 of Davenport [1983a]).

———, *Intégration algorithmique des fonctions élémentairement transcendantes sur une courbe algébrique*, Annales de l'Institut Fourier, 34 (1984), pp. 271-276.

J. H. DAVENPORT AND B. M. TRAGER, *Factorization over finitely generated fields*, Proc. SYMSAC 81, P. S. Wang, ed., ACM, New York, 1981, pp. 200-205.

C. HERMITE, *Sur l'intégration des fractions rationnelles*, Nouvelles Annales de Mathématiques, 2 Sér. 11 (1872), pp. 145-148.

R. H. RISCH, *The problem of integration in finite terms*, Trans. Amer. Math. Soc., 139 (1969), pp. 167-189.

M. ROSENLICHT, *On Liouville's theory of elementary functions*, Pacific J. Math., 65 (1976), pp. 485-492.

M. ROTHSTEIN, *Aspects of symbolic integration and simplification of exponential and primitive functions*, Ph.D. thesis, Univ. Wisconsin, Madison, WI, 1976.

M. ROTHSTEIN AND B. F. CAVINESS, *A structure theorem for exponential and primitive functions*, this Journal, 8 (1979), pp. 357-367.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986

© 1986 Society for Industrial and Applied Mathematics

DATA STRUCTURES FOR RETRIEVAL ON SQUARE GRIDS*

MARTIN DAVID KATZ† AND DENNIS J. VOLPER†

Abstract. Families of data structures are presented for retrieval of the sum of values of points within a half-plane or polygon, given that the points are at integral coordinates ($N \times N$) in the plane. Fredman has shown that the problem has a lower bound of $\Omega(N^{2/3})$ for intermixed updates and retrievals. When the points are not restricted to integral coordinates, Edelsbrunner and Welzl have shown a retrieval time of $O(N^{\approx 1.39})$ (update time $= O(N^2 \log N)$). One of the data structures presented here permits intermixed updates and retrievals in $O(N^{2/3} \log N)$.

We store multiple, rotated data structures to match against the query. Rotation appears to be an effective method for trading-off update time against retrieval time for geometric problems. We also present constructions for efficient retrieval of triangles and polygons. For our data structures, the expected complexity when the points are uniformly distributed is less than the worst case complexity when the points are at integral coordinates.

Key words. data structures, polygonal retrieval, range query

CR category. 5.25 computational complexity

AMS(MOS) subject classifications. 68P05, 68Q25, 68P20

1. Introduction.

Suppose there are values at various points in a plane and one needs to determine the total of the values within a region of the plane. For example, one might need to determine the distribution of people within a region for traffic planning, population growth studies, or electoral redistricting.

We develop families of data structures which support updating of the values and retrieval of the sum of the values associated with points within such regions. Within each family, update time can be traded off against retrieval time. Initially, we assume that the points are fixed in an $N \times N$ grid. Associated with each point is a value which is a member of a commutative semi-group. The work reported here is based on [Katz 85].

1.1. Background.

The simplest form of geometric retrieval is orthogonal range query ([Lueker 78], [Willard 78], [Burkhard, Fredman and Kleitman 81], [Fredman 82], [Lueker and Willard 82], and [Willard 85]). The regions retrieved by orthogonal range queries are rectangles with sides parallel to the axes. Structures exist in which update and retrieval times are both $O(\log n^d)$ for a d-dimensional range query on n points. The one-dimensional range query structure is generally a full binary tree, in which the data points are at the leaves and each internal node contains the sum of the values of its two children.

Fredman [Fredman 80] gives a method for deriving lower bounds for the complexity of intermixed update and retrieval operations. This paper and those cited below discuss retrieval of regions which are more general than rectangles. In fact, the examples in the paper include half-plane and polygon retrievals. Fredman gives a lower bound complexity of $\rho + t = \Omega(N^{2/3})$ for intermixed updates and retrievals on the $N \times N$ grid.

For problems containing n points, Willard [Willard 82] presents a data structure for performing half-plane and polygon retrieval of points in the plane with no restriction

* Received by the editors August 21, 1984, and in revised form June 3, 1985.

† Department of Information and Computer Science, University of California, Irvine, California 92717.

on their position. Willard's structure has retrieval time $O(n^{\log_6 4})$ and preprocessing time $O(n^2)$. Edelsbrunner and Welzl [Edelsbrunner and Welzl 83] improved the retrieval time bound to $O(n^{\approx .695})$, while Megiddo [Megiddo 84] improved the preprocessing time to $O(n \log n)$.

Yao [Yao 83] extends the structure to 3 dimensions. Yao's structure has retrieval time $O(n^{\approx .98})$ and preprocessing time $O(n^4)$. Cole [Cole 85] extends the result to 4 dimensions, while Avis [Avis 84] shows that the approach cannot be extended to 5 or more dimensions.

Cole and Yap [Cole and Yap 83] present static data structures for half-plane, triangle, disk, half-space, etc. retrieval, where the points are not restricted to a grid. Their half-plane structure has retrieval time $O(\log n)$, space $O(n^2)$, and preprocessing time $O(n^3)$.

One of the half-plane data structures presented in this paper permits intermixed half-plane updates and retrievals on the $N \times N$ grid with complexity $O(N^{2/3} \log N)$ (i.e., $O(n^{1/3} \log n)$). In our data structure, unlike the previous structures, parameter selection permits trade-off of update time against retrieval time. Although the data structures presented here have significantly lower worst case complexity, the worst case analysis only applies when the points are arranged in a grid; previous data structures did not restrict the location of points in the plane.

1.2. General approach. Many data structures (e.g., range query) have an orientation. Our approach is to build many of these structures, each with a different orientation, covering the same data. We call each of these oriented structures a *rotation.* Each rotation contains a substructure which solves a query with a given orientation, and substructures to solve other closely related queries.

1.3. Computational model. Following [Fredman 80], we formally define a *geometric retrieval problem* as a pair $(\mathcal{K}, \Gamma)$, where $\mathcal{K}$ is a set referred to as the key space, and Γ is a collection of subsets of $\mathcal{K}$ which covers $\mathcal{K}$ (Γ is the set of regions). The objective is to define a data structure representing a set $\mathcal{T}$ of records, where each record τ has a uniquely identifying key in $\mathcal{K}$ (denoted key (τ)), and an associated value (denoted value (τ)), which is an element of a commutative semi-group S (with operator, $\oplus$). Since we do not use any properties which are specific to a particular semi-group, one may think of the semi-group as the addition operator over the nonnegative integers. Other examples of commutative semi-group operators are maximum and set union.

The data structure must support the following:

Retrieve(R): Compute the sum (using $\oplus$) of the values associated with all points in $\mathcal{T}$ whose keys lie in the region R $(R \in \Gamma)$

Insert(k, x): Add a record $\tau \in \mathcal{T}$ such that key $(\tau) = k$ and value $(\tau) = x$.

Delete(k): Remove a record τ from $\mathcal{T}$ where key $(\tau) = k$.

Update(k, x): Change a record $\tau \in \mathcal{T}$ such that key $(\tau) = k$ and new-value $(\tau) =$ odd-value $(\tau) \oplus x$. $(k \in \mathcal{K}$, and $x \in S)$

The data structure $\mathcal{D}$ is an ordered collection of variables. Each variable's value field holds a value in S, and each variable's key field holds a representation of the set of keys which are covered by this variable (for all of our data structures, this key field can be represented in $O(\log N)$ bits).

Retrieval time t is defined as the number of operations on values in S which are necessary to perform a retrieval using $\mathcal{D}$. *Storage redundancy* ρ is defined as the maximum number of variables in $\mathcal{D}$ whose value depends on any single record in $\mathcal{T}$. For our data structures, update time is proportional to storage redundancy. The

complexities in this paper are measured in semi-group operations, because the time to perform semi-group operations dominates the time to traverse the data structures.

When the points are at integral coordinates, the records in $\mathcal{T}$ correspond to the points in a planar grid. The set of keys $\mathcal{K} = \{(i,j) | (i,j) \in (\{1, \cdots, N\}, \{1, \cdots, N\})\}$. That is, the key of a record is the coordinates of the point. If a record $\tau \in \mathcal{T}$ has not been assigned a value, value $(\tau) = 0$, where 0 is the identity value for the semi-group S.

2. The half-plane problem. In the half-plane problem, we define $\Gamma_{hp} = \{\{(i,j) | ai + bj \geqq 1\}$ for $(i,j) \in (\{1, \cdots, N\}, \{1, \cdots, N\})$ and $(a,b) \neq (0,0)$ and $a, b \in \Re\}$. That is, each region to be retrieved $g \in \Gamma_{hp}$ is a collection of points in the intersection of a half-plane and the grid.

2.1. The half-plane data structure. The half-plane data structure consists of a collection of rotations. Each is used to retrieve a range of angles. Each half-plane structure is divided into *slices* and each slice is divided into *rectangles.* The size and shape of each rectangle is calculated to limit the cost of retrieving the rectangles which overlap the half-plane boundary.

More specifically, the half-plane data structure is a collection of $2N^{\alpha}$ rotations equally spaced over one half circle. Each rotation is covered by $2N^{\beta}$ disjoint slices.

Each slice is covered by $2N^{\delta}$ disjoint rectangles (Fig. 1 shows the rectangles for one slice). Each rectangle is up to $(1/\sqrt{2})N^{1-\beta}$ wide and up to $(1/\sqrt{2})N^{1-\delta}$ long. Each rectangle in a rotation is identically shaped, with its long side parallel to the orientation of the rotation. The rectangles in a slice are stored in a range query structure. Likewise, the points within each rectangle are stored in a range query structure.

In the analysis below, we develop a description of the legal values for α, β, and δ. As we shall see, these restrictions limit the area of each rectangle to less than 1, while limiting the number of rectangles crossed by the half-plane edge to a constant.

2.2. Performing a half-plane retrieval. To perform a retrieval, select the rotation closest to the orientation of the half-plane edge. Each slice in this rotation is retrieved in two parts: the interior of the slice and the frontier of the slice.

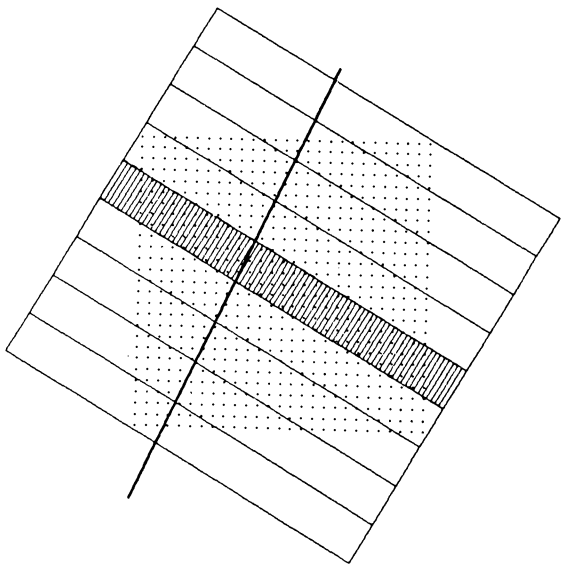


FIG.1. *One rotation in the half-plane data structure. For clarity, rectangles are shown for only one slice. The line extending across the structure is a sample half-plane edge whose orientation is covered by the rotation.*

The interior of the slice is the collection of rectangles in the slice which are entirely within the half-plane. The interior is retrieved as a range query. The frontier of the slice is the collection of rectangles in the slice which are crossed by the half-plane edge. As points in a rectangle are collinear (see Lemma 2.2), a range query is used to retrieve the points in each frontier rectangle.

2.3. Analysis of storage redundancy.

THEOREM 2.1. *The storage redundancy of the half-plane data structure is* $O(N^{\alpha} \log N)$.

Proof. In each rotation, each point is in a single rectangle and slice. The range query structures in the rectangle and slice both have a storage redundancy of $O(\log N)$. Thus, the storage redundancy is $O(\log N)$ times the number of rotations $(2N^{\alpha})$. □

2.4. Analysis of retrieval time.

LEMMA 2.2. *If the area of a rectangle is less than* 1, *then all points in the intersection of the integer grid and the rectangle must be collinear.*

Proof. By contradiction: Suppose that there are three noncollinear points in the rectangle. Three noncollinear points define a triangle. By the dot product rule, the area of the smallest triangle whose vertices are all integers is $\frac{1}{2}$. The area of the largest triangle which can be inscribed in a rectangle is half the area of the rectangle. This constitutes a contradiction since it was given that the area of the rectangle is less than 1. □

COROLLARY 2.2.1. *If* $\beta + \delta \geqq 2$, *where* β *and* δ *are as described in the data structure given above, then all points on an integer grid within a rectangle are collinear.*

LEMMA 2.3. *Given a half-plane and the most closely aligned rotation of the data structure, the maximum number of rectangles crossed by the half-plane edge within any slice of that rotation is limited to* $1+[N^{\delta-\alpha-\beta}]$.

Proof. The slope of the half-plane edge relative to the slices is no more than $N^{-\alpha}$ from perpendicular and the maximum width of the slice is $(1/\sqrt{2})N^{1-\beta}$. The section of the half-plane edge which intersects the slice can be contained in a rectangle $\mathcal{R}$ extending $(1/\sqrt{2})N^{1-(\alpha+\beta)}$ along the slice. The maximum width of each rectangle is $(1/\sqrt{2})N^{1-\delta}$. The maximum number of rectangles which can be overlapped by $\mathcal{R}$ is obtained by dividing. □

THEOREM 2.4. *Given* $2-\beta \leqq \delta \leqq \alpha+\beta$, *the worst case time to perform a half-plane retrieval using the data structure described above is* $O(N^{\beta} \log N)$.

Proof. Each retrieval requires at most $2N^{\beta}$ slice retrievals. Each slice retrieval requires a range query on the rectangles in the interior of the slice, and a range query on the points in each frontier rectangle. Lemma 2.3 shows that, for $\delta \leqq \alpha+\beta$, only two rectangle retrievals are required per slice. Thus, $O(N^{\beta})$ range queries must be performed, yielding a retrieval time of $O(N^{\beta} \log N)$. □

For $\delta = \alpha+\beta$, we can adjust β to trade-off storage redundancy and retrieval time. We obtain the following corollaries:

COROLLARY 2.4.1. *If* $\alpha = 2-2\beta$, $0 \leqq \beta \leqq 1$, *then* $[\rho, t] = [O(N^{2-2\beta} \log N), O(N^{\beta} \log N)]$.

If $\beta = \frac{2}{3}$ *then* $[\rho, t] = [O(N^{2/3} \log N), O(N^{2/3} \log N)]$.

COROLLARY 2.4.2. *If* $\beta = 0$, $\alpha = 2$, *then* $[\rho, t] = [O(N^2 \log N), O(\log N)]$ *and space* $= O(N^4)$ *for* N^2 *points.*

This retrieval time and space is of the same order of magnitude as in [Cole and Yap 83], but with less preprocessing.

COROLLARY 2.4.3. *If* $\beta = 1$, $\alpha = 0$, *then* $[\rho, t] = [O(\log N), O(N \log N)]$ *and space* $= O(N^2)$ *for* N^2 *points.*

The data structure presented in [Edelsbrunner and Welzel 83] also requires linear space, but has retrieval time of $O(N^{\approx 1.39})$ (the storage redundancies are within a constant). Note that unlike previous work, we restrict points to a grid.

3. Retrieval of triangular regions. The region to be retrieved consists of the points within a triangle superimposed on the grid. The asymptotic complexity bounds presented in previous work for triangle retrieval are the same as for half-plane retrieval.

3.1. The triangle data structure. We divide the grid and triangle with squares to convert the retrieval into simpler regions, such as half-planes (see Fig. 2). The squares form N^{λ} rows of N^{λ} squares each ($0 \leqq \lambda \leqq 1$).

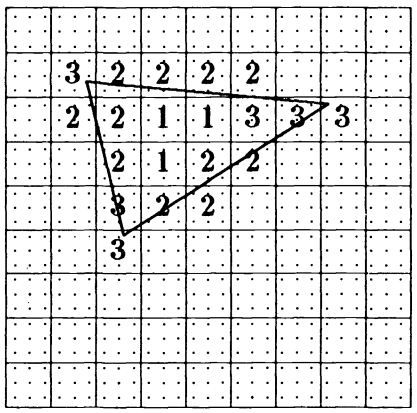


FIG. 2. *A sample triangle and the corresponding classified squares.*

A range query structure is associated with each row of squares to permit the retrieval of a contiguous subset of squares in the row. This structure is used to retrieve those squares which are completely contained in the triangle. Each square contains a half-plane retrieval structure which is used when one or more edges cross the square.

3.2. Performing a triangle retrieval. We superimpose the triangle on the lattice of squares, and classify each square according to its relationship to the edges of the triangle. The following cases are possible:

1. The square is entirely inside the triangle.
2. The square is crossed by one edge of the triangle.
3. The square is crossed by two or three edges of the triangle.

The squares which are entirely contained within the triangle (case 1) are retrieved by a range query on the rows of squares. Each square which has only one edge passing through it (case 2) is retrieved by half-plane retrieval.

When multiple edges cross the square (case 3), we retrieve in three steps. The first step is to select the rotation closest to one of the edges (called the *alignment edge*) and retrieve all rectangles crossing that edge. Next, the rectangles in the interior of each slice (not crossing an edge) are retrieved. Third, the rectangles which overlap the *other* edges are retrieved.

3.3. Analysis of storage redundancy.

THEOREM 3.1. *The storage redundancy ρ of the triangle data structure is* $\rho = O(N^{(1-\lambda)(2-2\beta)} \log N)$.

Proof. The storage redundance due to the range query structure on the rows of squares is $\rho = O(\log N^{\lambda})$. Each square contains a half-plane structure of size $N^{1-\lambda}$. For this structure, $\rho = O(N^{(1-\lambda)(2-2\beta)} \log N)$ (Corollary 2.4.1). The second term dominates. □

3.4. Analysis of retrieval time.

THEOREM 3.2. *The retrieval time of the triangle data structure is the maximum of*

$$O(N^{(1-\lambda)(2-\beta)} \log N) \quad \textit{and} \quad O(N^{(1-\lambda)\beta+\lambda} \log N).$$

Proof. We examine each case defined in §3.2 individually:

1. The number of rows of squares entirely inside the triangle is at most N^{λ} and each row requires $t = O(\log N)$. The retrieval time for all squares in case 1 is $t = O(N^{\lambda} \log N)$.

2. The square is crossed by one edge of the triangle. The retrieval time for the half-plane structure in each square is $t = O(N^{(1-\lambda)\beta} \log N)$ by Corollary 2.4.1. Only $O(N^{\lambda})$ squares can be crossed by triangle edges, yielding a total retrieval time of $t = O(N^{(1-\lambda)\beta+\lambda} \log N)$.

3. As in case 2, retrieving the frontier along the *alignment* edge and interior rectangles requires a total time of $O(N^{(1-\lambda)\beta+\lambda} \log N)$.

Because the *other* edges are in the same square as the alignment edge, they are no more than $D = \sqrt{2}N^{1-\lambda}$ away. The number of rectangles which can cross each other's edge is limited to the number of slices, plus D divided by the width of a rectangle.

This is $O(N^{(1-\lambda)\beta+\lambda}) + O(N^{(1-\lambda)\delta})$. Since $\delta = 2-\beta$, the total retrieval time for case 3 is $t = O(N^{(1-\lambda)(2-\beta)} \log N) + O(N^{(1-\lambda)\beta+\lambda} \log N)$. □

COROLLARY 3.2.1. *For* $\lambda = \frac{2}{3}$ *and* $\beta = 0$,

$$[\rho, t] = [O(N^{2/3} \log N), O(N^{2/3} \log N)].$$

COROLLARY 3.2.2. *For* $0 \leqq \lambda \leqq \frac{2}{3}$, *the retrieval time is minimal if* $(1-\lambda)(2-\beta) = (1-\lambda)\beta + \lambda$. *If we set* $\beta = (2-3\lambda)/(2-2\lambda)$, *a family of data structures which trades-off storage redundancy in the range* $[\log N \cdots N^{2/3} \log N]$ *and retrieval time in the range* $[N^{2/3} \log N \cdots N \log N]$ *can be constructed.*

COROLLARY 3.2.3. *On the square grid, the triangle data structure can be used to retrieve convex polygons (with v vertices) with complexity*

$$[\rho, t] = [O(N^{(1-\lambda)(2-2\beta)} \log N), O(v \max (N^{(1-\lambda)(2-\beta)} \log N, N^{(1-\lambda)\beta+\lambda} \log N))].$$

Proof. Any convex polygon with v vertices can be expressed (in linear time) as the disjoint union of $O(v)$ triangles. □

4. Retrieval of circular disks. We retrieve the sum of the values associated with those grid points which fall with a circle. We define $\Gamma_{\text{circle}} = \{\{(i,j) | (i-a)^2 + (j-b)^2 \leqq r^2\}$ for $(i,j) \in (\{1, \cdots, N\}, \{1, \cdots, N\})$ and $a, b, r \in \Re\}$. We observe that this is the same as the set of points which fall within the convex hull of the points within the circle. We refer to the convex hull as $\mathscr{C}$. $\mathscr{C}$ is a convex polygon with its vertices at integral coordinates. From Appendix A we know that $\mathscr{C}$ has $O(N^{2/3})$ edges.

Since we are only considering semi-group operations, our analysis ignores the time required to find the convex hull. General algorithms for obtaining the convex hull of a collection of n points require $O(n \log n)$ arithmetic operations in the worst case (the expected number of arithmetic operations is $O(n)$) [Shamos 78].

4.1. The circle data structure. The data structure is similar to that for triangle retrieval. For the circle data structure, we divide the grid into squares whose sides are

parallel to the grid axis. The squares form N^{λ} rows of N^{λ} squares ($0 \leqq \lambda \leqq 1$). There is a range query structure associated with each row of squares permitting the retrieval of a contiguous subset of squares in the row. The data structure in each square is a modified half-plane retrieval data structure. Thus, it is a collection of $2N^{(1-\lambda)(2-2\beta)}$ rotations, each containing $2N^{(1-\lambda)\beta}$ slices.

In the standard half-plane data structure (see § 2), each node in the range query structure for the slice has a variable associated with it containing the sum of the values covered by the node. In the half-plane data structure modified for circle retrievals, we replace the variable within each node with a two-dimensional orthogonal range query structure (see Fig. 3).

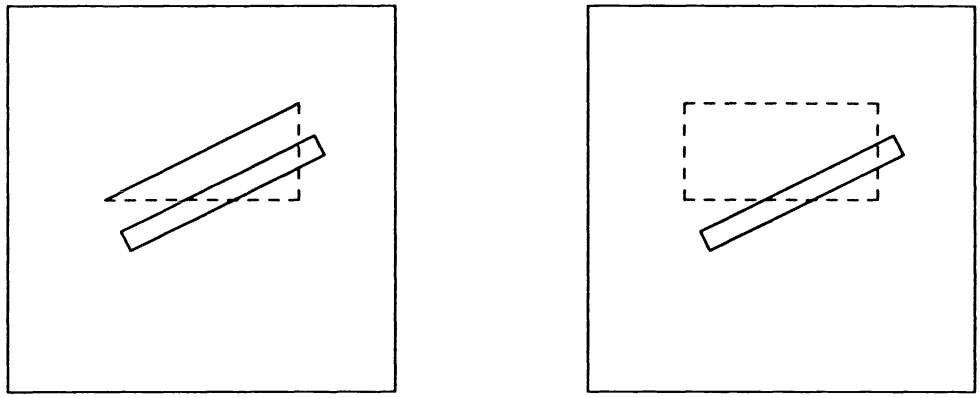


FIG. 3. *Retrieval of an orthogonal triangle. Left figure: Overlap of rectangle and triangle to be retrieved. Right figure: Rectangle and orthogonal range query. Retrieval yields intersection of rectangles.*

The two-dimensional orthogonal range query structure is oriented parallel to the grid axis and has storage redundancy and retrieval time equal to $O(\log^2 N)$. This structure permits retrieval of rectangles and right triangles aligned with the grid axes. A geometric construction converts the regions formed by obtuse angles into these two types of regions.

4.2. Performing a circle retrieval. For each square we have the following cases:

1. The diameter of $\mathscr{C}$ is less than $4N^{1-\lambda}$.
2. The square is entirely inside $\mathscr{C}$.
3. The square is crossed by exactly one edge of $\mathscr{C}$.
4. The square is crossed by two or more edges of $\mathscr{C}$ which are at an obtuse angle.

If $\mathscr{C}$ is very small (case 1), then we retrieve all of the rectangles in the square which overlap an edge and all of the rectangles in the interior of a slice.

The squares which are entirely contained within the region (case 2) are retrieved by range query on the rows of squares. The squares which have only one edge passing through them (case 3) are retrieved by half-plane retrieval.

If there is more than one edge in the square (case 4), we divide the region into subregions which have at most one edge which is not parallel to the grid axes. The number of subregions is no more than twice the number of edges of $\mathscr{C}$ passing through the square. Each subregion can be retrieved using the modified half-plane structure.

The subregions are formed by drawing a horizontal line from each vertex of $\mathscr{C}$ and from each point at which an edge passes through the side of the square until they meet a side of the square or an edge of $\mathscr{C}$. Then, a vertical line is drawn from each vertex of $\mathscr{C}$ and from each point at which an edge of $\mathscr{C}$ passes through the side of the

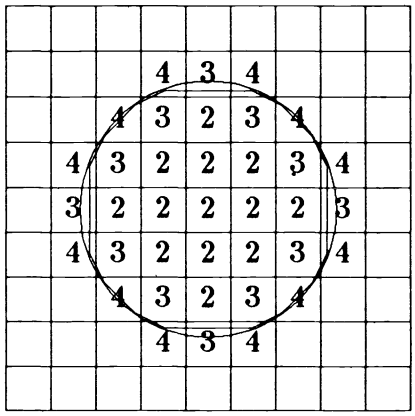


FIG. 4. *A sample circle and the corresponding classified squares.*

square until they meet a side of the square, an edge of $\mathscr{C}$, or one of the horizontal lines. This results in $O(N^{2/3})$ right triangular and rectangular regions (see Fig. 5). Each of these are retrieved using the modified half-plane structure.

4.3. Analysis of storage redundancy.

THEOREM 4.1. *The storage redundancy of this data structure is* $\rho = O(N^{(1-\lambda)(2-2\beta)} \log^3 N)$.

Proof. Each point is covered by the data structure for rows of squares and the data structures within the squares.

The component of the storage redundancy due to the range query structure on the rows of squares is $\rho = O(\log N)$.

The component of the storage redundancy per square due to the modified half-plane structure is the number of rotations ($O(N^{(1-\lambda)(2-2\beta)})$) times the redundancy per rotation. The redundancy per rotation is the redundancy of the two-dimensional orthogonal range query structure within each range query tree node ($O(\log^2 N)$) times the $O(\log N)$ height of the tree in the range query structure in each slice. Thus, the storage redundancy of the modified half-plane structure is $\rho = O(N^{(1-\lambda)(2-2\beta)} \log^3 N)$.

The complexity is dominated by the second term. □

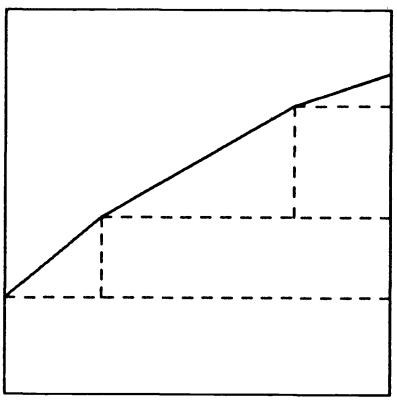


FIG. 5. *A square from the circle problem, showing the rectangles and triangles which form its subregions.*

4.4. Analysis of retrieval time.

THEOREM 4.2. *The retrieval time for the data structure is*

$$t = O(\max(N^{(1-\lambda)\beta+2/3}\log^3 N, N^{(1-\lambda)\beta+\lambda}\log^3 N, N^{(1-\lambda)(2-\beta)}\log^3 N)).$$

Proof. Each square falls into one of the cases below:

1. $\mathscr{C}$ is small enough so that only a constant number of squares are involved. The square is retrieved by retrieving each rectangle crossed by an edge and the interior of each slice in the square. The analysis resembles that for the triangle problem. Retrieval time for this case is $t = O(N^{(1-\lambda)(2-\beta)}\log^3 N + N^{(1-\lambda)\beta}\log^3 N)$.

2. For squares completely within $\mathscr{C}$, retrieve each row of squares by range query. There are at most N^{λ} rows, each taking $O(\log N)$ time. Retrieval time for this case is $t = O(N^{\lambda}\log N)$.

3. If exactly one edge crosses the square, then the square is retrieved by half-plane retrieval. Each square is $O(N^{1-\lambda})$ on a side; therefore, each square has a retrieval time of $O(N^{(1-\lambda)\beta}\log^3 N)$. There are at most $O(N^{\lambda})$ such squares. For this case, the total retrieval time is $t = O(N^{(1-\lambda)\beta+\lambda}\log^3 N)$.

4. The number of vertices in the convex hull of the points within a circle of radius R, whose vertices are at integral coordinates, is $O(R^{2/3})$ (see Appendix A). Further, for $R > N$, the number of convex hull vertices within a square N on a side is $O(N^{2/3})$. Therefore, the number of occurrences of this case is $O(N^{2/3})$.

Each of the subregions (the constructed triangles and rectangles) can be retrieved by modified half-plane retrieval in $t = O(N^{(1-\lambda)\beta}\log^3 N)$. There are no more than $O(N^{2/3})$ subregions per circle. Therefore, the total retrieval time for this case is $t = O(N^{(1-\lambda)\beta+2/3}\log^3 N)$.

The theorem is obtained by adding the complexities of the cases. □

COROLLARY 4.2.1. *Let* $\beta = 0$, *and* $\lambda = \frac{2}{3}$. *The complexity of the circle retrieval data structure is* $[\rho, t] = [O(N^{2/3}\log^3 N), O(N^{2/3}\log^3 N)]$.

5. General polygon retrieval. Given the $N \times N$ grid defined above, we retrieve the sum of the values attached to the points which fall within a region defined by a polygon $\mathscr{P}$. $\mathscr{P}$ is composed of one or more closed, piecewise-linear boundaries. We are told in advance which regions bounded by $\mathscr{P}$ are inside the region to retrieve and which are outside.

We define φ as a polygon's concavity factor (a measure of the complexity of the polygon). For a polygon $\mathscr{P}$, $\varphi(\mathscr{P})$ is the sum of

- the number of times the boundaries cross,
- the number of boundaries,
- $1/2\pi$ times the sum of the turning angles of the concave vertices (rounded up).

Each acute angle causes a turning angle (absolute value of the external angle minus π) of at least $\pi/2$. The total turning angle for a simple (connected) polygonal boundary is 2π plus the sum of the turning angles of the concave vertices. Dividing the total turning angle by the minimum turning angle of an acute vertex implies that the total number of acute angles in $\mathscr{P}$ can be no more than 4φ.

A boundary with length cN within the $N \times N$ boundaries of the square grid must have a turning angle of at least $c\pi/2$. If $\mathscr{P}$ is not a simple polygon (i.e., it is composed of several boundaries) the length of the perimeter is the sum of the lengths of the boundaries. Therefore, the perimeter of the portion of $\mathscr{P}$ within the square grid has a length of at most $4N\varphi$.

By drawing appropriate construction lines, each boundary of $\mathscr{P}$ can be divided into convex polygons. If such a convex polygon has more than $O(P^{2/3})$ vertices (P is the length of the perimeter of the convex polygon), then a convex hull for the points

within the convex polygon can be substituted for the convex polygon (see Appendix A). A piecewise convex hull can be constructed for any polygon such that the number of vertices within the square grid is $O(\varphi N^{2/3})$. We may therefore assume, without loss of generality, that the polygon has $O(\varphi N^{2/3})$ vertices.

THEOREM 5.1. *We can retrieve the sum of the values associated with the grid points within any polygon with complexity*

$$[\rho, t] = [O(N^{(1-\lambda)(2-2\beta)} \log^3 N), O(\varphi \max (N^{(1-\lambda)\beta+\lambda} \log^3 N), N^{(1-\lambda)\beta+2/3} \log^3 N, N^{(1-\lambda)(2-\beta)} \log^3 N)].$$

If $\beta = 0$ *and* $\lambda = \frac{2}{3}$, *then* $[\rho, t] = [O(N^{2/3} \log^3 N), O(\varphi N^{2/3} \log^3 N)]$.

Proof. We use the circle data structure. Connect all boundaries into a single concave polygon without increasing the concavity factor as follows:

1. If an edge separates two regions which are both inside (or outside) the region to retrieve, the edge is removed.

2. Given two boundaries, connect one vertex of the first boundary to a vertex on the other boundary (producing a zero area tunnel containing no points).

3. If the polygonal boundary crosses itself, the polygon is cut at the crossing and reattached such that the crossing is removed (possibly adding a vertex). This produces a zero area tunnel containing no points.

Superimpose the polygon on a square lattice as in the circle problem. If there are multiple portions of the region overlapping a square, retrieve each portion separately. Retrieve the regions as in the circle problem.

Since the time to retrieve vertices ($t = O(\varphi N^{(1-\lambda)\beta+2/3} \log^3 N + \varphi N^{(1-\lambda)(2-\beta)} \log^3 N)$), edges ($t = O(\varphi N^{(1-\lambda)\beta+\lambda} \log^3 N)$), and contiguous rows of squares entirely within the region ($t = O(\varphi N^{\lambda} \log N)$) are all limited by a value proportional to the concavity factor, the retrieval can be performed in the stated time. □

6. Uniformly distributed points. In discussing retrievals in previous sections, we required that the points be located at integral coordinates. Many applications do not meet this constraint. Here, we analyze the complexity of the data structures when the position of each point is randomly selected.

Analysis of the data structures on the integer grid yielded a worst case complexity. Analysis of these same data structures, on a square region of the plane in which points are uniformly distributed, yields the same storage redundancy as when the points are in a grid. However, the retrieval time analysis yields the expected value, not worst case.

Given N^p points uniformly and independently distributed over an $N \times N$ region, where the position of each point is randomly selected from a uniform distribution, we determine the expected retrieval time of the half-plane structure.

Lemma 2.2 assumes that the points are on an integer grid, so it no longer applies. The analysis of the time to retrieve the interior of each slice does not assume that the points are on integral coordinates; it remains valid. We analyze the expected time to retrieve the frontier of each slice.

Assume that the points in the frontier rectangles are individually retrieved. The expected time to retrieve a rectangle is proportional to the average number of points in the rectangle, which is $\frac{1}{2}N^{p-(\alpha+2\beta)}$.

This modifies the complexity given in Theorems 2.1 and 2.4 to yield an expected complexity of

$$[\rho, \mathbf{E}(t)] = [O(N^{\alpha} \log N^{\alpha+\beta}), O(N^{\beta} \log N^{\alpha+\beta}) + O(N^{p-(\alpha+\beta)})].$$

If we let $\alpha = \beta = (p - \alpha - \beta)$ this yields

$$[\rho, \mathbf{E}(t)] = [O(N^{p/3} \log N), O(N^{p/3} \log N)].$$

Note that the complexity is dependent on the number of points, not N.

When $\alpha = \beta = \frac{2}{3}$ and $p = 2$, then as expected

$$[\rho, \mathbf{E}(t)] = [O(N^{2/3} \log N), O(N^{2/3} \log N)].$$

The other data structures presented above are analyzed in a similar manner [Katz 85]. In each case, the storage redundancy is unchanged from that of the grid problem, and the expected retrieval time is less than the worst case retrieval time derived for the grid problem. For instance, the expected retrieval time for a triangle is

$$O(N^{(1-\lambda)(2-\beta)}) + O(N^{(1-\lambda)\beta+\lambda} \log N) + O(N^{p-\lambda+\beta-(1-\lambda)(\alpha+2\beta)}).$$

With appropriate values for the parameters, this yields

$$[\rho, \mathbf{E}(t)] = [O(N^{p/3} \log N), O(N^{p/3} \log N)].$$

7. The dynamic data structures. Thus far, we have only discussed *update* and *retrieve* operations. Further, we have assumed that all points are present or initialized to 0. This is the simplest case of a dynamic data structure, since the values being retrieved can change, but the structure in which they reside can not. In this chapter, we will examine two other ways in which a data structure can be said to be dynamic:

1. Allow the insertion and deletion of points whose keys are in a grid of known size.
2. Allow the insertion and deletion of points with arbitrary (integral) keys.

In both of these versions, we start with an empty grid and enter points by *inserts*. We may also *delete* a point or we may *update* the value attached to a point. We define modifications to the half-plane data structure which permit the data structure to represent these dynamic structures.

7.1. Keys in a grid of known size. We approach the first form of the problem by abstractly generating a version of the data structure in which all values are initially 0. Portions of the data structure are actually generated only as needed by operations.

The first insertion generates all of the rotations, but only those parts of each rotation necessary to represent the first point (i.e., only one slice and only one rectangle) are filled in. Subsequent insertions extend the data structure within each rotation to support the new point. Retrievals are as in the static structure.

Each deletion removes a single point from the data structure. Following Fredman, we assume that semi-group variables can only have values assigned once. When deleting a point, all $O(\log N)$ variables, in the retrieval structure from the point to the root of the data structure in each rotation must be replaced. *Updates* may be performed as a deletion followed by an insertion. This form of dynamic structure costs no more per operation than the static structure.

7.2. Arbitrary integral keys. In the second form of dynamic data structure, we are trying to solve the same problem; but we do not know how large the space is which contains the points. We guess N, the size of the grid and build the data structure. The initial guess for N is small and the data structure is adjusted as *updates* are made.

We can convert this problem into a decomposable problem by noting that we can assume N to be a power of 2 with only a constant factor increase in complexity. We can then use any of the methods described in [Bentley 79], [Bentley and Saxe 80], or [Overmars and van Leeuwen 81] to convert the static data structure into the dynamic one.

Since the size of the half-space data structure is at least doubled each time and the final structure is no more than twice the optimal size, the maximum number of times that a new structure must be built is $O(\log N)$. The worst case complexity of the data structure is $[\rho, t] = [O(N^{2-2\beta} \log^2 N), O(N^{\beta} \log^2 N)]$ per operation. Because the increase in size is exponential, the amortized complexity of the dynamic structure is no worse than four times the amortized complexity of the static structure.

7.3. Other data structures. Similar modifications can be applied to form dynamic versions of the other data structures discussed in this paper. The complexities resulting from the dynamic data structures are at most $O(\log N)$ worse than the complexities of the static case, even if N and p (in the nongrid problems) are not known in advance.

8. Summary. We have presented families of data structures which permit efficient geometric retrieval. The structures permit retrieval of the sum of the values associated with points in a half-plane or polygon. The points may be at integral coordinates or randomly distributed. We have also described the behavior of the data structures when used dynamically. The families of data structures permit storage redundancy (update time) to be traded off against retrieval time.

Appendix A. Bound on vertices in a convex polygon. In order to achieve a worst case bound on the retrieval time for a circle on the square grid (see § 4), we must have an upper bound on the number of vertices. L. L. Larmore [Larmore 84] provided a sketch of the proof presented below:

THEOREM A.1. *The number of vertices in a convex polygon whose vertices are at integral coordinates is $O(R^{2/3})$ where R is one half of the maximum distance between two vertices.*

Proof. Since the polygon is convex, each of its edges (taken in the clockwise orientation) has a different slope. If each of the edges is treated as a vector and translated to a common origin, each vector must terminate at a different integral position. The sum of the lengths of the translated vectors is between $4R$ and $2\pi R$. The smallest average length of the translated vectors occurs if they are packed into a circle with the smallest possible radius. In order to have a sufficiently large sum of lengths, such a circle must have a radius of $\Omega(R^{1/3})$. The average length of the vectors packed into a circle of this radius is $\Omega(R^{1/3})$. The number of vertices is the perimeter of the polygon divided by the average length of the vectors. Therefore, the number of vertices is limited to $O(R^{2/3})$. □

Acknowledgment. We wish to thank the referees for their extremely helpful suggestions for improving this paper.

REFERENCES

[Avis 84] D. AVIS, *Non-partitionable point sets*, Inform. Proc. Lett., 19 (1984), pp. 125-129.

[Bentley 79] JON LOUIS BENTLEY, *Decomposable searching problems*, Inform. Proc. Lett., 8 (1979), pp. 244-251.

[Bentley and Saxe 80] JON LOUIS BENTLEY AND JAMES B. SAXE, *Decomposable searching problems # 1: Static to dynamic transformations*, J. Algorithms, 1 (1980), pp. 302-358.

[Burkhard, Fredman and Kleitman 81] WALTER A. BURKHARD, MICHAEL L. FREDMAN AND DANIEL J. KLEITMAN, *Inherent complexity trade-offs for range query problems*, Theoretical Computer Science, 16 (1981), pp. 279-290. North-Holland, Amsterdam.

[Cole and Yap 83] RICHARD COLE AND CHEE K. YAP, *Geometric retrieval problems*, Proc. of the 1983 IEEE Symposium on Foundations of Computer Science, pp. 112-121.

[Cole 85] RICHARD COLE, *Partitioning points in 4-space*, Proc. ICALP, to appear.

[Edelsbrunner and Welzl 83] H. EDELSBRUNNER AND E. WELZL, *Half-planar range space in linear space and $O(N^{.695})$ time*, Univ. Graz, Technical Report F111, 1983.

[Fredman 80] MICHAEL L. FREDMAN, *The inherent complexity of dynanic data structures which accommodate range queries*, Proc. of the 1980 IEEE Symposium on Foundations of Computer Science, pp. 191–199.

[Fredman 82] MICHAEL L. FREDMAN, *The complexity of maintaining an array and computing its partial sums*, J. Assoc. Comput. Mach., 29 (1982), pp. 250–260.

[Katz 85] MARTIN DAVID KATZ, *Geometric retrievals: data structures and computational complexity*, Ph.D. dissertation, Information and Computer Science, University of California, Irvine, CA, April 1985.

[Larmore 84] LAURENCE L. LARMORE, *Personal communication*, 1984.

[Lueker 78] GEORGE S. LUEKER, *A data structure for orthogonal range queries*, Proc. of the 1978 IEEE Symposium on Foundations of Computer Science.

[Lueker and Willard 82] GEORGE S. LUEKER AND DAN E. WILLARD, *A data structure for dynamic range queries*, Inform. Proc. Lett., 15 (1982), pp. 209–213.

[Megiddo 84] N. MEGIDDO, *Partitioning points with 2 lines in the planes*, this Journal, to appear.

[Overmars and van Leeuwen 81] M. H. OVERMARS AND J. VAN LEEUWEN, *Two general methods for dynamizing decomposable searching problems*, Computing, 26 (1981), pp. 155–166.

[Shamos 78] MICHAEL IAN SHAMOS, *Computational geometry*, Ph.D. dissertation, Yale Univ., New Haven, CT, 1978.

[Willard 78] DAN E. WILLARD, *New data structures for orthogonal queries*, Technical Report TR-20-78, Center for Research in Computing Technology, Harvard Univ., Cambridge, MA, 1978.

[Willard 82] ———, *Polygon retrieval*, this Journal, 11 (1982), pp. 149–165.

[Willard 85] ———, *New data structures for orthogonal range queries*, this Journal, 11 (1982), pp. 232–253.

[Yao 83] F. FRANCES, YAO, *A 3-space partition and its applications*, Proc. of the 15th ACM Symposium on Theory of Computing, 1983, pp. 258–263.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
003

COMMUNICATION COMPLEXITY OF COMPUTING THE HAMMING DISTANCE*

KING F. PANG† AND ABBAS EL GAMAL‡

Abstract. Let $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$. Persons A and B are given $\mathbf{x}$ and $\mathbf{y}$ respectively. They communicate in order that both find the Hamming Distance $d_H^n(\mathbf{x},\mathbf{y})$. Three communication models, viz, deterministic, ε-error and ε-randomized, are considered. Let $C(d_H^n)$, $C_\varepsilon(d_H^n)$ and $D_\varepsilon(d_H^n)$ be the respective minimum number of bits that must be communicated under the three models. It is shown that

$$n+\log(n+1-\sqrt{n}) \leqq C(d_H^n) \leqq n+\lceil \log(n+1) \rceil.$$

It is also shown that both $C_\varepsilon(d_H^n)$ and $D_\varepsilon(d_H^n)$ are lower bounded by $\Omega(n)$, thus solving an open problem posed by Yao.

Key words. communication complexity, randomized protocol, Hamming distance, combinatorial extremal problem

AMS(MOS) subject classifications. 05, 68

1. Introduction. Let $\mathscr{X}$, $\mathscr{Y}$ and $\mathscr{Z}$ be three finite sets and $f: \mathscr{X} \times \mathscr{Y} \to \mathscr{Z}$. Person A is given $\mathbf{x} \in \mathscr{X}$ and person B is given $\mathbf{y} \in \mathscr{Y}$. They communicate according to an agreed-upon protocol, with the objective of computing $f(\mathbf{x},\mathbf{y})$. We consider three communication models which differ in the types of protocols employed and the level of correctness of the computation.

(i) *Deterministic model*: When A (or B) transmits, his message is a function of $\mathbf{x}$ (or $\mathbf{y}$) and all the previous messages. When the communication terminates, both A and B are required to know the correct value of $f(\mathbf{x},\mathbf{y})$, for all $(\mathbf{x},\mathbf{y}) \in \mathscr{X} \times \mathscr{Y}$. $C(f)$ is the minimum (over all deterministic protocols that satisfy the error-free requirement) number of bits communicated under the worst case input.

(ii) *ε-error model*: The ε-error model is deterministic in the sense of (i). However, when it terminates, both A and B are allowed to arrive at an incorrect value of $f(\mathbf{x},\mathbf{y})$, for as many as $\varepsilon \cdot \|\mathscr{X}\| \cdot \|\mathscr{Y}\|$ (arbitrary) pairs $(\mathbf{x},\mathbf{y}) \in \mathscr{X} \times \mathscr{Y}$. The ε-error communication complexity of f, $C_\varepsilon(f)$, is then similarly defined as $C(f)$, where the minimization is over all deterministic protocols satisfying the ε-error requirement. With a uniform density on $\mathscr{X} \times \mathscr{Y}$, the average case ε-error complexity $\bar{C}_\varepsilon(f)$ can also be defined.

(iii) *ε-randomized model*: When A (or B) transmits, he chooses randomly from a set of messages. The messages in this set and the probability density on it are specified by $\mathbf{x}$ (or $\mathbf{y}$) and the messages already transmitted. The error requirement is the following: averaged over all the possible sequences of messages sent during the communication, *for all* inputs $(\mathbf{x},\mathbf{y}) \in \mathscr{X} \times \mathscr{Y}$, the probability that the end result is different from $f(\mathbf{x},\mathbf{y})$ is no more than a constant $0 \leqq \varepsilon \leqq 1$. With a uniform density on $\mathscr{X} \times \mathscr{Y}$, let $D_P(f)$ be the average (over all random outcomes) number of bits communicated in protocol P averaged over all inputs. The ε-randomized communication complexity of f, $D_\varepsilon(f)$, is then defined as the minimum of $D_P(f)$, over all protocols that satisfy the ε-error condition.

* Received by the editors May 29, 1984, and in revised form July 2, 1985.

† Information Systems Laboratory, Electrical Engineering Department, Stanford University, Stanford, California 94305. The work of this author was partially supported by the National Science Foundation under contract 80-26102.

‡ Information Systems Laboratory, Electrical Engineering Department, Stanford University, Stanford, California 94305. The work of this author was partially supported by the Defense Advanced Research Projects Agency under contract MDA-0680 and by the U.S. Air Force under contract F49620-79C-0058.

In this paper, we examine the communication complexity of the Hamming distance function according to the three models. The Hamming distance between $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ is defined as

$$d_H^n(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} 1(x_i \neq y_i),$$

where $1(\cdot)$ is the indicator function. When the lengths of $\mathbf{x}$ and $\mathbf{y}$ are not explicitly specified, the notation $d_H(\mathbf{x}, \mathbf{y})$ is used for their Hamming distance. In § 2, we consider the deterministic model. Section 2.1 contains the formal definition of a deterministic protocol and upper and lower bounds to $C(f)$ for an arbitrary function f are shown in § 2.2 (Theorems 2.1 and 2.2). In § 2.3, we prove a lower bound on $C(d_H^n)$ that is at most one bit less than the upper bound (Theorem 2.3). As a by-product of this result, we solve an independently interesting two-family extremal combinatorial problem (Theorem 2.4 and Corollary 2.3). Section 3 is concerned with the ε-error model. We first formally define an ε-error protocol. The communication complexity (under all three models) of the Hamming distance function is then related to those of the inner product function and the parity of the inner product function (Lemma 3.2). By proving a lower bound on the ε-error communication complexity of the latter (Lemma 3.3), we prove an $\Omega(n)$ lower bound for $C_\varepsilon(d_H^n)$ (Theorem 3.1) and $\bar{C}_\varepsilon(d_H^n)$ (Theorem 3.2). In § 4, we combine the result of § 3 and a Theorem of Yao [Yao1] to show that $D_\varepsilon(d_H^n) = \Omega(n)$ (Theorem 4.1), thus solving one of the open problems posed in [Yao3].

2. Deterministic model.

2.1. Formal definition. To formally define a deterministic protocol for f, we need the following definitions:

DEFINITION 2.1 [Yao2]. A *monochromatic rectangle* (*m-rect*) is a product set $\mathcal{U} \times \mathcal{V}$ where $\mathcal{U} \subseteq \mathcal{X}$, $\mathcal{V} \subseteq \mathcal{Y}$ such that $f(\mathbf{a}, \mathbf{b})$ is constant for all $\mathbf{u} \in \mathcal{U}$ and $\mathbf{v} \in \mathcal{V}$. An $m(f)$-partition is a partition of $\mathcal{X} \times \mathcal{Y}$ into m-rect's and $k(f)$ is defined as the minimum number of m-rect's over all $m(f)$-partitions of $\mathcal{X} \times \mathcal{Y}$.

DEFINITION 2.2. Denote $\mathcal{X}$ and $\mathcal{Y}$ the *row-projection* and *column-projection* of the product set $\mathcal{X} \times \mathcal{Y}$. The pair of product sets $(\mathcal{X}' \times \mathcal{Y}', \mathcal{X}'' \times \mathcal{Y}'')$ is called a *row-partition* of $\mathcal{X} \times \mathcal{Y}$ if $\mathcal{X}' = \mathcal{X}'' = \mathcal{X}$ and $\mathcal{Y}', \mathcal{Y}''$ partition $\mathcal{Y}$. A *Column-partition* is defined similarly. A *decomposition tree* (d-tree) for $\mathcal{X} \times \mathcal{Y}$ is a binary tree whose nodes are product sets $\subseteq \mathcal{X} \times \mathcal{Y}$. Each internal node is the *disjoint* union of its children. The root of the tree is $\mathcal{X} \times \mathcal{Y}$ and the leaves are m-rect's of f. It is clear that since the tree is binary each node is either row- or column-partitioned by its children.

Given a d-tree for f, we label the children of each node "0" and "1" and associate with it a protocol P as follows: At each step of the communication A and B consider one node in the tree (the first node being the root). If the node is column-partitioned by its children, A transmits the label of the child whose row-projection contains $\mathbf{x}$. If the node is row-partitioned, B transmits the label of the node whose column-projection contains $\mathbf{y}$. Next, A and B move to the node whose label was transmitted and repeat the process. The communication terminates when they arrive at a leaf and obtain the value of $f(\mathbf{x}, \mathbf{y})$.

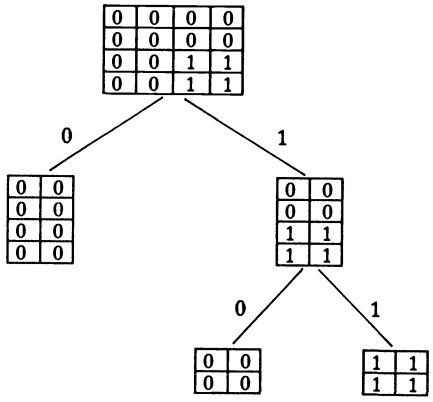
An easy induction (on the number of bits communicated) shows that at each step:

(i) A and B consider the same node of the tree.

(ii) This node contains $(\mathbf{x}, \mathbf{y})$.

(iii) If the node is internal then exactly one of its children contains $(\mathbf{x}, \mathbf{y})$.

An example of a d-tree for a function is shown in Fig. 2.1. Suppose in the d-tree for protocol P, the length of the (unique) path joining the root and the leaf containing



† $C_P = 2$ bits for this protocol.

FIG. 2.1. *An example of a protocol.*

$(\mathbf{x}, \mathbf{y})$ is $C_P(\mathbf{x}, \mathbf{y})$. The complexity of the communication protocol P is defined as

$$C_P(f) \triangleq \max_{(\mathbf{x},\mathbf{y}) \in \mathscr{X} \times \mathscr{Y}} C_P(\mathbf{x}, \mathbf{y}),$$

and the communication complexity of f is defined as

$$C(f) \triangleq \min_P C_P(f).$$

Remark. In our model, the transfer of information occurs in both directions, but *not* simultaneously. Since we are only concerned with the number of bits exchanged, it is straightforward to show that the lower bounds proved in this paper are not affected by this restriction.

2.2. General bounds for $C(f)$. A simple upper bound for $C(f)$ can be achieved by the following algorithm: Using $\lceil \log \|\mathscr{X}\| \rceil$[1] bits, A communicates $\mathbf{x}$ to B, who computes $f(\mathbf{x}, \mathbf{y})$. Another $\lceil \log \|\mathscr{Y}\| \rceil$ bits are then sufficient for B to inform A of the result. We therefore have the following theorems.

THEOREM 2.1. $C(f) \leqq \lceil \log \|\mathscr{X}\| \rceil + \lceil \log \|\mathscr{Y}\| \rceil$.

There are two general techniques for proving lower bounds for $C(f)$ for an arbitrary function f. The first one [Yao2] is based on $m(f)$-partitions of $\mathscr{X} \times \mathscr{Y}$. The other lower bound [MS] is obtained from the rank of the function table of f, which is being considered as an $\|\mathscr{X}\| \times \|\mathscr{Y}\|$ matrix. A statement of the first lower bound, according to our model, is the following:

THEOREM 2.2. $C(f) \geqq \lceil \log (k(f)) \rceil$.

Proof. The theorem follows from two properties of the d-tree corresponding to the protocol P:

(i) All the leaves of the d-tree are m-rect's (otherwise the result of the communication is not always correct).

(ii) The product set corresponding to the node of the d-tree is either row- or column-partitioned at each step of the protocol.

[1] All logarithms in this paper, unless otherwise specified, are of base 2.

By the definition of $k(f)$, every d-tree of f must have at least $k(f)$ leaves. Hence, the height of a d-tree is at least $\lceil \log (k(f)) \rceil$ and the theorem is proved. □

2.3. Upper and lower bounds for $C(d_H^n)$. In this section, we derive bounds for $C(d_H^n)$. The upper bound follows immediately as a corollary to Theorem 2.1:

$$C(d_H^n) \leqq n + \lceil \log (n+1) \rceil.$$

We next give a lower bound for $C(d_H^n)$, matching the upper bound up to smaller order terms, by a simple argument.

LEMMA 2.1. $C(d_H^n) \geqq n+1$.

Proof. For any $\mathbf{x} \in \{0,1\}^n$, $d_H^n(\mathbf{x}, \mathbf{x}) = 0$ and $d_H^n(\mathbf{x}, \bar{\mathbf{x}}) = n$, where $\bar{\mathbf{x}}$ is the complement of $\mathbf{x}$. Hence there are exactly 2^n m-rect's of Hamming Distance 0 and n respectively in any $m(d_H^n)$-decomposition of $\{0,1\}^n \times \{0,1\}^n$. Subsequently, $k(d_H^n) \geqq 2^{n+1}$ and by Theorem 2.2, $C_2(d_H^n) \geqq n+1$. □

This lower bound differs from the upper bound by $\lceil \log (n+1) \rceil - 1$ bits. A better lower bound, differing from the upper bound by no more than 1 bit, is stated in the following theorem.

THEOREM 2.3. $C_2(d_H^n) \geqq n + \lceil \log (n+1-\sqrt{n}) \rceil$.

Proof. We lower bound $k(d_H^n)$ by upper bounding the sizes of all the m-rect's in the function table. For $0 \leqq \delta \leqq n$, define

$$S_\delta^n \triangleq \{\mathcal{U} \times \mathcal{V} \subseteq \{0,1\}^n \times \{0,1\}^n : d_H^n(\mathbf{u}, \mathbf{v}) = \delta \text{ for all } \mathbf{u} \in \mathcal{U} \text{ and } \mathbf{v} \in \mathcal{V}\},$$

$$M(n, \delta) \triangleq \max \{\|\mathcal{U}\| \cdot \|\mathcal{V}\| : \mathcal{U} \times \mathcal{V} \in S_\delta^n\}.$$

In Lemma 2.2, we establish the fact $M(n, \delta) = M(n, n-\delta)$ by showing that for every m-rect $\in S_\delta^n$, there exists an m-rect $\in S_{n-\delta}^n$ with equal size. This reduces the task of upper bounding $M(n, \delta)$ to the range $0 \leqq \delta \leqq \lfloor n/2 \rfloor$. We then prove in Theorem 2.4 the crucial result that for $n = 2, 3, 4 \cdots$; $\delta = 0, 1, \cdots, \lfloor n/2 \rfloor$,

$$\frac{M(n, \delta)}{M(n-2, \delta-1)} \leqq \max \left[4, \frac{n(n-1)}{\delta(n-\delta)} \right].$$

As corollaries to Theorem 2.4, we show that

$$M(n, \delta) \leqq \begin{cases} \binom{n}{\delta} & \text{for all } n \text{ and } \delta < n/2 - \sqrt{n/4}, \\ \prod_{j=0}^{\delta-1} \dfrac{(n-2j)^2}{(\delta-j)(n-\delta-j)} & \text{for } n \geqq 4 \text{ and } \lfloor n/2 - \sqrt{n/4} \rfloor \leqq \delta \leqq \lfloor n/2 \rfloor. \end{cases} \tag{2.1}$$

Now denote

$$N(n, \delta) \triangleq \|\{(\mathbf{x}, \mathbf{y}) \in \{0,1\}^n \times \{0,1\}^n : d_H^n(\mathbf{x}, \mathbf{y}) = \delta\}\| \quad \text{for } 0 \leqq \delta \leqq n$$

$$= 2^n \cdot \binom{n}{\delta}.$$

It is clear that

$$k(d_H^n) \geqq \sum_{\delta=0}^{n} \frac{N(n, \delta)}{M(n, \delta)}$$

$$= 2 \cdot \sum_{\delta=0}^{\lceil n/2 - \sqrt{n/4} \rceil - 1} \frac{N(n, \delta)}{M(n, \delta)} + \sum_{\delta = \lceil n/2 - \sqrt{n/4} \rceil}^{\lceil n/2 + \sqrt{n/4} \rceil} \frac{N(n, \delta)}{M(n, \delta)}$$

$$\triangleq S_1 + S_2$$

where S_1 and S_2 are respectively the values of the first and second summations. From the first inequality of (2.1)

$$S_1 \geqq 2 \cdot \sum_{\delta=0}^{\lceil n/2-\sqrt{n/4}\rceil-1} \frac{2^n \cdot \binom{n}{\delta}}{\binom{n}{\delta}} = 2n \cdot \lceil n-\sqrt{n}\rceil.$$

On the other hand, as we show in Appendix 1,

$$S_2 \geqq 2^n \quad \text{for } n \geqq 4,$$

therefore $k(d_H^n) \geqq 2^n \lceil n+1-\sqrt{n}\rceil$ for $n \geqq 4$. For $1 \leqq n \leqq 3$, $k(d_H^n)$ can be verified from the Hamming Distance Function tables to be 4, 10 and 32 respectively, which still satisfy the lower bound above. Since $C_2(d_H^n) \geqq \lceil \log (k(d_H^n))\rceil$, we obtain

$$C_2(d_H^n) \geqq n + \lceil \log (n+1-\sqrt{n})\rceil. \qquad \square$$

We are ready to state and prove Lemma 2.2 and Theorem 2.4.

LEMMA 2.2. *$M(n, \delta) = M(n, n-\delta)$ for $\delta = 0, 1, \cdots, n$.*

Proof. Since for any $\mathbf{x}$ and $\mathbf{y} \in \{0,1\}^n$,

$$d_H^n(\mathbf{x}, \mathbf{y}) = \delta \Rightarrow \begin{cases} d_H^n(\bar{\mathbf{x}}, \bar{\mathbf{y}}) = \delta, \\ d_H^n(\bar{\mathbf{x}}, \mathbf{y}) = n-\delta, \\ d_H^n(\mathbf{x}, \bar{\mathbf{y}}) = n-\delta, \end{cases}$$

the following are equivalent:

1) $\mathcal{U} \times \mathcal{V} \in S_\delta^n$,
2) $\bar{\mathcal{U}} \times \bar{\mathcal{V}} \in S_\delta^n$,
3) $\bar{\mathcal{U}} \times \mathcal{V} \in S_{n-\delta}^n$,
4) $\mathcal{U} \times \bar{\mathcal{V}} \in S_{n-\delta}^n$.

Since $\|\mathcal{U}\| = \|\bar{\mathcal{U}}\|$ and $\|\mathcal{V}\| = \|\bar{\mathcal{V}}\|$, the lemma is proved. $\square$

This lemma shows that the analysis of $M(n, \delta)$ can be reduced to the range $0 \leqq \delta \leqq \lfloor n/2\rfloor$. The basis for upper bounding $M(n, \delta)$ for $\delta = 1, 2, \cdots, \lceil n/2\rceil$ is provided by the following theorem.

THEOREM 2.4. *For $n = 3, 4 \cdots$, $\delta = 1, 2, \cdots, \lfloor n/2\rfloor$,*

$$\frac{M(n, \delta)}{M(n-2, \delta-1)} \leqq \max\left(4, \frac{n(n-1)}{\delta(n-\delta)}\right).$$

Proof. We first introduce some notation: For a set $C \subseteq \{0,1\}^n$ and $\varepsilon \in \{0,1\}$,

i) The ith bit of $c \in C$ is denoted by c_i.

ii) $C_\varepsilon^t \triangleq \{(c_1, \cdots, c_n) \in C: c_t = \varepsilon\} \subseteq \{0,1\}^n$.

iii) $C_\varepsilon^{*t} \triangleq \{(c_1, \cdots, c_{t-1}, c_{t+1}, \cdots, c_n): (c_1, \cdots, c_{t-1}, \varepsilon, c_{t+1}, \cdots, c_n) \in C\} \subseteq \{0,1\}^{n-1}$.

iv) Analogously, for two components s, t we define $C_{\varepsilon\eta}^{st} \subseteq \{0,1\}^n$ and $C_{\varepsilon\eta}^{*st} \subseteq \{0,1\}^{n-2}$.

v) For $a \in \{0,1\}^i$ and $\mathbf{b} \in \{0,1\}^j$, $\mathbf{ab} \in \{0,1\}^{i+j}$ represents the concatenation of $\mathbf{a}$ and $\mathbf{b}$.

Consider $A \times B \in S_\delta^n$. Construct $\mathcal{U} \subseteq \{0,1\}^{n(n-1)/2}$ from A by the following procedure:

i) Let $\Gamma \triangleq \{(i,j): i = 1, 2, \cdots, n-1; j = i+1, \cdots, n\}$, i.e. the set of pairs of distinct indices between 1 and n. Clearly $\nu \triangleq \|\Gamma\| = n(n-1)/2$. Order the pairs lexicographically and denote the kth pair as (k_i, k_j).

ii) Map every $\mathbf{a} \in A$ to a $\mathbf{u} \in \mathcal{U}$ such that for $k = 1, \cdots, \nu$,

$$u_k = \begin{cases} 1 & \text{if } (a_{k_i}, a_{k_j}) = (0,1) \text{ or } (1,0), \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, $\mathcal{V} \subseteq \{0,1\}^{n(n-1)/2}$ is constructed from B.

FACT 1. $\forall \mathbf{u} \in \mathcal{U}$ *and* $\mathbf{v} \in \mathcal{V}$: $d_H^n(\mathbf{u}, \mathbf{v}) = \delta(n-\delta)$.

Proof (*of Fact* 1). There is a one-to-one correspondence between each pair $(\mathbf{u}, \mathbf{v}) \in \mathcal{U} \times \mathcal{V}$ and $(\mathbf{a}, \mathbf{b}) \in A \times B$. $\mathbf{u}$ and $\mathbf{v}$ differ in a certain bit position iff there is exactly one component different in the corresponding pairs of bits from $\mathbf{a}$ and $\mathbf{b}$. The number of such pairs is $\delta(n-\delta)$. □

FACT 2. *Let*

$$\lambda_i \triangleq \frac{\|\{\mathbf{u} \in \mathcal{U}: u_i = 1\}\|}{\|\mathcal{U}\|} \quad \textit{and} \quad \mu_i \triangleq \frac{\|\{\mathbf{v} \in \mathcal{V}: v_i = 1\}\|}{\|\mathcal{V}\|}.$$

$\exists 1 \leqq k \leqq n(n-1)/2$ *such that either*:

$$\lambda_k \cdot \overline{\mu_k} \geqq \frac{\delta(n-\delta)}{n(n-1)}, \tag{2.2}$$

or

$$\overline{\lambda_k} \cdot \mu_k \geqq \frac{\delta(n-\delta)}{n(n-1)}. \tag{2.3}$$

Proof (*of Fact* 2). Define $\sigma_k \triangleq \|\{(\mathbf{u}, \mathbf{v}) \in \mathcal{U} \times \mathcal{V}: u_k \neq v_k\}\|$. Clearly, $\sigma_k = \|\mathcal{U}\| \cdot \|\mathcal{V}\| \cdot (\lambda_k \bar{\mu}_k + \bar{\lambda}_k \mu_k)$. On the other hand,

$$\begin{aligned} \sum_{k=1}^{n(n-1)/2} \sigma_k &= \sum_{\mathbf{u} \in \mathcal{U}} \sum_{\mathbf{v} \in \mathcal{V}} \|\{(u_k, v_k): u_k \neq v_k\}\| \\ &= \sum_{\mathbf{u} \in \mathcal{U}} \sum_{\mathbf{v} \in \mathcal{V}} d_H^n(\mathbf{u}, \mathbf{v}) \\ &= \|\mathcal{U}\| \cdot \|\mathcal{V}\| \cdot \delta(n-\delta). \end{aligned}$$

By the pigeon hole principle, there must exist an index k such that

$$\sigma_k \geqq \frac{2\delta(n-\delta)}{n(n-1)} \cdot \|\mathcal{U}\| \cdot \|\mathcal{V}\|$$

and one of the two terms making up σ_k must be no less than half of this value. □

Without loss of generality, assume (2.2) is satisfied by the specific value of k and that $k_i = 1$ and $k_j = 2$. To construct an m-rect $\in S_{\delta-1}^{n-2}$, we consider the following cases.

Case 1. $A_{10}^{12} \cup A_{01}^{12} = A$ **and** $B_{00}^{12} \cup B_{11}^{12} = B$: If $\mathbf{w} \in A_{01}^{*12}$ (resp. B_{00}^{*12}), then either $\mathbf{w} \in A_{10}^{*12}$ (resp. B_{11}^{*12}), or $10\mathbf{w}$ (resp. $11\mathbf{w}$) can be appended to A (resp. B), and this only increases $\|A\| \cdot \|B\|$. We can therefore assume $A_{10}^{*12} = A_{01}^{*12}$ and $B_{00}^{*12} = B_{11}^{*12}$. Define $C \triangleq A_{01}^{*12}$ or A_{10}^{*12} and $D \triangleq B_{00}^{*12}$ or B_{11}^{*12}. Clearly, $C \times D \in S_{\delta-1}^{n-2}$ and we have shown that

$$\begin{aligned} \|A\| \cdot \|B\| &= 4 \cdot \|C\| \cdot \|D\| \\ &\leqq \max\left(4, \frac{n(n-1)}{\delta(n-\delta)}\right) \cdot M(n-2, \delta-1). \end{aligned}$$

Case 2a. $A_{10}^{12} \cup A_{01}^{12} = A$ **and** $B_{00}^{12} \cup B_{11}^{12} \subset B$: If $\mathbf{w} \in B_{00}^{*12}$, then either $\mathbf{w} \in B_{11}^{*12}$, or $11\mathbf{w}$ can be appended to B which only increases $\|A\| \cdot \|B\|$. We can therefore assume that $B_{00}^{*12} = B_{11}^{*12}$. On the other hand, if $\mathbf{z} \in A_{01}^{*12}$ then $\mathbf{z} \notin A_{10}^{*12}$, i.e. $A_{01}^{*12} \cap A_{10}^{*12} = \varnothing$. Consider the following two cases:

a) $\delta(n-\delta)/n(n-1) \geqq 4$: Recall from the proof of Fact 2 that we have $\lambda_k \bar{\mu}_k + \bar{\lambda}_k \mu_k \geqq 2\delta(n-\delta)/n(n-1)$. $A_{10}^{12} \cup A_{01}^{12} = A$ implies that $\lambda_k = 1$ and therefore $\|A_{10}^{*12} \cup A_{01}^{*12}\| \cdot \|B_{00}^{*12}\| \geqq \delta(n-\delta)/n(n-1) \cdot \|A\| \cdot \|B\|$. Define $C \triangleq A_{10}^{*12} \cup A_{01}^{*12}$ and $D \triangleq B_{00}^{*12}$. Clearly, $C \times D \in S_{\delta-1}^{n-2}$ and we have shown that

$$\|A\| \cdot \|B\| = \frac{n(n-1)}{\delta(n-\delta)} \cdot \|C\| \cdot \|D\|$$

$$\leqq \max\left(4, \frac{n(n-1)}{\delta(n-\delta)}\right) \cdot M(n-2, \delta-1).$$

b) $\delta(n-\delta)/n(n-1) < 4$: We can construct an m-rect $P \times Q \in S_\delta^n$ from $A \times B$ such that $\|P \times Q\| > \|A \times B\|$. Specifically,

$$P \triangleq A \cup (\{(0,1)\} \times A_{10}^{*12}) \cup (\{(1,0)\} \times A_{01}^{*12}),$$

$$Q \triangleq B_{00}^{12} \cup B_{11}^{12}.$$

Note that

$$\|P\| \cdot \|Q\| \geqq 2 \cdot \|A\| \cdot \|B\| \cdot \frac{2\delta(n-\delta)}{n(n-1)} > \|A\| \cdot \|B\|.$$

Next, define $C \triangleq P_{01}^{*12}$ or P_{10}^{*12} and $D \triangleq Q_{00}^{*12}$ or Q_{11}^{*12}. Clearly, $C \times D \in S_{\delta-1}^{n-2}$ and that

$$\|A\| \cdot \|B\| \leqq \|P\| \cdot \|Q\| = 4 \cdot \|C\| \cdot \|D\| \leqq \max\left(4, \frac{n(n-1)}{\delta(n-\delta)}\right) \cdot M(n-2, \delta-1).$$

Case 2b. $A_{10}^{12} \cup A_{01}^{12} \subset A$ **and** $B_{00}^{12} \cup B_{11}^{12} = B$: The argument used in Case 2a is symmetric between A and B. Therefore we obtain the same upper bound on $\|A\| \cdot \|B\|$.

Case 3. $A_{01}^{12} \cup A_{10}^{12} \subset A$ **and** $B_{00}^{12} \cup B_{11}^{12} \subset B$: We observe that $A_{01}^{*12} \cap A_{10}^{*12} = \varnothing$ and $B_{00}^{*12} \cap B_{11}^{*12} = \varnothing$. Consider the following two cases.

a) $\delta(n-\delta)/n(n-1) \geqq 4$: Define $C \triangleq A_{01}^{*12} \cup A_{10}^{*12}$ and $D \triangleq B_{00}^{*12} \cup B_{11}^{*12}$. It is clear that $C \times D \in S_{\delta-1}^{n-2}$ and

$$\|A\| \cdot \|B\| = \frac{n(n-1)}{\delta(n-\delta)} \|C\| \cdot \|D\|$$

$$\leqq \max\left(4, \frac{n(n-1)}{\delta(n-\delta)}\right) \cdot M(n-2, \delta-1).$$

b) $\delta(n-\delta)/n(n-1) < 4$: Define

$$P \triangleq A_{01}^{12} \cup A_{10}^{12} \cup (\{(0,1)\} \times A_{10}^{*12}) \cup (\{(1,0)\} \times A_{01}^{*12}),$$

$$Q \triangleq B_{00}^{12} \cup B_{11}^{12} \cup (\{(0,0)\} \times B_{11}^{*12}) \cup (\{(1,1)\} \times B_{00}^{*12}).$$

Now $\|P\| \cdot \|Q\| \geqq 4 \cdot \|A\| \cdot \|B\| \cdot \delta(n-\delta)/n(n-1) > \|A\| \cdot \|B\|$. Define $C \triangleq P_{01}^{*12}$ or P_{10}^{*12} and $D \triangleq Q_{00}^{*12}$ or Q_{11}^{*12}. Clearly, $C \times D \in S_{\delta-1}^{n-2}$.

$$\|A\| \cdot \|B\| \leqq \|P\| \cdot \|Q\|$$

$$= 4 \cdot \|C\| \cdot \|D\|$$

$$\leqq \max\left(4, \frac{n(n-1)}{\delta(n-\delta)}\right) \cdot M(n-2, \delta-1).$$

Hence in all the four cases, we have succeeded in proving that

$$\frac{M(n,\delta)}{M(n-2,\delta-1)} \leqq \max\left(4, \frac{n(n-1)}{\delta(n-\delta)}\right). \qquad \square$$

The proofs of the following three corollaries are given in Appendix 2.

COROLLARY 2.1. *For* $\delta < \lceil n/2-\sqrt{n/4}\rceil$ *and* $\delta > \lfloor n/2+\sqrt{n/4}\rfloor$, $M(n,\delta) = \binom{n}{\delta}$.

COROLLARY 2.2. *For* $n/2-\sqrt{n/4} \leqq \delta \leqq n/2+\sqrt{n/4}$,

$$M(n,\delta) \leqq \prod_{j=0}^{\delta'-1} \frac{(n-2j)^2}{(\delta'-j)(n-\delta'-j)},$$

where

$$\delta' = \begin{cases} \delta & \text{for } \delta \leqq \lfloor n/2\rfloor, \\ \lfloor n/2\rfloor - \delta & \text{otherwise.} \end{cases}$$

COROLLARY 2.3. *For* $n = 1, 2, \cdots$, $\max_{0\leqq\delta\leqq n} M(n,\delta) = 2^n$ *and the maximum is achieved by* $\delta = \lfloor n/2\rfloor$ *and* $\lceil n/2\rceil$.

The last corollary, being a special case of Theorem 2.4, was previously derived using a less general argument and reported in [AEP].

3. The ε-error model.

3.1. Definitions and general lower bound. In the ε-error model, there is still a one-to-one correspondence between a protocol and a binary tree, which we call an ε-tree. An ε-tree is nearly identical to a d-tree defined in § 2.1, except that since errors are allowed by an ε-error protocol, the leaves of an ε-tree are no longer necessarily m-rect's. Each leaf is now a product set $A \times B \subseteq \mathcal{X} \times \mathcal{Y}$ such that most of its elements yield the same function value. The following definitions parallel those in § 2.1.

DEFINITION 3.1. Given a function $f: \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$, a *q-monochromatic rectangle* (abbreviated as q-rect) with error ε is a pair $(\mathcal{U} \times \mathcal{V}, \mathbf{z})$, where $\mathcal{U} \times \mathcal{V} \in \mathcal{X} \times \mathcal{Y}$ and $\mathbf{z} \in \mathcal{Z}$, such that $f(\mathbf{u}, \mathbf{v}) = \mathbf{z}$ for at least $(1-\varepsilon) \cdot \|\mathcal{U}\| \cdot \|\mathcal{V}\|$ pairs $(\mathbf{u}, \mathbf{v}) \in \mathcal{U} \times \mathcal{V}$. We denote the size of the largest q-rect with error ε by $M_\varepsilon(f)$. An $m_\varepsilon(f)$-partition is a partition of $\mathcal{X} \times \mathcal{Y}$ into q-rect's S_i with error ε_i, where $i = 1, \cdots, m_\varepsilon(f)$, such that

$$\sum_{i=1}^{m_\varepsilon(f)} \varepsilon_i \|S_i\| \leqq \varepsilon \|\mathcal{X}\| \cdot \|\mathcal{Y}\|.$$

We define $k_\varepsilon(f)$ as the minimum of $m_\varepsilon(f)$ over all $m_\varepsilon(f)$-partitions of $\mathcal{X} \times \mathcal{Y}$.

With these definitions, it is straightforward to pinpoint the differences between a d-tree and an ε-tree. In contrast to a d-tree which has m-rect's as its leaves, the leaves of an ε-tree are q-rect's. In addition, suppose there are k leaves in the tree, where the jth leaf has weight (i.e. the number of elements in it) ω_j and has error ε_j, then the following condition (which we shall refer to as the "ε-error requirement") must be satisfied:

$$\sum_{i=1}^{k} \varepsilon_i \omega_i \leqq \varepsilon \|\mathcal{X}\| \cdot \|\mathcal{Y}\|.$$

Let P be a protocol satisfying the ε-error requirement and $C_P(\mathbf{x}, \mathbf{y})$ be the depth of the leaf in the ε-tree representation of P to which $(\mathbf{x}, \mathbf{y})$ belongs. The ε-error communication complexity of f is defined as

$$C_\varepsilon(f) \triangleq \min_P \max_{(\mathbf{x},\mathbf{y})\in\mathcal{X}\times\mathcal{Y}} C_P(\mathbf{x}, \mathbf{y}).$$

With a uniform density on $\mathscr{X} \times \mathscr{Y}$, the average case ε-error communication complexity of f can also be defined:

$$\bar{C}_\varepsilon(f) \triangleq \frac{1}{\|\mathscr{X}\| \cdot \|\mathscr{Y}\|} \min_P \sum_{(\mathbf{x},\mathbf{y}) \in \mathscr{X} \times \mathscr{Y}} C_P(\mathbf{x}, \mathbf{y}).$$

The following lemma provides a lower bound for $C_\varepsilon(f)$ in terms of $M_\varepsilon(f)$.

LEMMA 3.1. $C_\varepsilon(f) \geqq \log \|\mathscr{X}\| + \log \|\mathscr{Y}\| - \log M_{2\varepsilon}(f) - 1$.

Proof. The proof of the first inequality is similar to that of Lemma 2.1. To prove the second inequality, consider the $m_\varepsilon(f)$-partition which achieves $m_\varepsilon(f) = k_\varepsilon(f)$. For convenience in notation, we shall abbreviate $k_\varepsilon(f)$ by k. Let S_i, $i = 1, \cdots, k$ be the q-rect's constructed. The ε-error requirement stipulates that

$$\sum_{i=1}^{k} \varepsilon_i \|S_i\| \leqq \varepsilon \|\mathscr{X}\| \|\mathscr{Y}\|,$$

which can be written as

$$\sum_{i:\, \varepsilon_i \leqq 2\varepsilon} \varepsilon_i \|S_i\| + \sum_{i:\, \varepsilon_i > 2\varepsilon} \varepsilon_i \|S_i\| \leqq \varepsilon \|\mathscr{X}\| \|\mathscr{Y}\|.$$

Clearly,

$$\sum_{i:\, \varepsilon_i \leqq 2\varepsilon} \|S_i\| \geqq (\|\mathscr{X}\| \cdot \|\mathscr{Y}\|)/2.$$

Suppose the number of q-rect's involved in the above sum is k'; we have

$$\frac{1}{k'} \sum_{i:\, \varepsilon_i \leqq 2\varepsilon} \|S_i\| \geqq (\|\mathscr{X}\| \cdot \|\mathscr{Y}\|)/(2k').$$

The left-hand side of the above equation is the average size of k' q-rect's. There must exist one q-rect S_i whose size is at least as large as the average, i.e.

$$\|S_i\| \geqq (\|\mathscr{X}\| \cdot \|\mathscr{Y}\|)/(2k').$$

Since $\|S_i\| \leqq M_{2\varepsilon}(f)$ and $k' \leqq k$, we have

$$M_{2\varepsilon}(f) \geqq (\|\mathscr{X}\| \cdot \|\mathscr{Y}\|)/(2k),$$

which gives the second inequality. □

There is a similar result for the average case complexity.

LEMMA 3.2. $\bar{C}_\varepsilon(f) \geqq (\log \|\mathscr{X}\| + \log \|\mathscr{Y}\| - \log M_{2\varepsilon}(f) - 1)/2$.

Proof. Let P be the protocol achieving $\bar{C}_\varepsilon(f)$. Consider the ε-tree representation of P. As there is no ambiguity, we also call this ε-tree P. Consider those leaves of this tree with no more than 2ε error. Without loss of generality, let them be the first m leaves of the tree and denote their weights by w_j, $1 \leqq j \leqq m$. We must have

$$W \triangleq \sum_{i=1}^{m} w_i \geqq (\|\mathscr{X}\| \cdot \|\mathscr{Y}\|)/2,$$

for if otherwise, the remaining leaves already violates the ε-error requirement. Clearly

$$\bar{C}_\varepsilon(f) = \bar{C}_P \geqq \frac{\sum_{j=1}^{m} l_j w_j}{\|\mathscr{X}\| \cdot \|\mathscr{Y}\|}.$$

By the entropy bound for the external path length of a binary tree,

$$\sum_{j=1}^{m} l_j w_j \geqq \sum_{j=1}^{m} w_j \cdot \log\left(\frac{W}{w_j}\right).$$

Since $w_j \leqq M_{2\varepsilon}(f)$ for all j, the left-hand side of the above inequality is at least

$$\sum_{j=1}^{m} \left(w_j \log \left(\frac{W}{M_{2\varepsilon}(f)} \right) \right) = W \log \left(\frac{W}{M_{2\varepsilon}(f)} \right).$$

Subsequently

$$\begin{aligned} \bar{C}_\varepsilon(f) &\geqq \frac{W}{\|\mathscr{X}\| \cdot \|\mathscr{Y}\|} \cdot \log \left(\frac{W}{M_{2\varepsilon}(f)} \right) \\ &\geqq (\log \|\mathscr{X}\| + \log \|\mathscr{Y}\| - \log M_{2\varepsilon}(f) - 1)/2. \end{aligned}$$

□

3.2. Lower bounds for $C_\varepsilon(d_H^n)$ and $\bar{C}_\varepsilon(d_H^n)$. Letting $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$, their *inner product* is defined as

$$f_I^n(\mathbf{x}, \mathbf{y}) \triangleq \sum_{i=1}^{n} x_i y_i$$

and the parity of $f_I^n(\mathbf{x}, \mathbf{y})$ is denoted by $f_{IP}^n(\mathbf{x}, \mathbf{y})$. We first relate $C_\varepsilon(d_H^n)$, $C_\varepsilon(f_I^n)$ and $C_\varepsilon(f_{IP}^n)$.

LEMMA 3.3. *Given* $0 \leqq \varepsilon < 1$, $C_\varepsilon(d_H^n) + 2\lceil \log(n+1) \rceil \geqq C_\varepsilon(f_I^n) \geqq C_\varepsilon(f_{IP}^n)$.

Proof. The second inequality is easily proved by noting that knowing the inner product, the parity of the inner product can always be computed. However when an erroneous value of $f_I^n(\mathbf{x}, \mathbf{y})$ is used to compute $f_{IP}^n(\mathbf{x}, \mathbf{y})$, the latter is not necessarily in error. Hence in order to compute $f_{IP}^n(\mathbf{x}, \mathbf{y})$, with error no more than ε, one can always first compute $f_I^n(\mathbf{x}, \mathbf{y})$ with the same designated error. To prove the first inequality, consider the different values that the pair (x_i, y_i) can take. Let

$$t_{1,1} \triangleq \sum_{i=1}^{n} 1(x_i = 1, y_i = 1).$$

Similarly, $t_{1,0}$, $t_{0,1}$ and $t_{0,0}$ are defined. We have the following relations:

(i) $d_H^n(\mathbf{x}, \mathbf{y}) = t_{0,1} + t_{1,0}$.

(ii) $\text{wt}(\mathbf{x}) = t_{1,1} + t_{1,0}$, where $\text{wt}(\mathbf{x})$ is the number of ones in $\mathbf{x}$.

(iii) $\text{wt}(\mathbf{y}) = t_{1,1} + t_{0,1}$.

(iv) $f_I^n(\mathbf{x}, \mathbf{y}) = t_{1,1}$.

It is easy to see that

$$(\text{wt}(\mathbf{x}) + \text{wt}(\mathbf{y}) - d_H^n(\mathbf{x}, \mathbf{y})) = 2f_I^n(\mathbf{x}, \mathbf{y}).$$

Hence knowing the weights of both $\mathbf{x}$ and $\mathbf{y}$, there is a one-to-one correspondence between the Hamming distance and the inner product. Since the weight of one sequence can be communicated to the other person in $\lceil \log(n+1) \rceil$ bits, we have the first inequality. □

Clearly, the argument also holds for the average case complexities. Thus

COROLLARY 3.1. $\bar{C}_\varepsilon(d_H^n) + 2\lceil \log(n+1) \rceil \geqq \bar{C}_\varepsilon(f_I^n) \geqq \bar{C}_\varepsilon(f_{IP}^n)$.

Finally, restricting Lemma 3.3 to the case $\varepsilon = 0$, we have the following relationship among the deterministic communication complexities of the three functions.

COROLLARY 3.2. $C(d_H^n) + 2\lceil \log(n+1) \rceil \geqq C(f_I^n) \geqq C(f_{IP}^n)$.

We next prove an upper bound for $M_\varepsilon(f_{IP}^n)$.

LEMMA 3.4. *For* $0 \leqq \varepsilon \leqq \frac{1}{8}$, $M_\varepsilon(f_{IP}^n) \leqq (1 + c\varepsilon) \cdot 2^n$, *where c is a constant dependent only on* ε.

Proof. Define $A(n)$, the *function table* for f_{IP}^n as a $2^n \times 2^n$ matrix, whose (i,j)th component is $f_{IP}^n(\mathbf{b}(i), \mathbf{b}(j))$ (where $\mathbf{b}(k)$ is the binary representation of $0 \leqq k \leqq 2^n - 1$). Consider $\{\mathbf{r}_i, i = 1, \cdots, 2^n\}$, the rows of $A(n)$ as a set of binary 2^n sequences. We have the following:

FACTS: (1) $\text{wt}(\mathbf{r}_1) = 0$ *and* $\text{wt}(\mathbf{r}_i) = 2^{n-1}$ *for* $1 < i \leqq 2^n$. (2) $d_H(\mathbf{r}_i, \mathbf{r}_j) = 2^{n-1}$ *for all* $i \neq j$.

We prove the facts by induction. The case $n = 1$ is easily settled by inspection. Suppose that the claim is true for n, using $A(n+1)$ (Fig. 3.1), we shall show that it also holds for $n+1$.

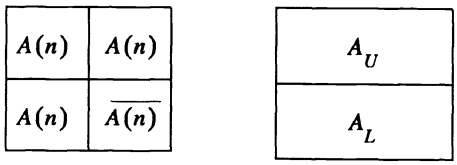


FIG. 3.1. *Inner product function table for sequence length* $n+1$.

To prove Fact 1, note that each row in $A(n+1)$ is a concatenation of two 2^n-sequences, each having weight 2^{n-1}. To prove Fact 2, we denote by A_U and A_L the upper and lower halves of $A(n+1)$ respectively, as shown in Fig. 3.1b. Consider any two sequences $\mathbf{r}_1$ and $\mathbf{r}_2$ in $A(n+1)$. If both of them are in A_U or A_L, then the Hamming distance between each half sequence is 2^{n-1}. If one of them is in A_U and the other in A_L, then there are two cases.

(i) $\mathbf{r}_i = \mathbf{aa}$, i.e. the concatenation of two copies of $\mathbf{a}$, which is a row in $A(n)$; and $\mathbf{r}_j = \mathbf{a}\bar{\mathbf{a}}$: The Hamming distance between the first half sequences is $d_H(\mathbf{a}, \mathbf{a}) = 0$ and that of the second half is $d_H(\mathbf{a}, \bar{\mathbf{a}}) = 2^n$.

(ii) $\mathbf{r}_i = \mathbf{aa}$, and $\mathbf{r}_j = \mathbf{b}\bar{\mathbf{b}}$, where $\mathbf{a}$ and $\mathbf{b}$ are rows in $A(n)$: The Hamming distance between the first half sequences is $d_H(\mathbf{a}, \mathbf{b}) = 2^{n-1}$ and that of the second half is $d_H(\mathbf{a}, \bar{\mathbf{b}}) = 2^{n-1}$.

In either case, the total Hamming distance is 2^n. □

For convenience of notation, denote 2^n by N. We are ready to prove that $M_\varepsilon(f_{IP}^n) \leqq (1 + c\varepsilon) \cdot N$ for $0 \leqq \varepsilon \leqq \frac{1}{8}$. First consider q-rect's giving function value 0. Suppose there exists such a q-rect $P = A \times B$ of size $(L+1) \times M$, such that $(L+1)M > (1+c\varepsilon) \cdot N$. Construct the product set $Q = A' \times B$, of size $L \times M$ from P by the following procedure: If there is a row of all zeros in P, remove it, otherwise remove an arbitrary row. Consider $R = A' \times \{0, 1\}^n$ (i.e. the rows of the function table of which Q is a part). We define α_i, $i = 1, \cdots, N$ as the proportion of ones in the ith column of R. From Fact 1,

$$L \cdot \sum_{i=1}^{N} \alpha_i = LN/2. \tag{3.1}$$

From the restriction on the amount of impurities in Q (which is the first M columns of R),

$$0 \leqq L \cdot \sum_{i=1}^{M} \alpha_i \leqq \varepsilon ML. \tag{3.2}$$

Combining with (3.1),

$$N/2 \geqq \sum_{i=M+1}^{N} \alpha_i \geqq N/2 - \varepsilon M. \tag{3.3}$$

Next, we compute the Hamming distance of all combinations of two rows of R, first counting vertically and then horizontally. From Fact 2,

$$L^2 \cdot \sum_{i=1}^{N} \alpha_i \bar{\alpha}_i = \frac{L(L-1)}{2} \cdot \frac{N}{2} \quad \text{where } \bar{\alpha}_i = 1 - \alpha_i.$$

Separating the sum into two parts,

$$L^2 \cdot \sum_{i=M+1}^{N} \alpha_i \bar{\alpha}_i = \frac{L(L-1)}{2} \cdot \frac{N}{2} - L^2 \sum_{i=1}^{M} \alpha_i \bar{\alpha}_i$$

$$\geqq \frac{L(L-1)}{2} \cdot \frac{N}{2} - L^2 \sum_{i=1}^{M} \alpha_i.$$

Substituting the right-hand inequality of (3.2) and dividing both sides by $N-M$,

$$\frac{1}{N-M} \sum_{i=M+1}^{N} \alpha_i \bar{\alpha}_i \geqq \frac{1}{N-M}\left[\frac{(L-1)}{2L} \cdot \frac{N}{2} - \varepsilon M\right]. \tag{3.4}$$

Substituting the left-hand inequality of (3.3),

$$\frac{1}{N-M} \sum_{i=M+1}^{N} \alpha_i^2 \leqq \frac{(L+1)}{4L(N-M)} \cdot \left[N + \frac{4\varepsilon LM}{L+1}\right]. \tag{3.5}$$

It is easy to see that

$$\frac{1}{N-M} \sum_{i=M+1}^{N} \alpha_i^2 \leqq \left(\frac{1}{N-M} \sum_{i=M+i}^{N} \alpha_i\right)^2.$$

Applying this to the left-hand inequality of (3.3),

$$\frac{1}{N-M} \sum_{i=M+1}^{N} \alpha_i^2 \geqq \frac{N^2}{4(N-M)^2}\left[1 - \frac{2\varepsilon M}{N}\right]^2. \tag{3.6}$$

Combining (3.5) and (3.6), we have

$$\left(1+\frac{1}{L}\right)\left(1-\frac{M}{N}\right) \geqq \frac{\left[1-\frac{2\varepsilon M}{N}\right]^2}{\left[1+\frac{4(L-1)\varepsilon M}{(L+1)N}\right]}. \tag{3.7}$$

The assertion we have made is that $(L+1)M > (1+c\varepsilon)N$. Substituting this into (3.7) and invoking the fact that $\varepsilon \leqq \frac{1}{8}$, we show in Appendix 3 that there exists a constant $c = c(\varepsilon)$ such that (3.7) is a contradiction. Hence $(L+1) \leqq (1+c\varepsilon)N$ as claimed.

To prove that the size of the largest q-rect for function value 1 also satisfies the same upper bound, note that Facts 1 and 2 still hold if we replace them by the corresponding statement after taking componentwise complements. This completes the proof of the lemma. □

Applying Lemmas 3.1 and 3.3 to this result, the main theorem follows readily.

THEOREM 3.1. *For* $0 < \varepsilon \leqq \frac{1}{16}$,

$$C_\varepsilon(d_H^n) + 2\lceil \log(n+1) \rceil \geqq C_\varepsilon(f_I^n) \geqq C_\varepsilon(f_{IP}^n) \geqq n - \log(1+c\varepsilon) - 1,$$

where c is a constant dependent only on ε.

Similarly, for the average case complexity

THEOREM 3.2. *For* $0 < \varepsilon \leqq \frac{1}{16}$,

$$\bar{C}_\varepsilon(d_h^n) + 2\lceil \log(n+1) \rceil \geqq \bar{C}_\varepsilon(f_I^n) \geqq \bar{C}_\varepsilon(f_{IP}^n) \geqq (n - \log(1+c\varepsilon) - 1)/2,$$

where c is a constant dependent only on ε.

4. The ε-randomized model. The ε-randomized protocol was introduced by Yao and a definition can be found in [Yao2]. The ε-randomized communication complexity of computing the Hamming distance, $D_\varepsilon(d_H^n)$, was investigated in [Yao3], where it was proved that $D_\varepsilon(d_H^n)$ grows faster than $\Omega(\log n)$. In the following Theorem, we use the results derived in § 3 to show that $D_\varepsilon(d_H^n) = \Omega(n)$, thus resolving an open problem posed by Yao in [Yao3]. The proof uses the following.

LEMMA 4.1 [Yao1]. *For any function f and* $0 \leqq \varepsilon < \frac{1}{2}$,

$$D_\varepsilon(f) \geqq (\bar{C}_{2\varepsilon}(f))/2.$$

THEOREM 4.1. *For* $0 \leqq \varepsilon < \frac{1}{2}$,

$$D_\varepsilon(d_H^n) = \Omega(n).$$

Proof. For $0 \leqq \varepsilon < \frac{1}{32}$, the theorem follows readily from Theorem 3.1 and Lemma 4.1. For $\frac{1}{32} \leqq \varepsilon < \frac{1}{2}$, given a randomized protocol with complexity $D_\varepsilon(f)$ and error probability $\varepsilon = \frac{1}{2} - \delta$, we can construct one with error probability less than $\frac{1}{64}$ as follows: Given the pair of values $(\mathbf{x}, \mathbf{y})$, repeat the protocol $2m-1$ times, such that $m(1-4\delta^2)^m < \frac{1}{64}$, and take the majority of the outcome as $f(\mathbf{x}, \mathbf{y})$. It is easy to show that the resulting error probability is no more than

$$\sum_{k \leqq m} \binom{2m-1}{k} (1-\varepsilon)^{2m-1-k} \leqq \frac{1}{64}.$$

Clearly, m is a function of ε only. Hence, there exists a constant $c = c(\varepsilon)$ such that

$$c \cdot D_\varepsilon(d_H^n) \geqq D_{1/64}(d_H^n),$$

and the theorem follows from the lower bound on the left-hand expression. □

5. Concluding comments. The upper and lower bounds for $C(d_H^n)$ can be compared by examining $\lceil \log(n+1-\sqrt{n}) \rceil$ and $\lceil \log(n+1) \rceil$. One finds that for all n, the two terms never differ by more than 1. (Actually, except for those n which satisfy $n+1 > 2^m$ and $n+1-\sqrt{n} \leqq 2^m$ for some integer m, they are identical.) Hence, our bounds are tight to within one bit. This difference is probably due to a combination of the facts that we are only considering m-rect's of maximal size for each δ, and that the optimal $m(f)$-partition is simply not achievable. It does not seem likely that there exists an algorithm whose complexity is lower than the obvious upper bound.

In Corollary 2.1, we showed that $M(n, \delta) = \binom{n}{\delta}$ for $\delta < \lceil n/2 - \sqrt{n/4} \rceil$ and $\delta > \lfloor n/2 + \sqrt{n/4} \rfloor$. We also showed in Corollary 2.3 that $M(n, \lfloor n/2 \rfloor) = M(n, \lceil n/2 \rceil) = 2^n$. However, it is not known whether the $M(n, \delta)$ upper bounds for $n/2 - \sqrt{n/4} \leqq \delta \leqq n/2 + \sqrt{n/4}$ are achievable. We believe that they are not. The interesting question then is whether one can prove tighter upper bounds for them.

Appendix 1. We prove in this Appendix the lower bound on

$$S_2 = \sum_{\delta = \lceil n/2 - \sqrt{n/4} \rceil}^{\lceil n/2 + \sqrt{n/4} \rceil} \frac{N(n, \delta)}{M(n, \delta)}$$

defined in the proof of Theorem 2.3. There are two cases.

Case 1. $n=2m$ for some m: By Corollary 3, for each δ in the range of S_2,

$$\frac{N(2m,\delta)}{M(2m,\delta)} \geqq \prod_{i=0}^{\delta-1} \frac{2m-2i-1}{2m-2i}.$$

For $\delta = m$, we have

$$\frac{N(2m,m)}{M(2m,m)} \geqq \binom{2m}{m} \geqq \frac{2^{2m}}{\sqrt{4m}}.$$

For any values of δ in the range, we have

$$\frac{N(2m,\delta)}{M(2m,\delta)} \geqq \frac{2^{2m}}{\sqrt{4m}} \cdot \prod_{i=1}^{m-\delta} \frac{2i}{2i-1}.$$

Note that each of these terms $\geqq 2(2^{2m}/\sqrt{4m})$ and there are $\lceil\sqrt{2m}\rceil$ of them. Hence

$$S_2 \geqq (2 \cdot \lceil\sqrt{2m}\rceil + 1) \cdot \frac{2^{2m}}{\sqrt{4m}}$$

$$\geqq 2^{2m}.$$

Case 2. $n=2m-1$ for some m: There are two middle terms.

$$\frac{N(2m-1,m)}{M(2m-1,m)} = \frac{N(2m-1,m-1)}{M(2m-1,m-1)} \geqq \binom{2m-1}{m} \geqq \frac{2^{2m-1}}{\sqrt{4m}}$$

and for any value of δ in the range, we have

$$\frac{N(2m-1,\delta)}{M(2m-1,\delta)} \geqq \frac{2^{2m-1}}{\sqrt{4m}} \cdot \prod_{i=1}^{m-\delta} \frac{2i+1}{2i}.$$

Note that each of these terms is $\leqq \frac{3}{2} \cdot 2^{2m-1}/\sqrt{4m}$ and there are $\lceil\sqrt{2m-1}\rceil - 2$ of them. Therefore

$$S_2 \geqq \left(\frac{3}{2} \cdot (\lceil\sqrt{2m-1}\rceil - 2) + 2\right) \cdot \frac{2^{2m-1}}{\sqrt{4m}} \geqq 2^{2m-1}.$$

Hence in both cases, the assertion $S_2 \geqq 2^n$ is true. □

Appendix 2. In this Appendix, we give the proofs for Corollaries 2.1, 2.2 and 2.3.

COROLLARY 2.1. *For* $\delta < \lceil n/2 - \sqrt{n/4}\rceil$ *or* $\delta > \lfloor n/2 + \sqrt{n/4}\rfloor$, $M(n,\delta) = \binom{n}{\delta}$.

Proof. By Lemma 2.2 and the fact that $\binom{n}{\delta} = \binom{n}{n-\delta}$, we only have to consider the range $\delta < \lceil n/2 - \sqrt{n/4}\rceil$. For any δ, define $A \triangleq \{0\}$ and $B \triangleq \{\mathbf{x}: d(\mathbf{x},0) = \delta\}$. It is clear that $A \times B \in S_\delta^n$ and that $\|A \times B\| = \binom{n}{\delta}$. Therefore one side of the equality is proved. To prove the other side, just note that the equation

$$4 = \frac{n(n-1)}{x(n-x)}$$

has positive root $x = n/2 - \sqrt{n/4}$. Hence, for $\delta < \lceil n/2 - \sqrt{n/4}\rceil$,

$$\max\left(4, \frac{n(n-1)}{\delta(n-\delta)}\right) = \frac{n(n-1)}{\delta(n-\delta)}.$$

Moreover, this still holds if we replace n by $n-2j$ and δ by $\delta-j$, for all $j<\delta$. Subsequently, by Lemma 2.2,

$$M(n,\delta)\leqq\prod_{j=0}^{\delta-1}\frac{(n-2j)((n-2j)-1)}{(\delta-j)((n-2j)-(\delta-j))}\cdot M((n-2\delta),0)=\binom{n}{\delta}. \qquad \square$$

COROLLARY 2.2. *For* $n/2-\sqrt{n/4}\leqq\delta\leqq n/2+\sqrt{n/4}$,

$$M(n,\delta)\leqq\prod_{j=0}^{\delta'-1}\frac{(n-2j)^2}{(\delta'-j)(n-\delta'-j)},$$

where

$$\delta'=\begin{cases}\delta & \text{for } \delta\leqq\lfloor n/2\rfloor,\\ \lfloor n/2\rfloor-\delta & \text{otherwise.}\end{cases}$$

Proof. First note that for $n\geqq 4$, the following holds for all $0\leqq\delta\leqq\lfloor n/2\rfloor$:

$$\frac{n^2}{\delta(n-\delta)}\geqq\max\left(4,\frac{n(n-1)}{\delta(n-\delta)}\right)$$

and the relation is definitely true for the range of δ in this corollary. Hence $M(n,\delta)/M(n-2,\delta-1)\geqq n^2/\delta(n-\delta)$ and it is clear that this still holds true when we replace n by $n-2j$ and δ by $\delta-j$, for $j\leqq\delta$. Apply Theorem 2.4 recursively δ times and since $M(n-2\delta,0)=1$

$$M(n,\delta)\leqq\prod_{j=0}^{\delta-1}\frac{(n-2j)^2}{(\delta-j)((n-2j)-(\delta-j))}=\prod_{j=0}^{\delta-1}\frac{(n-2j)^2}{(\delta-j)(n-\delta-j)},$$

which completes the proof of the corollary. $\square$

COROLLARY 2.3. *For* $n=1,2,\cdots,\max_{0\leqq\delta\leqq n}M(n,\delta)=2^n$ *and the maximum is achieved by* $\delta=\lfloor n/2\rfloor$ *and* $\lceil n/2\rceil$.

Proof. We first show that $M(n,\lfloor n/2\rfloor)=2^n$. By Lemma 2.2, this also establishes $M(n,\lceil n/2\rceil)=2^n$. The crucial observation is that for $\delta=\lfloor n/2\rfloor$, $n(n-1)/\delta(n-\delta)\leqq 4$. Hence $M(n,\lfloor n/2\rfloor)/M(n-2,\lfloor n/2\rfloor-1)\leqq 4$. Moreover, this relation is still true if we replace n by $n-2j$ and δ by $\delta-j$, for $j\leqq\delta$. For even n, apply Theorem 2.4 recursively $n/2-1$ times and since $M(2,1)=2$, we obtain $M(n,n/2)\leqq 2^n$. For odd n, apply Theorem 2.4 recursively $(n+1)/2$ times and since $M(1,0)=1$, we obtain $M(n,\lfloor n/2\rfloor)\leqq 2^n$. On the other hand, for even n, define $A\triangleq\{01,10\}^{n/2}$ and $B\triangleq\{00,11\}^{n/2}$. Clearly $A\times B\in S^n_{n/2}$ and $\|A\times B\|=2^n$. For odd n, define $C\triangleq A\times\{0\}$ and $C\triangleq B\times\{0\}$ and $C\times D\in S^n_{\lfloor n/2\rfloor}$. Therefore $M(n,\lfloor n/2\rfloor)\geqq 2^n$.

Now, suppose there exist $A\times B\in S^n_\delta$ such that $\|A\times B\|>2^n$. Consider the following two cases:

a) $n=2m$ for some m. By Lemma 2.2, $\bar{A}\times B\in S^n_{n-\delta}$. Define $C\triangleq A\times\bar{A}$ and $D\triangleq B\times B$. Clearly, $C\times D\in S^{4m}_{2\delta}$ and $\|C\times D\|>2^{4m}$ which is a contradiction to Corollary 2.3.

b) $n=2m+1$ for some m. Again $C\times D\in S^{4m}_{2\delta}$. However, since n is odd, $A\cap\bar{A}=\varnothing$ and $B\cap\bar{B}=\varnothing$ (cf. Lemma 2.2). Hence $C\cap\bar{C}=\varnothing$ and $D\cap\bar{D}=\varnothing$. Define $P\triangleq C\cup\bar{C}$ and $Q\triangleq D\cup\bar{D}$. Therefore $P\times Q\in S^{4m+2}_{2\delta}$ and $\|P\times Q\|>2^{4m+2}$ which is a contradiction to Corollary 2.3. $\square$

Appendix 3. We prove in this Appendix that

$$\left(1+\frac{1}{L}\right)\left(1-\frac{M}{N}\right)\geqq\frac{(1-2\varepsilon M/N)^2}{1+4(L-1)\varepsilon M/((L+1)M)} \tag{A3.1}$$

is a contradiction if $(L+1)M=\Lambda>(1+c\varepsilon)N$ for a constant $c=c(\varepsilon)$ and $\varepsilon>\frac{1}{8}$. The

left-hand side of (A3.1) is

$$\left(1+\frac{1}{L}\right)\left(1-\frac{M}{N}\right)=\left(1+\frac{1}{L}\right)\left(1-\frac{\Lambda}{N(L+1)}\right)$$
$$=1+\frac{1}{L}-\frac{\Lambda}{N(L+1)}-\frac{\Lambda}{NL(L+1)}.$$

The right-hand side of (A3.1) is larger than

$$\left(1-\frac{4\varepsilon}{N}\right)\cdot\left(1-\frac{4(L-1)\varepsilon M}{(L+1)N}\right)\geqq 1-\frac{8\varepsilon M}{N}\left(1+\frac{L-1}{L+1}\right)\geqq 1-\frac{8\varepsilon M}{N}.$$

For (A3.1) to be a contradiction, we want

$$1-\frac{8\varepsilon M}{N}>1+\frac{1}{L}-\frac{\Lambda}{N(L+1)}-\frac{\Lambda}{NL(L+1)},$$

which simplifies to

$$\frac{L\Lambda}{L+1}\left(1+\frac{1}{L}-8\varepsilon\right)>N. \tag{A3.2}$$

Since

$$1-8\varepsilon<\frac{L}{L+1}\left(1+\frac{1}{L}-8\varepsilon\right),$$

(A3.2) is true if

$$\Lambda(1-8\varepsilon)>N$$

which is equivalent to $\Lambda>(1+c\varepsilon)N$ for a constant $c=c(\varepsilon)$. Note that the condition $\varepsilon<\frac{1}{8}$ is certainly required for the above to hold. □

Acknowledgment. We are grateful to Alon Orlitsky for his contribution in the formalization of deterministic protocols.

REFERENCES

[AEP] R. AHLSWEDE, A. EL GAMAL AND K. F. PANG, *A two family extremal problem in Hamming space*, Discrete Math., 49 (1984), pp. 1-5.

[MS] K. MEHLHORN AND E. M. SCHMIDT, *Las Vegas is better then determinism in* VLSI *and distributed computing*, Proc. 14th Annual ACM Symposium on Theory of Computing, April 1982, pp. 330-337.

[Yao1] A. C. YAO, *Probabilistic computations: towards a unified measures of complexity*, Proc. 18th Symposium on Foundations of Computer Science, Oct. 1977, pp. 222-227.

[Yao2] ———, *Some complexity questions related to distributive computing*, Proc. 11th Annual ACM Symposium on Theory of Computing, May 1979, pp. 209-213.

[Yao3] ———, *Lower bounds by probabilistic arguments*, Proc. 25th Symposium on Foundations of Computer Science, November 1983, pp. 420-428.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986

© 1986 Society for Industrial and Applied Mathematics
004

IMPROVED BOUNDS FOR MATROID PARTITION AND INTERSECTION ALGORITHMS*

WILLIAM H. CUNNINGHAM†

Abstract. We give bounds on total lengths of augmenting paths in standard implementations of the matroid partition and intersection algorithms, and indicate how these observations can be used to improve the running times in certain applications. For example, for the matroid intersection algorithm on two r by n matrices the running time is shown to be $O(nr^2 \log r)$. We also give improved versions of the two algorithms, when running times are measured in terms of calls to an independence oracle. For example, there is a matroid partition algorithm on $O(n)$ n-element matroids using $O(n^{2.5})$ independence tests.

Key words. matroid partition, matroid intersection, shortest augmenting path, layered network

AMS(MOS) subject classifications. 68C25, 05B35

1. Introduction. Let E be a finite set and let $M = (E, \mathcal{I})$ be a matroid on E. Here $\mathcal{I}$ denotes the family of subsets of E which are *independent* in M. We assume familiarity with a few basics of matroid theory; see Welsh [17] for a reference. However, this paper can be read profitably by keeping in mind the class of matroids called *linear*: E indexes the columns of a matrix over some field, and independence means linear independence of columns. Another standard class consists of the *graphic* matroids: E is the edge-set of an undirected graph and independent means forest-forming.

The *matroid partition* problem is: Given matroids $M_i = (E, \mathcal{I}_i)$, $1 \leqq i \leqq k$, find a maximum cardinality set $J \subseteq E$ such that $J = \bigcup J_i$ where $J_i \in \mathcal{I}_i$, $1 \leqq i \leqq k$. Clearly, the J_i can be required to be disjoint; we call such a set J *partitionable.*

The *matroid intersection* problem is: Given matroids $M_i = (E, \mathcal{I}_i)$, $i = 1$ and 2, find a maximum cardinality set $J \in \mathcal{I}_1 \cap \mathcal{I}_2$. Both of these problems can be solved by polynomial-time algorithms, assuming the existence of similar algorithms for (say) recognizing independent sets in the relevant matroids [3], [6], [5], [11]. The two problems are also polynomially transformable to each other [4], and we shall make use of the resulting correspondence.

These algorithms are augmenting-path methods which generalize the classical bipartite matching algorithm. Like all such methods, they can be described in terms of successively finding source-sink directed paths in appropriately-defined auxiliary digraphs. A natural way to find such paths is via breadth-first search, so that the resulting paths will be as short as possible. For bipartite matching Hopcroft and Karp [10] devised an improved algorithm, using the fact that there exist extremely short augmenting paths. We establish similar results for matroid partition and intersection, and use them in two ways.

First, we point out that, for the important class of linear matroids, the running time depends directly on the total length of augmenting paths used, so that usual versions of these algorithms actually have better running times than previously realized. For matroid intersection on two r by n matrices, we obtain a running time of $O(nr^2 \log r)$. For matroid partition on $O(n)$ matrices, each having n columns and at most n rows, we obtain a running time of $O(n^3 \log n)$.

* Received by the editors September 25, 1984, and in revised form August 6, 1985. This research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

† Department of Mathematics and Statistics, Carleton University, Ottawa, Ontario, Canada, K1S 5B6.

Second, we improve both algorithms relative to the abstract complexity measure of the total number of tests of independence required. This improvement is based on the Dinic-Hopcroft-Karp "layered network" approach, showing that there are relatively few different lengths of shortest augmenting paths, and that all the augmentations of a fixed length can be found and performed quickly. The number of independence tests for the matroid intersection problem on two n-element matroids having maximum rank r is improved from $O(r^2n)$ to $O(r^{1.5}n)$. The number of such tests for the matroid partition problem for $O(n)$ n-element matroids is improved from $O(n^3)$ to $O(n^{2.5})$.

2. Matroid intersection. The matroid intersection algorithm can be initialized with $J=\varnothing$. At a general step of the algorithm we define an auxiliary digraph G relative to J as follows. The vertex-set of G is $E\cup\{r, s\}$, where $r, s\notin E$. There is an edge (r, e) for every $e\in E\backslash J$ for which $J+e\in\mathscr{I}_1$. There is an edge (f, e) for every $f\in J$, $e\in E\backslash J$, such that $J+e\notin\mathscr{I}_1$ and $J+e-f\in\mathscr{I}_1$. (In this paper, we use $J+e$, $J+e-f$ to abbreviate the more cumbersome expressions $J\cup\{e\}$, $(J\cup\{e\})\backslash\{f\}$.) There is an edge (e, s) for every $e\in E\backslash J$ such that $J+e\in\mathscr{I}_2$. There is an edge (e, f) for every $e\in E\backslash J$, $f\in J$ such that $J+e\notin\mathscr{I}_2$ and $J+e-f\in\mathscr{I}_2$.

If there is no directed path from r to s in G, then J is a maximum cardinality common independent set, and the algorithm terminates. If there is such a path, we choose a shortest one, having vertex-sequence $r, a_1, b_1, \cdots, a_{m-1}, b_{m-1}, a_m, s$. Notice that each $a_j\in E\backslash J$ and each $b_j\in J$. We update J by deleting $b_1, \cdots, b_{m-1}$ from J and adding $a_1, \cdots, a_m$ to J. The validity of the algorithm is not difficult to establish; see [12], [14], for example. The paths need not, in fact, be shortest. It is enough that they be *chordless*, that is, no proper subsequence yields an (r, s)-dipath. However, that they be shortest is crucial in the improvements we shall make.

Consider the implementation of the algorithm. Let r denote the maximum size of an independent set of M_1 or M_2, and let $n=|E|$. To construct the auxiliary digraph relative to $J\in\mathscr{I}_1\cap\mathscr{I}_2$ will involve determining, for $i=1$ and 2, for each $e\in E\backslash J$, whether $J+e\in\mathscr{I}_i$ and, if not, for each $f\in J$ whether $J+e-f\in\mathscr{I}_i$. Thus it requires $O(rn)$ tests of independence in each of M_1 and M_2. (Finding an (r, s)-dipath can be done in $O(rn)$ time, so this work is dominated by the effort of building G.) At most r augmentations will be performed, so the total effort is $O(r^2nQ)$, where Q is the maximum running time of the independence-testing routines in M_1 and M_2. We have just completed the usual textbook analysis of the matroid intersection algorithm; see for example [12], [14].

When we apply this result to the concrete special case in which M_1 and M_2 are linear, we see that the introduction of Q was a mistake. In that case we may assume that each of M_1, M_2 is given by a matrix having n columns and at most r rows. Then Q is $O(r^3)$, by some form of Gaussian elimination, assuming that arithmetic operations in the relevant field are treated as single steps. (The reader is no doubt aware that $O(r^3)$ could be replaced by $O(r^{2.496})$, or some such expression, due to sophisticated matrix multiplication methods. Making this replacement would improve some of the bounds in the next paragraphs. However, no algorithm using these methods seems to be competitive, even in theory, with the one we shall ultimately derive, so we shall ignore them.) The resulting estimate, $O(r^5n)$, grossly exceeds that for a sensible implementation. The reason, simply, is that the tests of independence required in the algorithm are closely related to each other. The tests of independence required to construct G can *all* be done in time $O(r^2n)$, rather than $O(r^4n)$, as follows. Once the columns of the matrix for M_i corresponding to J are transformed to elementary vectors by Gauss-Jordan elimination, all of the required information is at hand. Namely, for $e\notin J$, $J+e\in\mathscr{I}_i$ if and only if column e has a nonzero in a row in which all columns

from J are zero, and if $J+e \notin \mathcal{I}_i$, then $J+e-f \in \mathcal{I}_i$ if and only if column e has a nonzero in the row in which f does. This observation reduces the time bound to $O(r^3 n)$.

The analysis can be carried further. Given the matrices for M_1 and M_2 in the above "J-reduced form", how much work is required to update the matrices after an augmentation? Let us assume that the path which determines the augmentation has vertex-sequence $r, a_1, \cdots, b_{m-1}, a_m, s$. Then for each j, $2 \leqq j \leqq m$, we must pivot in the matrix for M_1 on the entry in column a_j and in the row which has a one in column b_{j-1}. In addition we must pivot on the entry in column a_1 and a row in which no column of J has a nonzero. (A "pivot" is the usual collection of column-clearing row operations.) Similarly, we must perform m pivots in the matrix for M_2. The amount of work in a pivot operation is $O(rn)$. Thus the time bound for constructing the new auxiliary digraph is $O(rnm)$. The only apparent bound for m is r, which leads to the same overall bound for the algorithm as before, $O(r^3 n)$. However, since at each step we find a shortest augmenting path, this analysis suggests improving the overall bound by finding better upper bounds on the lengths of augmenting paths.

A set of directed (r, s)-dipaths is *vertex-disjoint* if their vertex-sets are pairwise disjoint, except for r and s. The next two results generalize similar ones of Hopcroft and Karp [10]. They have been discovered independently by Gabow and Stallman [9].

THEOREM 2.1. *Let J be a common independent set of M_1, M_2 and let J' be a larger one. The auxiliary digraph relative to J contains $|J'|-|J|$ vertex-disjoint (r, s)-dipaths, all of whose internal vertices are contained in $J' \cup J$.*

Proof. The maximum number of such paths, by the well-known theorem of Menger, is the minimum cardinality of a set $T \subseteq J \cup J'$ such that every such path contains at least one vertex from T. Choose such a set T. Then there is a partition $\{R, T, S\}$ of $J \cup J'$ such that no edge of G goes from $R+r$ to $S+s$. Since there is no edge (r, e) for $e \in R \setminus J$, therefore $J+e \notin \mathcal{I}_1$ for all $e \in R \setminus J$. Since there is no edge (f, e) for $f \in S \cap J$ and $e \in R \setminus J$, therefore for every $e \in R \setminus J$, $\{f \in J: J+e-f \in \mathcal{I}_1\} \subseteq R \cup T$. It follows that $J \cap (R \cup T)$ is a maximal M_1-independent subset of $R \cup (J \cap T)$. By symmetry, $J \cap (S \cup T)$ is a maximal M_2-independent subset of $S \cup (J \cap T)$. Therefore,

$$|J|+|J \cap T| \geqq |J' \cap (R \cup (J \cap T))|+|J' \cap (S \cup (J \cap T))|.$$

It follows that $|J|+|T| \geqq |J'|$, as required. □

COROLLARY 2.2. *Let p be the maximum size of a common independent set and let J be a common independent set which is not maximum. Then there exists in the auxiliary digraph for J an augmenting path having length at most $2|J|/(p-|J|)+2$.*

Proof. There exist $p-|J|$ vertex-disjoint augmenting paths, so there is one having at most $|J|/(p-|J|)$ vertices from J. Its length will be at most $2|J|/(p-|J|)+2$. □

The next result, in the special case of bipartite matching, is due to Even and Tarjan [7], who also extended it to some other special network flow problems.

THEOREM 2.3. *The sums of lengths of the augmenting paths used in the matroid intersection algorithm is $O(p \log p)$, where p is the maximum size of a common independent set.*

Proof. After j augmentations, there exists a path of length at most $2j/(p-j)+2$, by (2.2). Therefore, the total length of all augmenting paths is at most $2 \sum (j/(p-j): 0 \leqq j \leqq p-1)+2p$. But

$$\sum (j/(p-j): 0 \leqq j \leqq p-1) = \sum ((p-j)/j: 1 \leqq j \leqq p)$$

$$\leqq p \sum (1/j: 1 \leqq j \leqq p) = O(p \log p),$$

as required. □

Let us apply the above result to the matrix case. For simplicity we use the inequality $p \leqq r$. The total number of pivots required over the whole application of the algorithm is $O(r \log r)$, and so the total contribution of pivoting to running time is $O(nr^2 \log r)$. Since this work dominates all other work (in particular, the total work in finding augmentations is $O(r^2 n)$), the overall running time is $O(nr^2 \log r)$.

We were able to use Theorem 2.3 to improve the time bound in the linear case because constructing the auxiliary digraphs is relatively expensive compared to finding augmentations. It is interesting to consider the implementation of the matroid intersection algorithm for two graphic matroids, where this is not the case. We may assume that M_1, M_2 are given by graphs $G_1 = (V_1, E)$ and $G_2 = (V_2, E)$. Here $|V_1|$, $|V_2|$ are $O(r)$. There are standard techniques for maintaining a forest J of a graph so that for each $e \notin J$ one can determine in $O(r)$ time whether $J + e$ is a forest, and, if not, compute the set $\{f \in J: J + e - f \text{ is a forest}\}$. Moreover, the representation can be updated in $O(r)$ time when J is replaced by either $J + e$ or $J + e - f$. Hence the total amount of time required to find augmentations in an application of the matroid intersection algorithm is $O(r^2 n)$. The total amount of time required to update the representation, without appealing to Theorem 2.3, is $O(r^3)$. Using that result lowers this bound to $O(r^2 \log r)$, but clearly that does not improve the order of the overall bound, which remains $O(r^2 n)$. (We remark that there exist more sophisticated implementations of the intersection algorithm for graphic matroids, which do yield better bounds; see [9].)

3. Matroid partition. The matroid partition algorithm can be initialized with $J = J_i = \varnothing$, $1 \leqq i \leqq k$. At a general step of the algorithm, we define an auxiliary digraph G relative to the J_i as follows. The vertex-set of G is $E \cup \{r, s\}$, where $r, s \notin E$. There is an edge (r, e) for every $e \in E$ such that $e \notin J = \cup J_i$. There is an edge (e, s) for every $e \in E$ for which there exists i, $1 \leqq i \leqq k$, with $e \notin J_i$ and $J_i + e \in \mathcal{I}_i$. There is an edge (e, f) for every pair of elements $e, f \in E$ such that there exists i, $1 \leqq i \leqq k$, with $J_i + e \notin \mathcal{I}_i$ and $J_i + e - f \in \mathcal{I}_i$.

If there is no directed path from r to s in G, then J is a maximum cardinality partitionable set, and the algorithm terminates. If there is such a path, we choose a shortest one, having vertex-sequence $r = e_0, e_1, e_2, \cdots, e_m, e_{m+1} = s$. For $1 \leqq j \leqq m$, let $i(j)$, $1 \leqq i(j) \leqq k$, be an index giving rise to the edge (e_j, e_{j+1}) of G. For each i, we update J_i by adding e_j and deleting e_{j+1} for every j such that $i = i(j)$. ($|J|$ will increase by one since, for some i, e_m will be added to J_i, with no corresponding deletion. Notice that elements are not deleted from J; they are only moved among the J_i.) The validity of the algorithm is not difficult to establish directly; see [17] for example. It also follows from the validity of the intersection algorithm, as we shall see.

Consider the implementation of the algorithm. Let r be the maximum size of an independent set in any of the M_i, let $n = |E|$, and let us make the simplifying assumption that $k \leqq n$. Constructing the auxiliary digraph relative to $J = \cup J_i$ will require determining, for $1 \leqq i \leqq k$, for each $e \in E \setminus J_i$, whether $J_i + e \in \mathcal{I}_i$ and, if not, whether $J_i + e - f \in \mathcal{I}_i$ for each $f \in J_i$. Since $|J| \leqq n$, this requires $O(n^2)$ tests of independence. (Finding an augmenting path requires $O(n^2)$ time.) Hence we obtain an overall time bound of $O(n^3 Q)$, where Q is the maximum running time of the independence-testing routines for the M_i.

If each of the M_i arises from a matrix having n columns and at most r rows, then Q is $O(r^3)$ and we get a running time of $O(r^3 n^3)$. However, this is easily improved. When the matrix for M_i is J_i-reduced, the auxiliary digraph is easily constructed. Since $\sum |J_i| \leqq n$, we need at most n pivots, each requiring $O(rn)$ time, so each main step can

be done in time $O(rn^2)$, giving an overall bound of $O(rn^3)$ for the algorithm. Again, pivoting is the bottleneck in this bound—the total time required to find augmentations is $O(n^3)$. The number of pivots needed to update the J_i-reduced matrices after an augmentation is related to the length of the path in G which determines the augmentation. If the path has vertex-sequence $r = e_0, e_1, \cdots, e_m, e_{m+1} = s$, then for each j, $1 \leqq j < m$, we must pivot, in the matrix for $M_{i(j)}$, on the entry in column e_j and the row which has a one in column e_{j+1}. We also must pivot in the matrix for $M_{i(m)}$ on the entry in column e_m and a row which has only zeros in the columns indexed by $J_{i(m)}$. So the total number of pivots in the whole algorithm is bounded by the sum of the lengths of all augmenting paths that are used. Therefore we want to find a better bound for this quantity.

It is possible to obtain our results on matroid partition by methods similar to those used for matroid intersection in §§ 2 and 4. In fact, an earlier version of this paper proceeded in just this way. However, as was pointed out by a referee, the well-known equivalence of the two problems can be exploited to obtain the desired results. Recall how the matroid partition problem can be solved by application of the matroid intersection algorithm. (In fact, as was proved in [4], each problem is reducible to the other.) We make k disjoint copies $E_1, E_2, \cdots, E_k$ of E, and imagine M_i, $1 \leqq i \leqq k$, as being defined on E_i rather than E. Then the direct sum [17] of the M_i is a matroid N_2 on $\cup E_i$. A set is independent in N_2 if and only if its intersection with E_i is independent in M_i, $1 \leqq i \leqq k$. We define another matroid N_1 on $\cup E_i$; a set is independent in N_1 if and only if it contains at most one copy of e for each $e \in E$. It is easy to see that common independent sets of N_1, N_2 correspond to partitionable subsets of E, with respect to $M_1, \cdots, M_k$, and this correspondence preserves cardinality. In general, applying results on the running time of the matroid intersection algorithm to N_1 and N_2 will yield bounds which depend too heavily on k, although these tend to be satisfactory for k fixed. However, there is a direct correspondence between paths in the auxiliary digraph for the partition algorithm applied to $M_1, \cdots, M_k$ and those in the auxiliary digraph for the intersection algorithm for N_1 and N_2. A dipath $r, e_1, e_2, \cdots, e_m, s$ in the former corresponds to a dipath $r, e_1'', e_2', \cdots, e_m', e_m'', s$ in the latter; here, for $2 \leqq j \leqq m$, e_j' is copy i of e_j and e_{j-1}'' is copy i of e_{j-1}, where $e_j \in J_i$. Thus the following result is an immediate consequence of Theorem 2.3.

THEOREM 3.1. *The sum of the lengths of paths on which augmentations are performed in the matroid partition algorithm is $O(p \log p)$, where p is the maximum cardinality of a partitionable set.*

Applying Theorem 3.1 to the linear case, we have that the total time required for pivoting is $O(rnp \log p)$. The time required to find an augmentation would seem to be $O(krn)$. (We make no special assumptions about sparsity of the matrices or about their storage.) However, identifying edges of the auxiliary digraph of the forms (r, e) and (e, f) requires only $O(pn)$ time per augmentation; it is edges of the form (e, s) which may require $O(rkn)$. Suppose that we remember, for each i and each $e \notin J_i$, whether $J_i + e \in \mathcal{I}_i$. Then the amount of work required to find an augmentation becomes $O(pn + kn)$. This information can be constructed initially in time $O(krn)$. After an augmentation, notice that the status of pairs (e, i) can change for only one value of i—the one for which $|J_i|$ increases. So the work required to update the extra information is $O(rn)$ per augmentation. Thus the overall time-bound is $O(p^2n + pkn + rkn + rnp \log p)$. For example, if k is constant, then $p = O(r)$, so the time is $O(r^2n \log r)$. If $k = O(n)$, then since $p \leqq n$, the time is $O(n^3 + rn^2 \log n)$.

4. Matroid intersection. In this section and the next we adopt the point of view, measuring the complexity of matroid algorithms in terms of the number of independence tests, which we criticized in the previous sections. In Theorem 4.1 we lower the previous best bound on the number of such tests for the matroid intersection algorithm by a factor of $\sqrt{r}$. In contrast to the result of § 2, there seems to be no concrete instance of matroid intersection for which the resulting time bound improves, or even equals, the previous best running time. Nevertheless, both the result and the techniques are interesting.

THEOREM 4.1. *There is a matroid intersection algorithm which, for two matroids on an n-element set having maximum independent set size r and maximum independence-testing complexity Q, has a running time of* $O(r^{1.5}nQ)$.

The algorithm extends several of the ideas behind the Hopcroft-Karp $O(\sqrt{|V|}\,|E|)$ bipartite matching algorithm. Some interesting difficulties are encountered, but they are not too different from the problems overcome in extending the Dinic-Edmonds-Karp maximum flow results to submodular network flows [15], [13], [8], [16]. (Knowledge of those papers is not necessary to read this one.)

The bipartite matching result is based on the following facts:

(I) The length of shortest augmenting paths never decreases as the matching size increases.

(II) The number of different lengths of shortest augmenting paths which will occur during execution of the algorithm is $O(\sqrt{|V|})$.

(III) It is possible to find and perform all the augmentations on paths of a single length in $O(|E|)$ time.

We shall establish analogues of these results for matroid intersection (and, in the next section, for matroid partition). We actually need a stronger form of (I), a result reminiscent of maximum flow theory. Given a digraph G and vertices e, f, $d(e,f)$ denote the length of a shortest dipath in G from e to f. (If none exists, $d(e,f)=\infty$.) For a digraph G', we replace d by d'.

THEOREM 4.2. *Let G be the auxiliary digraph for common independent set J in the matroid intersection algorithm, and let G' be the new auxiliary digraph after augmentation on a shortest augmenting path P in G. Then for all* $f \in E$, $d'(r,f) \geqq d(r,f)$ *and* $d'(f,s) \geqq d(f,s)$.

The difference between Theorem 4.2 and its analogue in the network flow and bipartite matching contexts is the way in which edges can appear in G' (after not being present in G). In the network flow case, an edge (e,f) can appear in G' only if (f,e) was an edge of the path on which the previous augmentation was performed. In the current more general context, there are other ways in which (e,f) can appear; they are essentially described in Theorem 4.3. This result is not new. It is essentially Lemma 3.2 of [1], and is implicit in [8], [13], [15].

THEOREM 4.3. *Let G, G' be digraphs, both having vertex-set* $E \cup \{r,s\}$, *and let P be a shortest* (r,s)*-dipath in G. Suppose that:*

(*) *If* (e,f) *is an edge of G' but not of G, then* $e, f \in E$, *and there exist vertices a, b of P with a preceding b on P such that*
$a = f$ *or* (a,f) *is an edge of G, and*
$b = e$ *or* (e,b) *is an edge of G.*

Then for every $f \in E$, $d'(r,f) \geqq d(r,f)$ *and* $d'(f,s) \geqq d(f,s)$.

Proof. Let us suppose that there exists some $f \in E \cup \{r,s\}$ such that $d'(r,f) < d(r,f)$, and choose f so that $d'(r,f)$ is as small as possible. Clearly, $d'(r,f) \geqq 1$, so we can choose the second-last vertex e of a dipath in G' of length $d'(r,f)$. By the choice

of f, $d'(r, e) \geqq d(r, e)$. Since $d'(r, e) = d'(r, f) - 1$, it must be that (e, f) is not an edge of G. Therefore, we may choose vertices a, b of P as in (*). Of course, $d(r, a) < d(r, b)$. But

$$\begin{aligned} d(r, a) &\geqq d(r, f) - 1 \\ &\geqq d'(r, f) \\ &= d'(r, e) + 1 \\ &\geqq d(r, e) + 1 \\ &\geqq d(r, b), \end{aligned}$$

a contradiction. This proves that $d'(r, f) \geqq d(r, f)$. The proof that $d'(f, s) \geqq d(f, s)$ is similar. □

To prove Theorem 4.2 from Theorem 4.3 we need only check that (*) is always satisfied in the matroid intersection algorithm. This can be done with the help of the following technical lemma (Lemma 4.4 from [1]), whose proof we omit. We remark that the proof is a reasonably straightforward induction. We also point out that (i) is the essential result for validating the intersection algorithm, and hence is well known. Moreover, (ii) is an immediate consequence of the fact that sets of the form $(J \cup \{a_1, \cdots, a_i\}) \setminus \{b_1, \cdots, b_i\}$ have the same span as J, which is also well known. So the only (relatively) new part is (iii).

LEMMA 4.4. *Let $M = (E, \mathcal{I})$ be a matroid, let $J \in \mathcal{I}$, let e, f be distinct elements of E, and let $a_1, b_1, \cdots, a_{m-1}, b_{m-1}, a_m$ be a sequence of distinct elements of E satisfying*:

(a) $a_j \notin J$, $b_j \in J$, $1 \leqq j < m$, *and* $a_m \notin J$;
(b) $J + a_j \notin \mathcal{I}$, $J + a_j - b_j \in \mathcal{I}$, $1 \leqq j < m$;
(c) $J + a_j - b_i \notin \mathcal{I}$, $0 \leqq i < j < m$;
(d) $J + a_m \in \mathcal{I}$.

Then:

(i) $J' = (J \cup \{a_1, \cdots, a_m\}) \setminus \{b_1, \cdots, b_{m-1}\} \in \mathcal{I}$;
(ii) *If* $e \notin J'$ *and* $J' + e \in \mathcal{I}$, *then* $e \notin J$ *and* $J + e \in \mathcal{I}$;
(iii) *If* $J' + e \notin \mathcal{I}$, $J' + e - f \in \mathcal{I}$, *and* $J + e - f \notin \mathcal{I}$, *then there exist* k, q, $1 \leqq k \leqq q < m$, *such that*

$$J + a_k \notin \mathcal{I},\ J + a_k - f \in \mathcal{I} \text{ and}$$

$$(b_q = e \text{ or } (J + e \notin \mathcal{I} \text{ and } J + e - b_q \in \mathcal{I}) \text{ or } (e \notin J \text{ and } J + e \in \mathcal{I})).$$

Proof of Theorem 4.2. First, we consider edges (e, f) of G' determined by M_2, so we take $M = M_2$ in Lemma 4.4. Then (ii) tells us that $f \neq s$. So suppose $e, f \in E$. Then we can use (iii) to obtain a_k, b_q. Put $a = a_k$; put $b = s$ if $e \notin J$ and $J + e \in \mathcal{I}_2$ and otherwise put $b = b_q$. Then a precedes b on P. Moreover, $a = f$ or (a, f) is an edge of G. Also $b = e$ or (e, b) is an edge of G. So (*) is satisfied. By symmetry, the same argument can be applied to edges determined by M_1, so the result follows from Theorem 4.3. □

It follows from Theorem 4.2 that the matroid intersection calculation can be divided into *stages*, during each of which the length of augmenting paths remains the same. We remark that the proof in [10] of this result for bipartite matching is different, and more elegant. Moreover, that proof can be generalized to the present context. The problem is that we need the stronger result Theorem 4.2 (that is, $d'(r, s) \geqq d(r, s)$ is not enough), which does not seem to be obtainable by the same technique.

The next result is the analogue of (II) for matroid intersection, and is proved by a similar method.

THEOREM 4.5. *Let p be the maximum size of a common independent set. Then the matroid intersection algorithm has $O(\sqrt{p})$ stages.*

Proof. We use Corollary 2.2, as in the proof of Theorem 2.3. Let J be the first common independent set for which there is no augmenting path of length $\leqq 2\lceil\sqrt{p}\,\rceil$, and let $|J| = j$. Then

$$2\lceil\sqrt{p}\,\rceil + 2 \leqq 2j/(p-j) + 2$$

so

$$j \geqq p\lceil\sqrt{p}\,\rceil/(\lceil\sqrt{p}\,\rceil + 1) \geqq p - \sqrt{p}.$$

The number of stages until we reach J is at most $\lceil\sqrt{p}\,\rceil$, and the number of stages thereafter is at most the number of augmentations thereafter, which is at most $p - j \leqq \sqrt{p}$. The total number of stages is at most $\lceil\sqrt{p}\,\rceil + \lfloor\sqrt{p}\,\rfloor = O(\sqrt{p})$. □

In order to find and perform efficiently all augmentations of a single stage, we consider, as in Dinic's maximum flow algorithm [2], a "layered network". (We remark that the ideas needed to extend Dinic's layered network method to submodular flows, which we use here, were introduced by Tardos, Tovey and Trick [16].) Given $J \in \mathcal{I}_1 \cap \mathcal{I}_2$ with associated auxiliary digraph G, let $d(r, s) = 2m$. Let L_i denote $\{e \in E \cup \{r, s\}\colon d(r, e) = i, d(e, s) = 2m - i\}$, $0 \leqq i \leqq 2m$. Of course, the elements of the L_i are precisely the vertices of shortest augmenting paths. At the beginning of a stage, we can compute the L_i (and m) by a standard technique. In the "forward phase" we use breadth-first search to find $L_i' = \{e \in E \cup \{r, s\}\colon d(r, e) = i\}$, $0 \leqq i \leqq 2m$, and to compute m. This requires $O(rnQ)$ time, since the number of possible edges in G is at most $2|J|(n - |J|) \leqq 2rn$. In the "reverse phase" we put $L_{2m} = \{s\}$ and successively obtain L_i from L_i', $i = 2m-1, 2m-2, \cdots, 1$ by deleting from L_i' those elements e for which there is no $f \in L_{i+1}$ such that (e, f) is an edge. This procedure requires $O(rn)$ time, and no tests of independence, if we have remembered the edges from L_i to L_{i+1} discovered in the forward phase.

Now we are ready to explain how to find augmentations. Unlike [10] we do not attempt to update the L_i after an augmentation. This would be too time-consuming, due to the somewhat strange way in which edges can enter and leave G. Hence, L_i denotes the same set throughout the stage. We do use the fact that by Theorem 4.2 every augmenting path during the stage has vertex-sequence $r, e_1, \cdots, e_{2m-1}, s$ where $e_i \in L_i$ for all i. We occasionally mark an element $e \in E$ *useless*, when we recognize that it can no longer be a vertex of any augmenting path of length $2m$, using one of the following two conditions:

(1) $e \in L_i$ and there is no edge of G from e to any nonuseless element of L_{i+1}.

(2) e is an internal vertex of the path on which we just performed an augmentation.

These two conditions must be justified. The first one is easy: obviously in this case, $d(e, s) > 2m - i$. The second is only a little harder. Obviously $d(r, e)$ is even if and only if $e \in J$. After an augmentation on path P, every internal vertex of P is either deleted from or added to J. So for every such element e, $d(r, e)$ changes. Since it cannot decrease, it must increase.

The method of finding augmentations is an ordinary depth-first search, and is similar to Dinic's method [2], for network flows. From $e \in L_i$ we search for $f \in L_{i+1}$ such that (e, f) is an edge of G, and f is not marked useless. For this we must do an independence test, because G may have been changed by augmentations. If such f is

found, we advance to it and make e its predecessor. If no such f is found, we mark e useless and retreat to its predecessor. (Of course, if $e = r$, then the stage is finished.) If we reach s, we have an augmenting path. We perform the augmentation, and mark all internal vertices of the path useless.

The estimate of running time is based on the following simple observation. After the L_i are constructed, the algorithm tests at most once whether (e, f) is an edge of G, where $e \in L_i$, $f \in L_{i+1}$. (This, in spite of the fact that (e, f) may enter and leave G many times during the stage!) The reason is that, once (e, f) is checked, either: (1) the next augmentation is through e; or (2) we retreat from e before the next augmentation is found. In either case, e is marked useless, and will never be visited again. Since there are $O(rn)$ such pairs (e, f), the running time per stage is $O(rnQ)$. Since $p \leqq r$ and there are $O(\sqrt{p})$ stages, the running time is $O(r^{1.5}nQ)$, completing the proof of Theorem 4.1.

5. Matroid partition. This section is parallel to the last one. The main result answers a question of Welsh [17], who asked whether $O(n^3 + n^2k)$ independence tests are optimal for the matroid partition problem.

THEOREM 5.1. *There is a matroid partition algorithm which, for k matroids on an n-element set having maximum independence-testing complexity Q and maximum partitionable-set size p, has a running time of $O((p^{1.5} + k)nQ + \sqrt{p}\, kn)$.*

Hence the number of independence tests required is $O(n^{2.5})$, if $k = O(n)$, since $p \leqq n$. If k is fixed, then $p = O(r)$, where r is the maximum rank of the M_i, so the number of independence tests is $O(r^{1.5}n)$.

Using the correspondence between paths in the matroid partition auxiliary digraph and paths in the auxiliary digraph of the associated intersection problem, we obtain directly from Theorem 4.2 a similar result for the distance function D. In addition, it is a consequence of Theorem 4.5 that the number of different values of $D(r, s)$ is $O(\sqrt{p})$. Thus we can apply the layered network approach to the partition algorithm.

In counting independence tests, we handle separately those of the form $J_i + e \in \mathcal{I}_i$, in order to reduce the effect of k on the bound. As in § 3, we remember for each i and $e \notin J_i$, whether $J_i + e \in \mathcal{I}_i$. Initially this requires kn independence tests. After an augmentation it can be updated with $O(n)$ independence tests, using the fact that, if $|J_i'| = |J_i|$ and $e \notin J_i \cup J_i'$, then $J_i' + e \in \mathcal{I}_i$ if and only $J_i + e \in \mathcal{I}_i$. (That is, successive J_i of the same cardinality have the same M_i-span; as was mentioned in § 4, this is well known.) So the total number of independence tests of the form $J_i + e \in \mathcal{I}_i$ is $O(kn + pn)$.

We can construct the layered digraph at the beginning of a stage in time $O(pnQ + kn)$, since there are $\leqq pn$ pairs $e, f \in E$ which could be edges of the digraph, and finding the edges of the form (e, s) can be done in $O(kn)$ time. We shall show that only $O(pn)$ independence tests of the form $J_i + e - f \in \mathcal{I}_i$ are required to complete the stage, because at most one such test is required for each (e, f) with $e \in L_i$, $f \in L_{i+1}$, $1 \leqq i < m$. The algorithm is the same depth-first search, with the same rules for marking a vertex useless, which must be justified. Of course, by the analogue of Theorem 4.2, we can mark a vertex useless when the search retreats from it. We also need to show that $D(r, e)$ increases for each internal vertex e of a path on which an augmentation is performed, since these vertices are also marked useless. This is obvious, if $D(r, e) = 1$, since e enters J after the augmentation. Otherwise, using the correspondence between the partition algorithm and the intersection algorithm, we have that $D(r, e) = \frac{1}{2}(1 + d(r, e^i))$, where $e \in J_i$ and e^i is copy i of e. After the augmentation, suppose that $e \in J_j'$. Then $D'(r, e) = \frac{1}{2}(1 + d'(r, e^j))$. We know that $d'(r, e^j) > d(r, e^j) = d(r, e^i) + 1$, so $D'(r, e) > D(r, e)$, as required.

We conclude that the algorithm requires $O(p^{1.5}n + kn)$ independence tests in all. There is also $O(\sqrt{p}\, kn)$ additional work required for finding edges of the form (e, s). Hence we obtain the bound of Theorem 5.1.

REFERENCES

[1] W. H. CUNNINGHAM, *Testing membership in matroid polyhedra*, J. Combin. Theory, B, 36 (1984), pp. 161-188.

[2] E. DINIC, *Algorithm for solution of a problem of maximum flow in a network with power estimation*, Soviet Math. Dokl., 11 (1970), pp. 1277-1280.

[3] J. EDMONDS, *Minimum partition of a matroid into independent sets*, J. Res. Nat. Bur. Stand., 69B (1965), pp. 67-72.

[4] ———, *Submodular functions, matroids, and certain polyhedra*, in Combinatorial Structures and Their Applications, R. K. Guy et al. eds., Gordon and Breach, New York, 1970, pp. 69-87.

[5] ———, *Matroid intersection*, Ann. Discr. Math., 4 (1979), pp. 39-49.

[6] J. EDMONDS AND D. R. FULKERSON, *Transversals and matroid partition*, J. Res. Nat. Bur. Stand., 69B (1965), pp. 147-153.

[7] S. EVEN AND R. E. TARJAN, *Network flow and testing graph connectivity*, this Journal, 4 (1975), pp. 507-518.

[8] A. FRANK, *Finding feasible vectors of Edmonds-Giles polyhedra*, J. Combin. Theory, B, 36 (1984), pp. 221-239.

[9] H. N. GABOW AND M. STALLMAN, *Efficient algorithms for graphic matroid intersection and parity*, extended abstract, in Automata, Languages and Programming, Lecture Notes in Computer Science, Springer, Berlin, 1985.

[10] J. E. HOPCROFT AND R. M. KARP, *A $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, this Journal, 2 (1973), pp. 225-231.

[11] E. L. LAWLER, *Matroid intersection algorithms*, Math. Programming, 9 (1975), pp. 31-56.

[12] ———, *Combinatorial Optimization: Networks and Matroids*, Holt-Rinehart-Winston, New York, 1976.

[13] E. L. LAWLER AND C. MARTEL, *Finding maximal polymatroidal network flows*, Math. Oper. Res., 7 (1982), pp. 334-347.

[14] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[15] P. SCHÖNSLEBEN, *Ganzzahlige Polymatroid-Intersektions-Algorithmen*, Ph.D. thesis, ETH Zürich, 1980.

[16] É. TARDOS, C. A. TOVEY AND M. A. TRICK, *Layered augmenting path algorithms*, Math. Oper. Res., to appear.

[17] D. J. A. WELSH, *Matroid Theory*, Academic Press, New York, 1976.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
005

AN INHOMOGENEITY IN THE STRUCTURE OF KARP DEGREES*

KLAUS AMBOS-SPIES†

Abstract. We show that there is a recursive nonzero Karp (polynomial time many-one) degree which is not supremum of a minimal pair, i.e. of an incomparable pair of degrees with infimum **0**, the degree of polynomial time computable sets. By existence of minimal pairs, this implies that there are nonisomorphic initial segments of Karp degrees.

Key words. polynomial degrees, polynomial time reducibilities, relative complexity, NP-sets

AMS(MOS) subject classifications. 68C25, 03D15, 03D30

Introduction: Cook [4] and Karp [6] introduced polynomial time bounded counterparts of the recursive Turing and many-one reducibilities, respectively. These reducibility notions serve for a classification of the solvable but intractable problems according to their relative complexity. While Cook's reducibility notion is the more adequate formalization of the underlying intuitive notion of feasibly computable reducibility, it turned out that the more restrictive but conceptually simpler Karp reducibility suffices for most practical purposes. So it has been shown that all problems which are known to be NP-complete in the sense of Cook also constitute a single Karp degree, whence for the study of problems in NP Karp's reducibility notion is usually preferred (see e.g. Garey and Johnson [5]). Moreover, membership in NP is downward inherited by Karp reducibility but not by Cook reducibility under the reasonable hypothesis that NP $\neq$ co-NP.

The study of the structure of the polynomial degrees induced by these reducibility notions on the recursive sets began with Ladner [7]. He showed, for Cook and Karp reducibility, that the degrees form a dense upper semi-lattice with least element **0**, the degree of polynomial time computable sets, but do not form a lattice. Moreover, any degree $>\mathbf{0}$ splits, i.e. is the supremum of two lesser ones, and there are minimal pairs, i.e. incomparable degrees with infimum **0**. By refining Ladner's technique, Landweber et al. [8] and Chew and Machtey [3] extended some of his results. Recently we unified and extended the cited results on polynomial degrees by showing that any countable distributive lattice can be embedded in any interval of degrees by maps which preserve the greatest or least element, respectively [1], [2]. In [1] we also give a first example of an algebraic property which distinguishes the Karp degrees from the Cook degrees: The Karp but not the Cook degrees form a *distributive* upper semi-lattice. This result indicates that the structure of Karp degrees is more well behaved than that of Cook degrees.

The above results show that if P $\neq$ NP then not all problems in NP $-$ P are NP-complete but quite on the contrary the structure of the degrees of NP-sets will be extremely rich, e.g. allowing any countable distributive lattice—and thus any countable partial ordering—to be embedded into it.

If we analyze the results obtained so far on the polynomial degrees, then we observe that they hint at homogeneity of the structure of these degrees. For most of the phenomena, like sublattices or suborderings, which have been studied for the polynomial degrees, once one had established that they occur somewhere in the

* Received by the editors April 3, 1984, and in revised form August 15, 1985.
† Lehrstuhl für Informatik II, Universität Dortmund, D-4600 Dortmund 50, West Germany.

structure of degrees one could also show that they in fact occur in any interval, and for many algebraic properties, like being supremum or infimum of an incomparable pair of degrees, one could not only show that they hold for some degees but also that they are shared by all nonzero degrees. For instance, all known facts about the structure of the Karp degrees of NP-sets (assuming $P \neq NP$) have been obtained by proving them for *all* nontrivial initial segments of the partial ordering of the Karp degrees of recursive sets and then using the observation that the degrees of NP-sets coincide with the initial segment $[\mathbf{0}, \mathbf{0}']$, where $\mathbf{0}'$ is the degree of the NP-complete problems. So it is natural to ask whether any two nontrivial initial segments (or in fact any two proper intervals) of polynomial degrees are isomorphic, and whether all nonzero degrees share the same elementary algebraic properties.

For Karp degrees, we here give a negative answer to these questions. We show that there is a nonzero Karp degree $\mathbf{a}$ which is not supremum of any minimal pair, i.e.

$$(*) \qquad \forall \mathbf{b} < \mathbf{a}\, \forall \mathbf{c} < \mathbf{a} \qquad (\mathbf{b} | \mathbf{c} \,\&\, \mathbf{0} = \mathbf{b} \cap \mathbf{c} \Rightarrow \mathbf{a} > \mathbf{b} \cup \mathbf{c}).$$

So if, by Ladner's minimal pair theorem, we let $\mathbf{b}$ be any Karp degree which is supremum of a minimal pair, then the intervals $[\mathbf{0}, \mathbf{a}]$ and $[\mathbf{0}, \mathbf{b}]$ are not isomorphic and the degrees $\mathbf{a}$ and $\mathbf{b}$ can be distinguished by the elementary algebraic property $(*)$.

1. Preliminaries. Let Σ be a finite alphabet which contains the letters 0 and 1, and let Σ^* be the set of (finite) strings over Σ. We will denote elements of Σ^* by lower case letters from the end of the alphabet, while capital letters will denote recursive subsets of Σ^*. $|x|$ is the length of x and $|A|$ is the cardinality of A. In our notation we do not distinguish between a set and its characteristic function. So $x \in A$ iff $A(x) = 1$ and $x \notin A$ iff $A(x) = 0$. $\mathbb{N}$ is the set of natural numbers; k, m, n, p denote elements of $\mathbb{N}$.

P (NP) is the class of subsets of Σ^* which can be (non)deterministically computed in polynomial time. We let $\{P_n : n \in \mathbb{N}\}$ be a recursive enumeration of P, and let $\tau : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a bijection which is computable and invertible in polynomial time.

A set A is *Karp* (or *polynomial time many-one*) *reducible* to a set B, $A \leqq_m^p B$ for short, if there is a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that $\forall x(A(x) = B(f(x)))$ (in this case we say $A \leqq_m^p B$ via f). Sets A and B are *Karp equivalent*, $A \equiv_m^p B$, if $A \leqq_m^p B$ and $B \leqq_m^p A$. The *Karp degree* of A, denoted by $\deg A$, is the class of all sets which are Karp equivalent to A. Bold face lower case letters denote Karp degrees of recursive sets. The partial ordering on the Karp degrees induced by $\leqq_m^p$ is denoted by $\leqq$. As usual we write $\mathbf{a} < \mathbf{b}$ if $\mathbf{a} \leqq \mathbf{b}$ and $\mathbf{a} \neq \mathbf{b}$. Two degrees $\mathbf{a}$ and $\mathbf{b}$ are incomparable, $\mathbf{a} | \mathbf{b}$, if $\mathbf{a} \not\leqq \mathbf{b}$ and $\mathbf{b} \not\leqq \mathbf{a}$. The supremum of two degrees $\mathbf{a}$ and $\mathbf{b}$ is denoted by $\mathbf{a} \cup \mathbf{b}$, the infimum (if it exists) by $\mathbf{a} \cap \mathbf{b}$. Note that, for $A \oplus B = \{0x : x \in A\} \cup \{1y : y \in B\}$, $\deg A \oplus B = \deg A \cup \deg B$. Also note that—ignoring the empty set $\varnothing$ and the set Σ^* which both constitute their own degrees—there is a least Karp degree, denoted by $\mathbf{0}$, which contains just the elements of P.

Finally recall that a recursive function $f : \mathbb{N} \to \mathbb{N}$ is *polynomially honest* if there is a polynomial p such that $f(n)$ can be computed in $p(f(n))$ steps. Note that the step counting functions of Turing machines are polynomially honest. So any recursive function is dominated by a strictly increasing polynomially honest function.

2. The theorem. Recall that two polynomial degrees $\mathbf{a}$ and $\mathbf{b}$ form a minimal pair if $\mathbf{a}$ and $\mathbf{b}$ are incomparable and $\mathbf{a} \cap \mathbf{b} = \mathbf{0}$.

THEOREM 1. *There is a recursive Karp degree* $\mathbf{a} > \mathbf{0}$ *which is not a supremum of any minimal pair of Karp degrees.*

To prove the theorem, we first reduce it to a simpler theorem.

DEFNITION. Let A be a recursive set. A *p-splitting* of A is a pair of sets A_0 and A_1 such that $A_0 = A \cap B$ and $A_1 = A \cap \bar{B}$ for some polynomial time computable set B. A p-splitting A_0, A_1 of A is *strict* if $\deg A_0$ and $\deg A_1$ form a minimal pair.

Note that for a p-splitting A_0, A_1 of A, $A \equiv_m^p A_0 \oplus A_1$, i.e. $\deg A = \deg A_0 \cup \deg A_1$.

LEMMA 1. *Let A be a recursive set. Then A has a strict p-splitting iff* $\deg A$ *is a supremum of a minimal pair.*

Proof. If A_0, A_1 is a strict p-splitting of A, then $\deg A_0 \cup \deg A_1 = \deg A$ and $\deg A_0$ and $\deg A_1$ form a minimal pair. For a proof of the other direction let $\mathbf{b}$, $\mathbf{c}$ be a minimal pair such that $\mathbf{b} \cup \mathbf{c} = \deg A$ and fix sets $B \in \mathbf{b}$ and $C \in \mathbf{c}$. Then $A \equiv_m^p B \oplus C$ and thus there is a polynomial time computable function f such that $A \leqq_m^p B \oplus C$ via f. Now the set $D = \{x : f(x) = 0y, y \in \Sigma^*\}$ is in P, whence $A_0 = A \cap D$ and $A_1 = A \cap \bar{D}$ is a p-splitting of A. Furthermore, $A_0 \leqq_m^p B \oplus \varnothing$ via f and $A_1 \leqq_m^p \varnothing \oplus C$ via f whence $\deg A_0 \leqq \mathbf{b}$ and $\deg A_1 \leqq \mathbf{c}$. So, by $\mathbf{b} \cap \mathbf{c} = \mathbf{0}$, $\deg A_0 \cap \deg A_1 = \mathbf{0}$ too. Moreover, by incomparability of $\mathbf{b}$ and $\mathbf{c}$ and by $\deg A = \deg A_0 \cup \deg A_1 = \mathbf{b} \cup \mathbf{c}$, it follows that $\deg A_0$ and $\deg A_1$ are incomparable, whence A_0, A_1 is a strict p-splitting of A. □

Now, by Lemma 1, Theorem 1 is an immediate consequence of the following theorem.

THEOREM 2. *There is a recursive but not polynomial time computable set which has no strict p-splitting.*

The proof of Theorem 2 requires the following technical lemma.

LEMMA 2. *There is a recursive function $f : \mathbb{N} \to (\mathbb{N} \times \mathbb{N}) \cup \{\uparrow\}$ such that*

(i) $f(0) = \uparrow$,

(ii) *if* $f(m) \in \mathbb{N} \times \mathbb{N}$ *then* $f(m+1) = \uparrow$,

(iii) *if* $f(m) = (n, k)$ *then* $0^m \in P_n$ *and* $0^{m+1} \notin P_n$, *and*

(iv) *if* $P_n \cap \{0\}^*$ *and* $\bar{P}_n \cap \{0\}^*$ *are infinite then* $\forall k \, \exists m (f(m) = (n, k))$.

Intuitively, the function f selects infinitely many mutually disjoint pairs of consecutive members of $\{0\}^*$ such that, for any polynomial time computable set B which splits $\{0\}^*$ into two infinite parts, there are infinitely many such pairs corresponding to B, each having the property that the first but not the second element of the pair belongs to B.

For the proof of Lemma 2 we require the following notation. Given numbers n and m such that $n \leqq m$, we say m is *n-positive* if $0^m \in P_n$ and $0^{m+1} \notin P_n$ and m is *n-negative* if $0^{m+1} \in P_n$ and $0^{m+2} \notin P_n$. Note that a number cannot be both n-positive and n-negative and that m is n-negative iff $m+1$ is n-positive.

Proof of Lemma 2. The function f is inductively defined by

$$f(m) = \begin{cases} (n, k) & \text{if } m > 0,\ f(m-1) = \uparrow,\ m \text{ is } n\text{-positive, and} \\ & \tau(n, k) \text{ is minimal such that } m \text{ is } n\text{-positive or} \\ & n\text{-negative and } \forall p < m \ (f(p) \neq (n, k)), \\ \uparrow & \text{otherwise.} \end{cases}$$

Obviously f is recursive and satisfies (i). Furthermore, by a straightforward induction on m, (ii) and (iii) are satisfied and

$$(1) \qquad \forall m, n, k \qquad (f(m) = (n, k) \Rightarrow \forall p < m (f(p) \neq (n, k))),$$

whence for each (n, k) there is at most one m with $f(m) = (n, k)$. To verify property (iv) we first prove the following claim.

Claim. If k, m, n are numbers such that $n \leqq m$, $f(m-1) = \uparrow$, m is n-positive or n-negative, and $\forall p < m \ (f(p) \neq (n, k))$, then there is $m' \geqq m$ such that $\tau \circ f(m') \leqq \tau(n, k)$.

Proof. The proof of the claim is by induction on $\tau(n, k)$. Fix numbers k, m, n satisfying the premise of the claim. W.l.o.g. $\tau(n, k)$ is minimal such that, for some $m' \geqq m$, $f(m'-1) = \uparrow$, m' is n-positive or n-negative and $\forall p < m'\ (f(p) \neq (n, k))$, since otherwise the claim follows by inductive hypothesis. So if m is n-positive, then $f(m) = (n, k)$ while if m is n-negative then $f(m) = \uparrow$, $m+1$ is n-positive and $f(m+1) = (n, k)$, proving the claim.

Now, for a proof of (iv), fix n and for a contradiction assume that

$$|\{0\}^* \cap P_n| = |\{0\}^* \cap \bar{P}_n| = \infty \tag{2}$$

but $\exists k\, \forall m\ (f(m) \neq (n, k))$, say k_0 is such a k. Note that, by (2),

$$\overset{\infty}{\exists} p \qquad (0^p \in P_n \text{ and } 0^{p+1} \notin P_n),$$

i.e. there are infinitely many n-positive numbers. So, since f is 1-1 on its domain (following (1)), we may choose $m \geqq n$ such that

$$m > \max\{m' \colon \tau \circ f(m') \leqq \tau(n, k_0)\} + 1 \tag{3}$$

and m is n-positive. By the latter, $m-1$ is n-negative. Moreover, by (ii), either $f(m-2) = \uparrow$ or $f(m-1) = \uparrow$. So, by choice of k_0 and by the claim above, there is $m' \geqq m-1$ such that $\tau \circ f(m') \leqq \tau(n, k_0)$ contrary to (3). □

Proof of Theorem 2. We will define a recursive set $A \notin \mathrm{P}$ which has no strict p-splitting. For this sake let f be a function as in Lemma 2 and let $g : \mathbb{N} \to \mathbb{N}$ be a polynomially honest and strictly increasing function such that $f(n)$ can be computed in less than $g(n)$ steps. Note that, by honesty of g, there is a polynomial p_g such that for given n we can tell in $p_g(n)$ steps whether $n = g(m)$ for some m and if so for which one.

Now $A \subseteq \{0\}^*$ is defined by

$$A(0^0) = 0$$

and

$$A(0^{m+1}) = \begin{cases} A(0^m) & \text{if } f(m+1) = \uparrow, \\ 1 - P_k(0^{g(m+1)}) & \text{if } f(m+1) = (n, k). \end{cases}$$

Obviously A is recursive. Moreover, by property (ii) of f,

$$f(m) \in \mathbb{N} \times \mathbb{N} \quad \Rightarrow \quad A(0^m) = A(0^{m+1}). \tag{4}$$

Now, for a proof that A has no strict p-splitting, let any $B \in \mathrm{P}$ be given. If $A \cap B$ or $A \cap \bar{B}$ is finite then the p-splitting of A by B is not strict. So w.l.o.g. we may assume that $A \cap B$ and $A \cap \bar{B}$ are infinite. Since $A \subseteq \{0\}^*$ this implies $|\{0\}^* \cap B| = |\{0\}^* \cap \bar{B}| = \infty$. So it suffices to prove the following claim.

Claim. If $|\{0\}^* \cap P_n| = |\{0\}^* \cap \bar{P}_n| = \infty$ then there is a recursive set B_n such that $B_n \leqq_m^p A \cap P_n$, $B_n \leqq_m^p A \cap \bar{P}_n$ and $B_n \notin \mathrm{P}$.

Proof. Fix n such that the premise of the claim holds. The set B_n is defined by

$$\forall x (x \in B_n \Leftrightarrow \exists m, k (x = 0^{g(m)} \,\&\, f(m) = (n, k) \,\&\, 0^m \in A)). \tag{5}$$

Note that, by (4),

$$x \in B_n \quad \Leftrightarrow \quad \exists m, k \qquad (x = 0^{g(m)} \,\&\, f(m) = (n, k) \,\&\, 0^{m+1} \in A).$$

Moreover, by property (iii) of f,

$$\forall m (\exists k (f(m) = (n, k)) \Rightarrow 0^m \in P_n \,\&\, 0^{m+1} \notin P_n).$$

So, for functions h_n^i, $i=0, 1$, defined by

$$h_n^i(x)=\begin{cases}1 & \text{if there are no numbers } m \text{ and } k \text{ such that } x=0^{g(m)} \text{ and}\\ & f(m)=(n,k),\\ 0^{m+i} & \text{otherwise, where } x=0^{g(m)},\end{cases}$$

we have

$$(6)\qquad \forall x \quad (x\in B_n \Leftrightarrow h_n^0(x)\in A\cap P_n \Leftrightarrow h_n^1(x)\in A\cap \bar{P}_n).$$

Note that the function h_n^i can be computed in polynomial time. Namely, for given x, first use $p_g(|x|)$ steps to search for an m such that $x=0^{g(m)}$. If such an m exists, in $|x|$ steps compute $f(m)$. Now if $f(m)=(n,k)$ for some k, then $h_n^i(x)=0^{m+i}$. If any of the above conditions fail, set $h_n^i(x)=1$. It follows, by (6), that $B_n\leqq_m^p A\cap P_n$ via h_n^0 and $B_n\leqq_m^p A\cap \bar{P}_n$ via h_n^1. It remains to show that $B_n\notin \mathrm{P}$, i.e. $B_n\neq P_k$ for each k. So fix k. Note that, by property (iv) of f, $\forall k\,\exists m\ (f(m)=(n,k))$, whence we may choose m such that $f(m)=(n,k)$. Then $A(0^m)=1-P_k(0^{g(m)})$, whence by (5), $B_n(0^{g(m)})\neq P_k(0^{g(m)})$.

This completes the proof of the claim.

It remains to show that $A\notin \mathrm{P}$. Consider the set $E=\{0^{2^n}: n\in\mathbb{N}\}$. Obviously $E\in \mathrm{P}$, say $E=P_n$. Then $|\{0\}^*\cap P_n|=|\{0\}^*\cap \bar{P}_n|=\infty$, whence by the claim above there is some set B_n such that $B_n\notin \mathrm{P}$ and $B_n\leqq_m^p A\cap E$. Since $A\cap E\leqq_m^p A$ this implies $A\notin \mathrm{P}$. □

COROLLARY. *There are Karp degrees* $\mathbf{a}, \mathbf{b}>\mathbf{0}$ *such that the partial orderings of Karp degrees below* $\mathbf{a}$ *and* $\mathbf{b}$, *respectively, are not isomorphic.*

Proof. Choose $\mathbf{a}$ as in Theorem 1 and, by Ladner [7], let $\mathbf{b}$ be the supremum of some minimal pair, say $\mathbf{c}$, $\mathbf{d}$. Then $[\mathbf{0}, \mathbf{a}]$ and $[\mathbf{0}, \mathbf{b}]$ are not isomorphic, since any isomorphism from $[\mathbf{0}, \mathbf{b}]$ to $[\mathbf{0}, \mathbf{a}]$ would map the degrees $\mathbf{c}$ and $\mathbf{d}$ to a minimal pair with supremum $\mathbf{a}$, which is impossible. □

Since the property of being supremum of some minimal pair can be expressed by some first order formula in the language of partial orderings, the partial orderings of $[\mathbf{0}, \mathbf{a}]$ and $[\mathbf{0}, \mathbf{b}]$ are not even elementarily equivalent.

3. Concluding remarks. By refining our proofs for Theorem 2 and Lemma 2, we can obtain a set A computable in exponential but not polynomial time without strict p-splittings. Also note that Theorem 2 a fortiori holds for Cook reducibility in place of Karp reducibility. Still we do not obtain a proof of Theorem 1 for Cook degrees. The reason is that in the reduction of Theorem 1 to Theorem 2 the proof of Lemma 1 implicitly exploits distributivity of the Karp degrees whence it does not translate to the nondistributive (see [1]) structure of Cook degrees.

REFERENCES

[1] K. AMBOS-SPIES, *On the structure of polynomial time degrees*, in STACS 84, Symposium on Theoretical Aspects of Computer Science, M. Fontet and K. Mehlhorn, eds., Lecture Notes in Computer Science 166, Springer Verlag, Berlin, 1984, pp. 198-208.

[2] ———, *Sublattices of the polynomial time degrees*, Inform. Control, 65 (1985), pp. 63-84.

[3] P. CHEW AND M. MACHTEY, *A note on structure and looking back applied to the relative complexity of computable functions*, J. Comput. System Sci., 22 (1981), pp. 53-59.

[4] S. A. COOK, *The complexity of theorem-proving procedures*, in Proc. 3rd Annual ACM Symposium on the Theory of Computing, ACM, New York, 1971, pp. 151-158.

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

[6] 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.

[7] R. E. LADNER, *On the structure of polynomial time reducibility*, J. Assoc. Comput. Mach., 22 (1975), pp. 155-171.

[8] L. H. LANDWEBER, R. J. LIPTON AND E. L. ROBERTSON, *On the structure of sets in* NP *and other complexity classes*, Theor. Comput. Sci., 15 (1981), pp. 181-200.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
006

HEAPS ON HEAPS*

GASTON H. GONNET† AND J. IAN MUNRO†

Abstract. As part of a study of the general issue of complexity of comparison based problems, as well as interest in the specific problem, we consider the task of performing the basic priority queue operations on a heap. We show that in the worst case:

$\lg \lg n \pm O(1)$ comparisons are necessary and sufficient to insert an element into a heap. (This improves the previous upper and lower bounds of $\lg n$ and $O(1)$.)

$\lg n + \log^* n \pm O(1)$ comparisons are necessary and sufficient to replace the maximum in a heap. (This improves the previous upper and lower bounds of $2 \lg n$ and $\lg n$.)

$1.625n + O(\lg n \log^* n)$ comparisons are sufficient to create a heap. 1.37... n comparisons are necessary not only in the worst case but also on the average.

Here lg indicates the logarithm base 2 and log* denotes the iterated logarithm or number of times the logarithm base 2 may be taken before the quantity is at most 0.

Key words. heap, comparisons, lower bound

1. Introduction. One of the most elegant of storage structures is the representation of a priority queue as a heap. A heap [8], [1], [4], [2] is defined as a structure on locations 1 through n of an array with the property that the element in location i is smaller than that in location $\lfloor i/2 \rfloor$, thus inducing a complete binary tree with the property that the value of the parent is greater than that of the child. Such a pointer free representation has been called an *implicit data structure* [5]. It is well known that a heap enables us to perform the basic priority queue operations, insert an element and rebalance after extracting the maximum in $O(\lg n)$ basic operations. Furthermore, a heap can be created in about $2n$ comparisons [1], [4]. These results are very old by the standards of our field, dating back to the decade before the last. Our aim, in this paper, is to re-examine the algorithms for performing these basic operations on a heap. We are able to establish new upper and lower bounds on the number of comparisons necessary in the worst case to perform these tasks. While our algorithms may be of some interest in implementing heaps, we are using this structure primarily as a paradigm for the study of computation complexity of comparison based problems.

2. Insertion and promotion. Observe that the elements on the path from any node to the root must be in sorted order. Our idea is simply to insert the new element by performing a binary search on the path from location $n+1$ to 1. As, for $n \geqq 2$, this path contains $\lfloor \lg (n+1) \rfloor$ old elements, the algorithm will require $\lceil \lg (\lfloor 1 + \lg (n+1) \rfloor) \rceil$ comparisons in the worst case. This expression may be rewritten as $\lceil \lg \lg (n+2) \rceil$ which also indicates the number of comparisons required when $n = 0$ or 1. We note that the number of moves will be the same as those required in a carefully coded standard algorithm. It was this simple observation, also used in [3] and [6] for priority queues on a bounded domain, that sparked our interest in heap manipulation algorithms. Indeed, it is very useful as a basis for the extraction algorithm presented in the next section. However, the reader may find the fact that this bound is tight more interesting.

* Received by the editors August 7, 1984, and in revised form August 23, 1985. A preliminary report on some of the results in this paper appeared in ICALP 1982. This research has been supported by the Natural Sciences and Engineering Research Council of Canada under grants A8237 and A3353.

† Data Structuring Group, Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada.

THEOREM 1. *$\lceil \lg \lg (n+2) \rceil$ comparisons are sufficient and $\lceil \lg \lg (n+2) \rceil - 2$ are necessary in the worst case to insert an element into a heap of size n.*

Proof. The upper bound has been given. The lower bound comes from considering a path in the updated heap from the root through the new element and on to a leaf. Such a path is of length either $\lceil \lg (n+2) \rceil$ or $\lfloor \lg (n+2) \rfloor$ and so contains at least $\lfloor \lg (n+2) \rfloor - 1$ values from the old heap.

We develop an adversary strategy as follows. The adversary answers queries in a manner consistent with the elements in positions 2^i through $2^{i+1}-1$ of the old heap being of ranks 2^i through $2^{i+1}-1$ in the structure, although not necessarily in consecutive order among themselves. We refer to such elements, which were the same distance from the root in the old heap, as being of the same *level.* The adversary answers queries involving the new element in a manner consistent with it falling in one of the $\lceil \lg (n+1) \rceil + 1$ interlevel gaps. Indeed it will answer any comparison so as to maximize the number of gaps into which the new element could fall.

On the other hand, observe that if k elements from any level of the old heap occur on such a path, then at least $k-1$ comparisons between elements of that level must have been performed. Hence the algorithm outlined above appears to be optimal. We must, however, take into account that every such path could be of length $\lfloor \lg (n+1) \rfloor$ rather than $\lceil \lg (n+1) \rceil$ as in our algorithm, and that it may contain an element from the bottom (incomplete) level of the old heap. Taking these into account we can make the more modest claim that $\lceil \lg \lg (n+2) \rceil - 2$ comparisons are necessary in the worst case. □

We emphasize that the above bound is on the number of comparisons required to perform on insertion. The number of data items that are moved is also an interesting metric. Our method and the standard one use exactly the same number of moves. If the new element is larger than any currently in the structure this number is $1 + \lfloor \lg (n+1) \rfloor$. Under such circumstances, and if the old heap is arranged in the manner suggested in the proof, it follows immediately that this number of elements must be moved to perform an insertion.

3. Extraction and demotion. Based on the insertion algorithm of the previous section, we can easily extract the maximum and reorder the heap in $\lg n + \lg \lg n$ comparisons. Simply let the "empty location" filter down to the bottom level ($\lceil \lg (n+1) \rceil - 1$ comparisons) and then perform an insertion of the element previously in location n (or a new element if one is to be added) along the path from the empty spot to the root. This bound can, however, be improved as follows. For simplicity assume we are removing the maximum and simultaneously inserting a new element.

```
begin
  Remove the maximum, creating a "hole" at the top of the heap;
  Find the path of maximum children down r levels to A(i);
  if New element > A(i)
  then Perform a binary search with new element along path of length r
  else Promote each element on the path to the location of its parent and
    recursively apply the method starting at location A(i)
end
```

The number of comparisons required is, then,

$$C(n) = r + 1 + \max(\lceil \lg (r+1) \rceil, C(\lceil n/2^r \rceil)).$$

Choosing $r \simeq \lceil \lg n - \lg \lg n \rceil$, we see that $C(n)$ can be reduced to $\lceil \lg n \rceil + \log^* n$ where $\log^*(x) = 0$ for $x \leqq 1$ and $\log^* n = \log^* (\lceil \lg n \rceil) + 1$. Indeed the optimal choice

of r may differ by 1 from the bound suggested and the bound on $C(n)$ may also be reduced by 1 for certain values of n. However, we omit these awkward details.

As it happens, this algorithm, with judicious choice of r, essentially minimizes the number of comparisons necessary to perform the update. The key idea of our proof is, very informally, to give outcomes to comparisons in a manner consistent with the worst case of the algorithm outlined and to provide extra information so that any algorithm "might as well" have followed the given technique. The lower bound then follows from the optimal choice of r in the method presented. More formally:

THEOREM 2. $\lg n + \log^* n \pm O(1)$ *comparisons are necessary and sufficient to perform the operation replace maximum on a heap.*

Proof. Suppose that the heap upon which the extraction is to be performed is of the form indicated in Fig. 1.

(i) The largest j (j unknown but $\leqq \lg n$) elements are arranged along a path from the root. Call this path of elements j the *shaft.*

(ii) The smallest elements in the structure are the ($n/2^j$ or so) descendants of the bottom element of the shaft.

(iii) The other elements lie between these, satisfying the heap property, and furthermore, tend to be arranged so that the higher their closest shaft ancestor is located, the larger the element.

(iv) The new element is smaller than all shaft elements but larger than all others.

The crucial property is that the last element of the shaft must be determined in order to perform the update. This follows since on removal of the maximum, the shaft element of level i is the $(i-1)$st largest element in the heap and so must be moved to a higher level. On the other hand, the largest of the small elements has the property that it cannot be raised to a higher level as there are not enough smaller elements to support it.

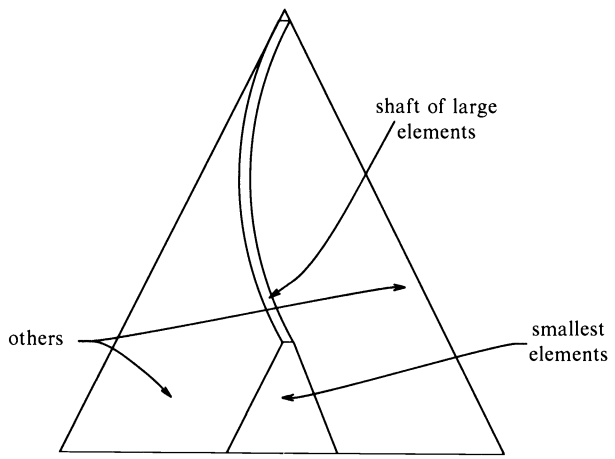


FIG. 1. *A hard case for heap update "the way it is".*

The adversary strategy is based on viewing the information which has been gathered as finding the path of maximum children to some level (see Fig. 2). Call this path the *chain.* The chain partitions the remaining elements into those which are descendants of the last chain element (the *inside*) and those which are not (the *outside*). Notice that the chain (what we have learned) and the shaft (what we are to discover) coincide for the length of the shorter. The general approach of the adversary is to respond to

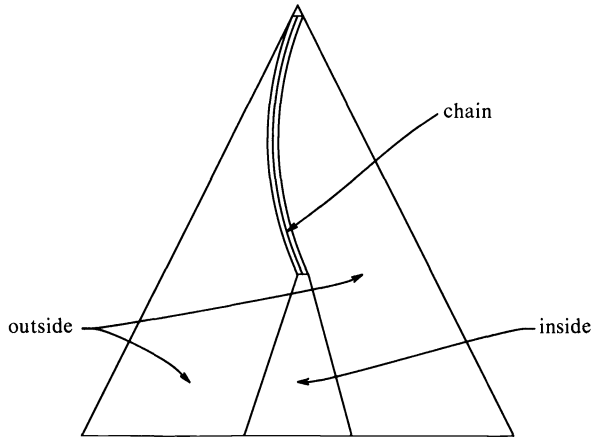


FIG. 2. *Information yielded by the adversary "what we know".*

queries so that the algorithm learns (almost) nothing about the relation of the inside elements to any but their ancestors. Furthermore, note that anything learned about the outside alone is of no help in determining the end of the shaft. The chain is, of course, permitted to be extended one level per comparison, and the algorithm can always check to see whether the shaft is shorter than the chain.

The outcome of comparisons is given below:

Chain element—New element
—Answer according to the worst case as implied by the algorithm
Chain element—Outside element
—Answer as Chain element—New element.
Inside—Outside
—Outside element is larger, supply the additional information that the chain is extended by the one element which is not an ancestor of the (hitherto) inside element considered.
Inside—New
—As inside-outside. The new element is declared to be smaller and the additional information is given that the chain can be extended by an element avoiding the path to the hitherto inside element
Outside—New
—The new element is smaller. No other information need be given.
Outside—Outside
—Answer arbitrarily but consistently.
Inside—Inside (this may be a compound step)
—If both elements are descendants of the same child of the last chain element, give an arbitrary outcome and extend the chain 1 step to avoid both.
—If both are children of the last chain member, extend the chain arbitrarily.
—If one is a child of the last chain member and the other is not, extend the chain to include the former.
—Otherwise we cannot avoid giving up some information about inside elements and so delay advancing the chain as sketched below.

If the next comparison does not involve an inside element it is answered as indicated above. If it does involve at least 1 inside element but is not of this (awkward)

subtype, then the outcome is as indicated above and the chain is extended 2 steps avoiding all hitherto inside elements that were involved in comparisons. We must be careful of a minor lacuna that this strategy may prohibit an element on the chain from being the end of the shaft. It will, however, not prevent 2 elements in a row from being the end. Hence the lower bound is weakened by at most one comparison over all.

This leaves the case in which the next comparison involves a pair of inside elements which are proper descendants of different children of the chain end. If one is a grandchild of the chain end, it wins the comparison and the chain is extended two positions as indicated above. Otherwise, we declare the higher element to be larger and defer chain extension for the last time. On the next query involving an inside element we follow the basic approach by the previous query, except that the chain may be extended 3 steps. This follows since at most 6 elements on the inside have been involved in comparisons and there are 8 subtrees three steps from the chain end, and so an "open" node may be found to which the chain may be extended. Again some chain elements may be precluded from being the chain end, but no more than 3 in a row. Hence we see the algorithm presented is within one comparison of optimal. □

A bound on performing a simple extraction follows easily.

COROLLARY 3. $\lg n + \log^* n \pm O(1)$ *comparisons are necessary and sufficient to remove the maximum element from a heap and reconstitute the heap structure.*

4. Creating a heap. The usual algorithm for creating a heap [1] requires $2n - O(\lg n)$ comparisons. It is most easily described by a call to Create $(A, 1, n)$ which creates a heap in place on elements in locations 1 to n of the array A.

```
Create (A, i, n)
  Do case 2 * i: n
    =if A(i) < A(n) then Swap (A(i), A(n))
    >do nothing
    <Begin
        Create (A, 2 * i, n)
        Create (A, 2 * i + 1, n);
        Perform replace maximum operation as if a large element in
        A(i) has been replaced by the actual value of A(i)
      end
```

Using the "standard" replace maximum technique this leads to the recurrence

$$T(n) = 2T(n/2) + 2 \lg n, \qquad T(2^k - 1) = 2(2^k - k - 1)$$

and hence the stated bound on the number of element to element comparisons. Clearly the method of the previous sections can be employed to reduce this bound. This is an improvement, but disappointing, as about 1.77... n comparisons are required. By way of contrast the following lower bound is easily derived.

THEOREM 4. 1.3644... $n + O(\lg n)$ *comparisons are necessary, not only in the worst case, but also on the average to create a heap on n elements.*

Proof. A reasonably straightforward enumeration shows that there are $H(n) = n!/\Pi t_i$ valid heaps on a set on n numbers where t_i is the size of the heap rooted at node i. A lower bound on the average number of comparisons required to permit one of $n!$ possible input sequences to one of these orders is

$$\lg (n!/H(n)) = \sum \lg t_i. \qquad \square$$

We are unable to give an algorithm which achieves this bound. Indeed we conjecture that it is not achievable and that the algorithm below minimizes the number of comparisons to create a heap on n elements in the worst case.

THEOREM 5. $\frac{13}{8}n + O(\log^* n \lg n)$ *comparisons are sufficient to construct a heap on* n *elements.*

Proof. We will outline a method of constructing a heap on 2^k elements using $\frac{13}{8}2^k - k - 2$ comparisons ($k \geqq 3$). A heap on n nodes can be viewed as a maximum element, a heap on $2^k - 1$ elements ($k = \lceil \lg(n-1) - 1 \rceil$ or $\lfloor \lg(n-1) - 1 \rfloor$) and a heap on the remaining nodes. Using the technique below to form structures of size $2^k - 1$ for appropriate values of $k < \lg n$, a set of at most $\lg n$ such heaps are grafted in about $(\lg n)(\lg n + \log^* n)/2$ comparisons using the technique of the preceding section. The $(\lg n)^2/2$ term in this expression balances the "$-k$" term in the following construction for $n = 2^k$, leaving the bound claimed.

As seen in the discussion above, what we require is a method of constructing a heap of size $2^k - 1$. We find it easier to express our method for size 2^k. Since these structures are created and used serially, the task is easily completed by "throwing away" the single element on the "bottom level". The basis of the method is the formation of binomial trees of size 2^i (see [7]). As illustrated in Fig. 3, this is simply a tree structure on 2^i elements such that

(i) a single element is a binomial tree of order 2^0;

(ii) a binomial tree of order 2^i is constructed from two of order 2^{i-1} by making the smaller of the maximum values in these trees a child of the larger.

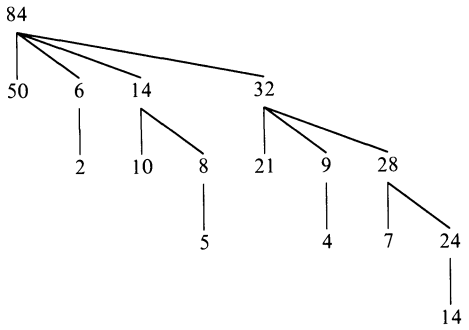


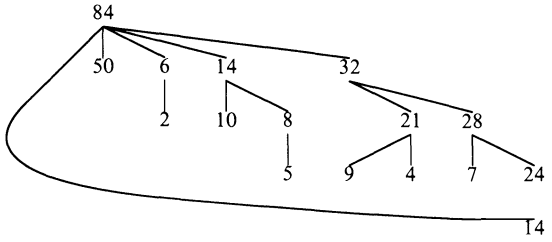
FIG. 3. *A binomial tree of order* 16.

Our method proceeds as follows; the procedure Convert (see also Fig. 4) converts a binomial tree to a heap.

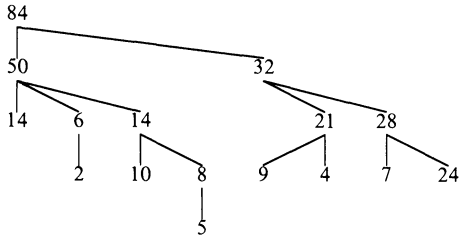
```
Procedure Convert (T, 2^r)
Begin
  Convert (subtree of root of order 2^(r-1), 2^(r-1));
  This leaves an "extra element" on the bottom level
  of this heap, make it a child of the root of T;
  We now have 1 binomial tree of order 2^i
    (i = 1, ···, r-2) and 2 singleton nodes
    all hanging from the root;
  Construct a binomial tree, S, of order 2^(r-1) from these;
  Convert (S, 2^(r-1))
end
```

The number of comparisons required by this method, including binomial tree creation,

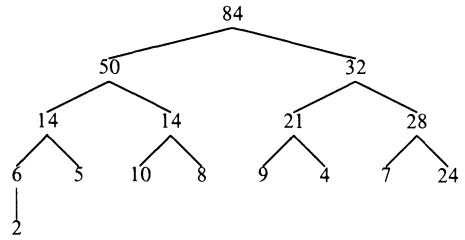
Procedure Convert (T, 2^r)
begin
  Convert (subtree of root of order 2^{r-1}, 2^{r-1});
  This leaves an "extra element" on the bottom level
  of this heap, make it a child of the root of T;



We now have 1 binomial tree of order 2^i ($i = 1,...,r-2$) and 2
singleton nodes all hanging from the root;
Construct a binomial tree, S, of order 2^{r-1} from these;



Convert(S, 2^{r-1})



end.

FIG. 4. *Illustration of heap the construction algorithm with the approach construct a binomial tree of* 2^r *nodes and convert it to a heap.*

can be shown to be

$$T(2^k) = 2T(2^{k-1}) + k.$$

A binomial tree of order 2 or 4 is a heap; hence the recursive call on these small binomial trees can be omitted. A more important observation is that a binomial tree on 8 nodes can be converted to a heap in 1 comparison (rather than 2). Jumping out of the recursion in the above algorithm on trees of size 8 and using $T(8) = 8$ as a basis yields the solution.

$$T(2) = 1, \quad T(4) = 3 \quad \text{and}$$
$$T(2^k) = \tfrac{13}{8}2^k - k - 2 \quad (\text{for } k \geqq 3). \qquad \square$$

We are inclined to believe our technique is optimal in the worst case when n is of the form 2^k or $2^k - 1$ and within a lower order term otherwise. By a careful

examination of structures on 3 and 4 elements we can show that the method is optimal for 7-heaps.

THEOREM 6. 8 *comparisons are necessary and sufficient, in the worst case, to form a 7-heap.*

However, a 7-heap can be constructed using $7\frac{2}{7}$ comparisons on the average based on the binomial tree "building blocks". This yields an improvement in the average behavior of our algorithm:

THEOREM 7. $1\frac{15}{28}n = 1.5357...$ *n comparisons are sufficient, on the average, to construct a heap on n nodes.*

REFERENCES

[1] R. W. FLOYD, *Algorithm* 245, *Treesort* 3, Comm. ACM, 7 (1964), p. 701.
[2] G. H. GONNET, *A Handbook of Algorithms and Data Structures*, Addison-Wesley, Reading, MA, 1984.
[3] D. B. JOHNSON, *A priority queue in which initialization and queue operations take* $O(\log \log D)$ *Time*, Math. Systems Theory, 15 (1982), pp. 295-309.
[4] D. E. KNUTH, *The Art of Computer Programming, Vol.* 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
[5] J. I. MUNRO AND H. SUWANDA, *Implicit data structures for fast search and update*, J. Comput. System Sci., 21 (1980), pp. 236-250.
[6] P. VAN EMDE BOAS, R. KAAS AND E. ZILJLSTRA, *Design and implementation of an efficient priority queue*, Math. Systems Theory, 10 (1977), pp. 99-127.
[7] J. VUILLEMIN, *A data structure for manipulating priority queues*, Comm. ACM, 21, 4 (1978), pp. 309-314.
[8] J. W. J. WILLIAMS, *Algorithm* 232, *Heapsort*, Comm. ACM, 7 (1964), pp. 347-348.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
007

ON APPROXIMATIONS AND INCIDENCE IN CYLINDRICAL ALGEBRAIC DECOMPOSITIONS*

DAVID PRILL†

Abstract. Let $P \subset \mathbb{Z}[x_1, \cdots, x_r]$ be a finite set. This paper describes and analyzes a variant of the algorithm of Collins and others for decomposing $\mathbb{R}^r$ into semi-algebraic cells so that the value of each $f \in P$ has constant sign (positive, negative, or zero) on the points of each cell. The version here has several advantages:

1. The boundary of each cell is a disjoint union of lower-dimensional cells. For each bounded cell α the pair $(\bar{\alpha}, \alpha)$ is homeomorphic to a closed ball and its interior.
2. An algorithm is presented which for fixed r computes incidence of cells in polynomial time.
3. A priori estimates of the accuracy of approximations of roots of polynomials required in order to determine the combinatorial structure of the cell complex are given. This avoids computation in algebraic number fields.

Key words. cylindrical algebraic decomposition, Tarski decision procedure, semi-algebraic set, cell incidence, polynomial time, regular cell complex

AMS(MOS) subject classification. 52

1. Introduction. The results of this paper were obtained independent of the work of Schwartz and Sharir [18] in an attempt to apply the cell decomposition of Collins [6] to the "piano movers" problem.[1] The relation of the cell decomposition to that problem is explained well in [18]. We shall concentrate on the cell decomposition. Its effective use for computation of topological invariants entails knowing cell incidences. We show that cell incidences can be computed in polynomial time provided that the number of variables is fixed. The desirability of such a result is mentioned by Schwartz and Sharir [18]. It implies that the homology groups of an algebraic variety or closed semi-algebraic set defined over the integers can be obtained in time polynomial in the number of defining polynomials, their degrees, and the maximum binary length of coefficients. Our time estimates are in the spirit of those of Collins [6]. We posit that two d digit integers can be added in time d and multiplied or divided with remainder in time d^2. Like Collins, we must keep track of growth of coefficients and degrees of polynomials and other parameters during the iterative processes. To compute cell incidence we do not use fractional power series like Schwartz and Sharir. Instead we find roots of certain polynomials. Several steps are left in a crude form to simplify the exposition. Routine improvements in certain estimates would yield a more efficient algorithm of the same type. Whether it could be fast enough for many practical uses remains to be seen.

2. Summary of the algorithm. This section outlines the construction of the cell decomposition. It describes the properties of closures of cells and how incidence of cells will be computed. It defines some of the quantities used in analysis of the method. More details appear in later sections.

In this paper we shall produce an abstract complex, called a set of approximate sample points, which is isomorphic to Collins' cell decomposition. This is described briefly later in this section. We shall show how to compute this complex by a sequence

* Received by the editors January 7, 1985, and in revised form August 15, 1985.

† Mathematics Department, University of Rochester, Rochester, New York 14627.

[1] The author thanks J. H. B. Kemperman for acquainting him with that problem. He thanks C. Silva for drawing Collins' work [6] to his attention.

of rational approximations to roots of polynomials. Unlike other approaches, no computations in algebraic number fields are needed.

The geometry of the cell decomposition is substantially that of Koopman and Brown [14]. They even call the process algorithmic. Nevertheless, the algorithmic aspects were first studied by Collins [6].

Let $P=\{f_1, \cdots, f_s\}$ be a nonempty finite subset of the set $\mathbb{Z}[x_1, \cdots, x_r]$ of polynomials with integer coefficients in r variables. Suppose each member of P has total degree at most n and coefficients the sum of whose magnitudes has binary length at most L. We refer to (r, s, n, L) as the parameters of P. Call a cell decomposition of $\mathbb{R}^r$ appropriate to P if each $f \in P$ has constant sign (positive, negative, or zero) on each cell. Collins constructs such a cell decomposition by induction on r. To P he associates a finite set $P' \subset \mathbb{Z}[x_1, \cdots, x_{r-1}]$ and a finite cell decomposition K' appropriate to it in such a way that cells of a decomposition K appropriate to P stack up over each cell of K' like layers of a cake. Let $(r-1, s', n', L')$ be the parameters for P'. Then $(s'+n'+L')$ is bounded by $(s+n+L)^3$. Assume by induction that a cell decomposition of $\mathbb{R}^{r-1}$ appropriate to P' can be constructed within time polynomial in $(s'+n'+L')$ with exponents exponential in $(r-1)$. Then one has the same type of bound in the parameters for P. This remains true when the additional steps needed for computing incidence are adjoined. One main result of this paper is a method for computing incidence in polynomial time. The construction of K from K' and determination of incidence of cells of K proceeds in polynomial time. The construction when $r=1$ resembles the inductive step. The construction for $r=0$ is immediate.

Before using Collins' algorithm, we make a linear change of coordinates so that for each $f \in P$ the total degree of f equals the degree of f as a polynomial in x_r. This idea presumably could be traced back to Weierstrass (cf. Koopman and Brown, [14]). As we shall see, it guarantees that the bounded cells form a regular cell complex and that the boundary of any cell is a finite union of lower-dimensional cells. (By a k-cell, we mean a set homeomorphic to an open ball in $\mathbb{R}^k$. A 0-cell is a point.) This is not always true without such a coordinate change. In [3] no such coordinate change is made. Instead of incidence the authors study adjacency, i.e., when the union of two cells is connected.

Let β be a cell in the cell decomposition K' of $\mathbb{R}^{r-1}$ appropriate to P'. It has the following property (Collins [6, pp. 139–141]): For each $f \in P$ and $x \in \beta$, the number of distinct complex roots of $g(z)=f(x, z)$ and their multiplicities are constants depending on f and β but not on x. It follows that all real roots at x of g are values at x of continuous real root functions, i.e. functions $\psi: \beta \to \mathbb{R}$ such that the multiplicity of $\psi(x)$ as a root of g is independent of x. Moreover, the construction is such that whenever s and t are such real root functions for not necessarily distinct elements of P, then either $s=t$ or $s(x) \neq t(x)$ for all x in β.

Recall the construction of cells over β. There are two types of cells over β. Let $r_1, \cdots, r_k$ be the real root functions of β, arranged so that $r_1 < r_2 < \cdots < r_k$ at each point of β. The horizontal cell $H_j(\beta)$ is the graph of $r_j, j=1, \cdots, k$. The vertical cell $V_j(\beta)$ is defined by

$$V_j(\beta)=\{(x, y) \in \beta \times \mathbb{R} | r_j(x) < y < r_{j+1}(x)\}.$$

We define $V_0(\beta)$ and $V_k(\beta)$ as above by setting $r_0=-\infty$; $r_{k+1}=+\infty$. (In [8] horizontal and vertical cells are called Zellen erster bzw. zweiter Art. In [3] and the paper preceding it they are called sections and sectors.)

By induction, we may suppose that in fewer than r variables all the root functions which are similarly defined extend continuously to the closure of the cell. It follows

by induction that for each point $z \in \bar{\beta}$ there exists a basis of neighborhoods N in $\bar{\beta}$ such that $N \cap \beta$ is connected and open in β. A classical argument (cf. Giesecke [8, p. 184]) then shows each root function extends to a continuous function on $\bar{\beta}$. This idea occurs also in Koopman and Brown [14], Hironaka [12], Schwartz and Sharir [18].

The set of cells of K is the union of the sets of horizontal and vertical cells over the various cells of K'.

Let α be a cell of K' contained in $\bar{\beta} - \beta$. By induction on dimension, $\bar{\beta} - \beta$ is the union of such cells. Let $\rho_1, \cdots, \rho_q$ be the (real) root functions for α arranged in increasing order. Each root functions r_j for β extends to $\bar{\beta}$. Then $r_j | \alpha = \rho_i$ for some $i = i(j)$ depending on j. Then as α runs through the cells of $\bar{\beta} - \beta$, $H_{i(j)}(\alpha)$ runs through the cells of $\overline{H_j(\beta)} - H_j(\beta)$. For vertical cells we have that $\bar{V}_j(\beta) - V_j(\beta)$ is the union of $H_j(\beta)$, $H_{j+1}(\beta)$ and certain cells over the cells α of $\bar{\beta} - \beta$. The portion of $\bar{V}_j(\beta) - V_j(\beta)$ over α consists of the horizontal cells $H_i(\alpha)$ for which $i(j) \leqq i \leqq i(j+1)$ and the vertical cells $V_i(\alpha)$ for which $i(j) \leqq i < i(j+1)$.

We now describe how incidence will be computed. Let α and $\beta \in K'$ be as in the previous paragraph. Let $a \in \alpha$ be the point which Collins calls the sample point. Then $(a, \rho_i(a))$ is the sample point of $H_i(\alpha)$ for $1 \leqq i \leqq k$. The sample point of $V_i(\alpha)$ is $(a, \frac{1}{2}(\rho_i(a) + \rho_{i+1}(a)))$, $0 < i < k$. The sample point of $V_0(\alpha)$ is $(a, \rho_1(a) - 1)$. For $V_k(\alpha)$ the sample point is $(a, \rho_k(a) + 1)$. Let $c > 0$ be such that $16c$ is less than the minimum distance between sample points of distinct cells over α. Let $b^* \in \beta$ be very near a. For each j

$$|r_j(b^*) - \rho_{i(j)}(a)| < c;$$

while for $\nu \neq i(j)$

$$|r_j(b^*) - \rho_\nu(a)| > 15c.$$

In the next section we shall determine how small $\|b^* - a\|$ need be, construct such a b^*, and approximate $r_j(b^*)$ and $\rho_\nu(a)$ within c for all ν and j. Call the approximations $\bar{r}_j$ and $\bar{\rho}_\nu$. Then $|\bar{r}_j - \bar{\rho}_\nu| < 3c$ if $\nu = i(j)$ and $|\bar{r}_j - \bar{\rho}_\nu| > 13c$ otherwise. Thus, $i(j)$ may be computed for each j and this determines incidence of cells.

Maintain the notation above. A number c is obtained by techniques of number theory that use bounds for the degrees and coefficients of the polynomials in P. The value at a of each root function is a root of a polynomial, say $f(a, z)$. We can replace a by an approximation without much change in its roots. To do so we use a quantitative form of the continuity of the roots of a polynomial as "functions" of its coefficients. As b^* approaches a, several root functions may approach the same limit. Thus, if b^* were extremely near a then the separation of the numbers $r_j(b^*)$ for various j might be very small. If so, it would be difficult to compute the numbers $\bar{r}_j$ sufficiently well to use them as substitutes for the numbers $r_j(b^*)$. We choose b^* with some care to avoid this. We call b^* an incidence point.

All steps must be submitted to the iterative process associated to repeated projection to a Euclidean space of one less dimension. This leads to tolerance limits for the approximations of each of the components of a sample point. Tolerance limits for earlier components are much more severe. Tolerance limits are also imposed on incidence points. (They are more severe than for sample points.)

Finally, we are able to describe what is for us the basic object to compute. We call it a set of approximate sample points. (The complete definition appears in § 3.6.) It consists of an approximation to each sample point and considerable additional structure. There are incidence relations which correspond to those in the Collins

decomposition. There is an approximation to each incidence point. These approximations are specified so that we can work upward by dimension through the projections to lower dimensional Euclidean spaces. The data on hand at each stage suffice to compute incidence and other data in one or more dimension. No direct reference to the Collins decomposition is necessary.

To conclude this brief sketch, let us discuss the topology of bounded cells. Suppose $\beta \in K'$ is bounded. By induction on dimension the pair $(\bar{\beta}, \beta)$ is homeomorphic to a closed ball of some Euclidean space and its interior. For each horizontal cell $H_i(\beta)$ over β, $(\overline{H_i(\beta)}, H_i(\beta))$ is homeomorphic to $(\bar{\beta}, \beta)$. For $0<j<k(\beta)$ the pair $(\overline{V_j(\beta)}, V_j(\beta))$ is homeomorphic to $(\bar{B}, B)$, where B is a closed ball of dimension one more than the dimension of β. This is proved by Giesecke [8, p. 179], as a lemma, and Koopman and Brown [14, p. 235].

3. The cell decompositions. This section studies the growth of the set of parameters associated to a finite set of polynomials under the iterative process in Collins' cell decomposition. This will be needed to know how accurately certain computations used in finding incidences must be carried out. It will also be used in bounding the time the algorithm takes. The time bounds obtained will depend more on the growth of the parameters than on the efficiency of subroutines. This differs from the approach of Schwartz and Sharir which mainly counts the number of operations.

3.1. The defining equations. We now describe the finite sets of polynomials with which one must deal during the iterative process. Such sets are used by Collins [6], Arnon [2], and Schwartz and Sharir [18]. Let $P \subset \mathbb{Z}[x_1, \ldots, x_r]$ be a finite set of s polynomials each of degree at most n. Collins uses certain polynomials called principal subresultants which are determinants of minors of the Sylvester matrix, the usual matrix whose determinant is the resultant of two polynomials. Let $\mathrm{psc}_k(f, g)$ denote the kth principal subresultant of $f, g \in \mathbb{Z}[x_1, \cdots, x_r]$, regarded as polynomials in x_r. (See [6].) For $f \in \mathbb{Z}[x_1, \cdots, x_r]$ define $f' = \partial f / \partial x_r$. Define

$$A = \{\mathrm{psc}_k(f, g) | f, g \in P;\ 0 \leqq k < \min(\deg f, \deg g)\},$$

$$B = \{\mathrm{psc}_k(f, f') | f \in P;\ 0 \leqq k < \deg(f')\},$$

$$P' = A \cup B.$$

Define $P^{(j)}$ inductively by $P^{(0)} = P$ and $P^{(j)} = (P^{(j-1)})'$.

Let $f \in P$. For each fixed point x' in $\mathbb{R}^{r-1}$ regard f as a polynomial in x_r. The degree of this polynomial is a function of x'. In [6] additional polynomials are included in P' in order to assure that such degree functions are constant on each cell of the decomposition of $\mathbb{R}^{r-1}$ and that the subresultants are computed for each fixed x' using polynomials of the correct degrees in x_r. We shall first choose good coordinates so that the leading coefficients of members of P, as polynomials in x_r, are nonzero constants. Hence we do not need to make P' as large as in [6].

We now discuss the construction of good coordinates. Let $P \subset \mathbb{Z}[x_1, \cdots, x_r]$ consist of polynomials of degree at most n. Suppose P contains at most s polynomials different from zero. If $ns > 1$, then there exists $v \in \mathbb{Z}^r$ with $v_r \neq 0$ and each component bounded by $ns-1$ such that for all $f \in P$ the degree of f equals the degree of $f(x, v)$. Let $\varphi(x_1, \cdots, x_r) = (x_1, \cdots, x_{r-1}, 0) + x_r v$. The sum of the absolute values of the coefficients of $x_1, \cdots, x_r$ in each component of φ is, of course, bounded by ns and for each $f \in P$ the polynomials $(f \circ \varphi)(0, \cdots, 0, x_r)$ and $f \circ \varphi$ have equal degrees. If $ns = 1$, then a linear map $\varphi : \mathbb{R}^r \to \mathbb{R}^r$ with the property of the last sentence can also be obtained by (perhaps) permuting two coordinates. We call a map with this property a regularizing

coordinate change. For any linear map $\varphi : \mathbb{R}^r \to \mathbb{R}^r$ which has an integer matrix we define

$$P \circ \varphi = \{f \circ \varphi \mid f \in P\}.$$

Define $Q(j) \subset \mathbb{Z}[x_1, \cdots, x_{r-j}]$ inductively

$$Q(0) = P,$$

$$Q(j+1) = (Q(j) \circ \psi_j)' \quad \text{for } 0 \leqq j < r-1,$$

where ψ_j is a regularizing coordinate change for $Q(j)$. Regard $Q(j)$ as a subset of $\mathbb{Z}[x_1, \cdots, x_r]$. Define

$$\varphi_j(x_1, \cdots, x_r) = (\psi_j(x_1, \cdots, x_{r-j}), x_{r-j+1}, \cdots, x_r),$$

$$P(j) = Q(j) \circ \varphi_j \circ \cdots \circ \varphi_{r-2}.$$

Then

$$P(0) = P \circ \varphi_0 \circ \cdots \circ \varphi_{r-2}.$$

Note

$$P(j) = (P(0))^{(j)} \quad \text{for } 0 \leqq j < r,$$

because $\varphi_j, \cdots, \varphi_{r-2}$ do not change the last j coordinates and the subresultants involved depend upon the coefficients of polynomials as functions of the last j variables. Then $\varphi_0 \circ \cdots \circ \varphi_{r-2}$ (or its inverse, depending on one's point-of-view) is a change to a good coordinate system. Henceforth, a Collins decomposition will mean a change to a good coordinate system and application of the process of Collins [6] in that coordinate system.

We now investigate the parameters of the sets $P(j)$ and $Q(j)$.

Let n_j be a bound for the degrees of the polynomials in $P(j)$ or $Q(j)$. Then $n_{j+1} \leqq n_j^2$. Thus, $n_j \leqq n^{(2^j)}$. For convenience we take $n_j = n^{(2^j)}$.

Let s_j be a bound for the number of nonzero polynomials in $P(j)$ or $Q(j)$. Then

$$s_{j+1} \leqq \binom{s_j}{2} n_j + s_j(n_j - 1) \leqq n_j s_j^2.$$

Thus,

$$s_j \leqq n^{j \cdot 2^{j-1}} s^{2^j}.$$

For convenience we define s_j so that equality holds.

Let f be a polynomial in r variables of degree at most n and with coefficients bounded by M. If $f = \sum c_\alpha x^\alpha$ is the representation of f as a sum of monomials in the r coordinates, then define $\|f\| = \sum |c_\alpha|$. Then

$$\|f\| = \sum |c_\alpha| \leqq M \binom{n+r}{r} \leqq M(n+1)^r.$$

Henceforth, we shall denote $\log_2$ *by* log.

Let L_j be a bound for $\{\log \|f\| \mid f \in Q(j); f \neq 0\}$. We may take L_0 so $1 \leqq L_0 \leqq r \log(n+1) + \log M$. For $f \in Q(j)$,

$$\|f \circ \psi_j\| \leqq (s_j n_j)^{n_j} \|f\|.$$

Let $t = r - j$.

$$\left\| \frac{\partial}{\partial x_t} (f \circ \psi_j) \right\| \leqq n_j (s_j n_j)^{n_j} \|f\|.$$

The nonzero entries of each row of the Sylvester matrix of two polynomials are the coefficients of one of the polynomials. It follows that for $f, g \in Q(j)$ by Hadamard's inequality

$$\|\mathrm{psc}_k(f \circ \psi_j, g \circ \psi_j)\| \leqq (\|f \circ \psi_j\| \cdot \|g \circ \psi_j\|)^{n_j} \leqq s_j^{2n_j^2} n_j^{2n_j^2} 2^{2L_j n_j}.$$

The same bound holds for $\|\mathrm{psc}_k(f \circ \psi_j, (\partial/x_t)(f \circ \psi_j))\|$. Thus, in order that L_{j+1} be a bound of the required type it suffices that

$$L_{j+1} \geqq 2L_j n_j + 2n_{j+1}(\log s_j + \log n_j).$$

Let $\lambda_j = 2^{-j} L_j / n_j$. It suffices that

$$\lambda_{j+1} \geqq \lambda_j + (j+1)\log n + \log s.$$

Let

$$\lambda_j = \lambda_0 + (j^2+3j)\log n + j \log s,$$

$$\lambda_0 = L_0,$$

$$L_j = 2^j n_j \lambda_j \quad \text{for each } j.$$

For $f \in P(j)$, and $0 \leqq j \leqq r-2$, $f = g \circ \psi_j \circ \cdots \circ \psi_{r-2}$ for some $g \in Q(j)$. Then $\|f\| \leqq (s_j n_j)^{n_j} \cdots (s_{r-2} n_{r-2})^{n_j} \|g\|$. Let M_j be a bound for $\{\log \|f\| \mid f \neq 0, f \in P(j)\}$. Then

$$M_j \leqq 2n_j \log(s_{r-2} n_{r-2}) + L_j \leqq L_j + L_{r-1} - 2n_{r-2} L_{r-2} \leqq L_{r-1} - 1.$$

Also, $P(r-1) = Q(r-1)$ so we may take $M_{r-1} = L_{r-1}$. We have for $0 \leqq j \leqq r-1$:

$$1 + M_j \leqq L, \quad \text{where } L = L_{r-1} + 1.$$

3.2. Bounds for roots.

LEMMA. *Let f be a polynomial in one variable. Suppose the coefficient of the highest degree term of f is a nonzero integer. Then each root of f is bounded by the maximum of* 1 *and* $\|f\| - 1$.

Proof. Easy.

We shall determine incidence of the cells in a decomposition from considerations involving distance between points in certain sets $S(j, P, k)$. We now define these sets and establish some of their properties.

Let $k = (k_0, \cdots, k_{r-1})$ be a vector of nonnegative integers. Define $S(j\,P, k)$ inductively as j decreases from $r-1$ to 0 by

$$S(r-1, P, k) = \{a \in \mathbb{C} \mid a = 0 \text{ or } \exists f, g \in P(r-1) \text{ with roots } a_1 \text{ and } a_2 \text{ and } a - \tfrac{1}{2}(a_1 + a_2) \text{ has value } 0, \pm 1, \text{ or } \pm 2^{-k_{r-1}}\}.$$

$$S(j, P, k) = \{(b, a) \in S(j+1, P, k) \times \mathbb{C} \mid a = 0 \text{ or } \exists f, g \in P(j) \text{ such that } f(b, x) \text{ and } g(b, x) \text{ have roots } a_1 \text{ and } a_2 \text{ and } a - \tfrac{1}{2}(a_1 + a_2) \text{ has value } 0, \pm 1, \text{ or } \pm 2^{-k_j}\}.$$

In the above definitions, the numbers a_1 and a_2 need not be distinct. When k and P are fixed, we shall denote $S(j, P, k)$ by $S(j)$. Later we shall determine incidence of the cells in a decomposition from considerations involving distance between points in $S(j, P, k)$ for an appropriate k. We now establish some basic properties of these sets.

Let R_j be the maximum modulus of components of points in $S(j)$. We shall bound R_j.

First note that by the lemma

$$1+\log(R_{r-1}) \leqq L.$$

For $j<r-1$, let $a \in S(j+1)$ and suppose each component of a is bounded by R_{j+1}, $f \in P(j)$ and z is a root of f. Suppose f has degree σ and $f=\sum c_\alpha x^\alpha$. Let $g(t)=\sum c_\alpha b^\alpha R_{j+1}^{|\alpha|-\sigma}$, where $b=(a/R_{j+1}, t)$. Then z/R_{j+1} is a root of g and $\|g\| \leqq \|f\|$, provided $R_{j+1} \geqq 1$. By the lemma,

$$|z| \leqq R_{j+1} \max(1, \|g\|-1).$$

Then $|z|+1 \leqq R_{j+1}2^{M_j}$ so

$$\log R_j \geqq \log R_{j+1} + M_j.$$

For $0 \leqq j \leqq r-1$ we have

$$1+\log R_j \leqq (r-j)L.$$

3.3. Separation of roots. Let $\bar{k}$ be the maximum of the components of k. We want an integer $\delta_j(P, k)$ so that for each $a \in S(j, P, k)$ there exists a positive integer t with $\log t \leqq \delta_j$ such that each component of $2^{\bar{k}}ta$ is an algebraic integer. Such integers δ_j will be constructed inductively as j decreases from $(r-1)$ to 0.

To find δ_{r-1}, let $f \in P(r-1)$;

$$f=\sum_{\nu=0}^{\sigma} c_\nu x^\nu$$

where the coefficients are integers and $c_\sigma \neq 0$. If z is a root of f then $c_\sigma z$ is easily seen to be an algebraic integer. Then we take

$$\delta_{r-1}=2L.$$

Suppose $b \in S(j+1)$, $f \in P(j)$ and $f(b, z)=0$. Then

$$f(b, z)=\sum_{\nu=0}^{\sigma} c_\nu(b) z^\nu,$$

where each c_ν is a polynomial in b with integer coefficients and c_σ is a nonzero rational integer. Pick t with $\log t < \delta_{j+1}$ so that with $q=2^{\bar{k}}t$ each component of qb is an algebraic integer. Then $w=qc_\sigma z$ is an algebraic integer, because

$$\sum_{\nu=0}^{\sigma} q^{\sigma-\nu} c_\nu(b) c_\sigma^{(\sigma-1-\nu)} (w)^\nu = 0.$$

For two roots of $f(b, x)$ and $g(b, x)$, where $f, g \in P(j)$ we may use the same factor q, but may have different leading coefficients. It suffices then that

$$\delta j \geqq 2M_j + 1 + \delta_{j+1}$$

or that

$$\delta_j \geqq 2L + \delta_{j+1}.$$

Take

$$\delta_j = 2(r-j)L.$$

We need a bound N_j for the degree over the rationals of the field extension obtained by adjoining the components of a vector in $S(j)$. Clearly

$$N_{r-1} \leqq (n_{r-1})^2,$$
$$N_j \leqq n_j^2 N_{j+1} \quad \text{for } j+1 < r.$$

Therefore

$$N_j \leqq n_j^2 n_{j+1}^2 \cdots n_{r-1}^2 \leqq n^{2^{r+1}} = n_{r+1} \quad \text{for } j \leqq r-1.$$

We need a lower bound for $|x-y|$ when $x \neq y$ and for some b the points (b, x) and (b, y) belong to $S(j, P, k)$. There exists a positive integer t with $\log t \leqq 2\delta_j$ for which $t2^{\bar{k}}(x-y)$ is an algebraic integer of degree is bounded by $N_{j+1}n_j^4 \leqq n_{r+1}$. Its conjugates are bounded by $2t2^{\bar{k}}R_j$. Its norm is the product of its conjugates and is a rational integer. Therefore,

$$2^{\bar{k}}t|(x-y)|(2t2^{\bar{k}}R_j)^{n_{r+1}-1} \geqq 1.$$

Then

$$\begin{aligned}\log|x-y| &\geqq -n_{r+1}(1+\bar{k}+\log R_j+2\delta_j)\\ &\geqq -n_{r+1}(\bar{k}+5rL)\\ &\geqq -C+6,\end{aligned}$$

where $C = n_{r+1}(\bar{k}+5rL)+6$.

3.4. Continuity of roots.

LEMMA 3.4.1. *Let $r>1$. Let C be as above. Let $\mu_0 > C$ be an integer. Then one can compute a vector $\mu = (\mu_0, \cdots, \mu_{r-1})$ of integers with $\mu_0 < \mu_1 < \cdots < \mu_{r-1}$ such that for all j with $1 \leqq j \leqq r-1$ the following holds*:

Let $b \in S(j, P, k) \subset \mathbb{C}^{r-j}$ and $a \in \mathbb{C}^{r-j}$. Suppose that $\log|a_i - b_i| \leqq -\mu_{r-i}$ for $0 < i \leqq r-j$. Let $\Sigma = \{z \in \mathbb{C} | \exists g \in P(j-1)$ and $g(b, z) = 0\}$. Then for every $f \in P(j-1)$ and each root w of $f(a, z)$ there is exactly one z in Σ for which $\log|z-w| < -1-\mu_{j-1}$. This point is a root of $f(b, z)$. For all other $p \in \Sigma$, $\log|p-w| > 5-C$.

Proof. We shall use Rouché's theorem. (We just need a special case, viz: Let g and h be analytic in a disc D and $|h-g| < |g|$ at all points of the boundary of D. Then h and g have the same number of zeros in D, counting multiplicities.) Let f have degree $\sigma \leqq n_{j-1}$. Write f as a sum of monomials

$$f(a, z) = \Sigma c_{\alpha,\beta} a^\alpha z^\beta$$

where the sum of the weights of α and β is at most σ. For each α

$$|a^\alpha - b^\alpha| \leqq \sum_{k=j}^{r-1} 2^{-\mu_k}(R_0+1)^{\|\alpha\|} \leqq 2 \cdot 2^{-\mu_j}(R_0+1)^{\|\alpha\|}.$$

Then for $|z| \leqq R_{j-1}$:

$$|f(a, z) - f(b, z)| \leqq 2^{M_{j-1}} 2 \cdot 2^{-\mu_j}(R_0+1)^\sigma$$

so

$$\begin{aligned}\log|f(a, z) - f(b, z)| &\leqq L+1-\mu_j+n_{r+1}(rL+1)\\ &< -n_{j-1}-2-(\mu_j/2).\end{aligned}$$

Let $f(b, \zeta) = 0$; $\mu_{j-1} \geqq C$. Since the roots of $f(b, z)$ are separated by at least 2^{-C+6} and $f(b, z)$ has leading coefficient an integer, when $-\log|z-\zeta| = 1+\mu_{j-1}$:

$$\log|f(b, z) \geqq -n_{j-1}(1+\mu_{j-1}).$$

By Rouché's theorem μ_{j-1} will have the desired property if

$$-\tfrac{1}{2}\mu_j \leqq -n_{j-1}\mu_{j-1}, \quad \text{i.e.,} \quad \mu_j \geqq 2n_{j-1}\mu_{j-1}.$$

For $j > 0$ take

$$\mu_j = 2^j n_j \mu_0.$$

Then $\mu_j = 2n_{j-1}\mu_{j-1}$ for $j > 0$. If $\zeta, p \in \Sigma$ and $p \neq \zeta$ then for every z with $|z-\zeta| \leqq 2^{-\mu_{j-1}-1}$

$$|p-z| \geqq |p-\zeta| - |\zeta - z| \geqq 2^{6-C} - 2^{-\mu_{j-1}-1} \geqq 2^{5-C}.$$

DEFINITION 3.4.2. Let $k_j = 2^j n_j A$ for $0 \leqq j \leqq r-1$, where $A > n_{r+1}(5rL)+6$ is an integer.

Remark 3.4.3. We shall later approximate certain real points of $S(j, P, k)$. This will be iterated for decreasing values of j. When a approximates a point $b \in \mathbb{R}^{r-j}$ of $S(j, P, k)$ as in Lemma 3.4.1 and $f \in P(j-1)$ the real roots of $f(b, z)$ may not be approximated by *real* roots of $f(a, z)$. We can, nevertheless, approximate real roots of $f(b, z)$ by finding real roots of polynomials related to $f(a, z)$. For the duration of this remark, let $\nu = n_{j-1}$; $m = 1+\mu_{j-1}$.

Suppose that in the lemma z and ζ are real, $f(b, \zeta) = 0$, and $z - \zeta = \pm 2^{-m}$. We shall use implicitly the estimates in the proof of the lemma. Then $|f(a, z)| \geqq \frac{3}{4}2^{-\nu m}$ and $|f(a, \zeta)| \leqq \frac{1}{4}2^{-\nu m}$. It follows that one or both of the equations

$$f(a, x) = \pm 2^{-\nu m - 1}$$

has a real root within 2^{-m} of ζ. On the other hand, every real solution y of such an equation lies within 2^{-m} of some root of $f(a, \lambda)$. Then y lies within 2^{1-m} of a root of $f(b, \lambda)$ so $|y| \leqq R_{j-1}$. Then $|f(b, y)| \leqq \frac{3}{4}2^{-\nu m}$ so y is at distance less than 2^{-m} from a root of $f(b, \lambda)$. Since all nonreal roots of $f(b, \lambda)$ have their imaginary parts bounded away from zero by 2^{-C+5}, y lies within 2^{-m} of some *real* root of $f(b, \lambda)$.

Let $\Sigma = \{z \in \mathbb{C} | \exists f \in P(j-1)$ with $f(b, z) = 0\}$ and $T = \{y \in \mathbb{R} | \exists f \in P(j-1)$ with $f(a, y) = \pm\frac{1}{2}2^{-\nu m}\}$. Each element of T is within $2^{-1-\mu_{j-1}}$ of some element of $\Sigma \cap \mathbb{R}$. Elements of Σ have distance at least 2^{-C+6} apart. Introduce an equivalence on T by calling points of T equivalent if they are within 2^{-C} of each other. There is a one-to-one and order-preserving correspondence between elements of $\Sigma \cap \mathbb{R}$ and equivalence classes on T. Each element of T is within 2^{-m} of the corresponding element of $\Sigma \cap \mathbb{R}$. Inequivalent elements of T have distance at least 2^{5-C} apart.

3.4.4. *Summary.* Here is a summary of the notation and estimates that we shall need in later sections. Parameters with the subscript j are for $j \leqq r-1$ usually related to projection onto $\mathbb{R}^{r-j}$.

r = dimension of Euclidean space.

P is a finite subset of $\mathbb{Z}[x_1, \cdots, x_r]$.

$P(j)$ is a finite subset of $\mathbb{Z}[x_1, \cdots, x_{r-j}]$, also regarded as a subset of $\mathbb{Z}[x_1, \cdots, x_r]$, constructed from P.

n is a bound for the total degree of each member of P.

s is a bound for the cardinality of P.

$n_j = n^{2^j}$; $s_j = n^{j\cdot 2^{j-1}} s^{2^j}$.

For each $j < r$ the quantity n_j is a bound for the total degree of members of $P(j)$ and s_j is a bound for the cardinality of $P(j)$.

$P(j)$ is constructed from a set $Q(j)$ by a coordinate change.

L_0 is a bound for $\{\log \|f\| \mid f \in P; f \neq 0\}$. (If M is a bound for the coefficients of each $f \in P$ we choose L_0 so $1 \leqq L_0 \leqq r \log (n+1) + \log M$.)

L_j is a bound for $\{\log \|f\| \mid f \in Q(j)\}$; $L_j = 2^j n_j (L_0 + (j^2+3j) \log n + j \log s)$; $L = 1 + L_{r-1}$.

M_0 is a bound for $\{\log \|f\| \mid f \neq 0, f \in P(0)\}$.

Define A, C, k, μ as below:

$$A = 5rLn_{r+1} + 7,$$

$$k_j = 2^j n_j A,$$

$$\bar{k} = k_{r-1},$$

$$C + 1 = A + \bar{k} n_{r+1},$$

$$\mu_0 = C + 1,$$

$$\mu_j = 2^j n_j \mu_0,$$

$$k = (k_0, \cdots, k_{r-1}), \qquad \mu = (\mu_0, \cdots, \mu_{r-1}).$$

$S(j, P, k)$ is a finite subset of $\mathbb{R}^{r-j}$ which contains points called sample points in [6] and another set of points called incidence points which we shall define in § 3.5. It also contains points whose components are algebraic conjugates over $\mathbb{Q}$ to the components of such points. The quantities 2^{-k_j} relate to continuity considerations for roots of members of $P(j)$ evaluated at points of $S(j+1, P, (0, \cdots, 0))$. R_0 is a bound for the components of members of $S(0, P, k)$, $1 + \log R_0 \leqq rL$.

3.5. Sample points and incidence. A cell decomposition of $\mathbb{R}$ is associated to $P(r-1)$ by taking all real zeros of members of $P(r-1)$ as 0-cells and the intervals into, which they divide the line as 1-cells. This cell decomposition has the virtue that each member of $P(r-1)$ has constant sign (positive, negative, or zero) on each cell. Collins constructs by induction on j a cell decomposition of $\mathbb{R}^j$ such that every element of $P(r-j)$ has constant sign on each cell. Having obtained such a cell decomposition of $\mathbb{R}^{j-1}$, the cell decomposition of $\mathbb{R}^j$ is obtained by using the root functions of $P(r-j)$ over each cell Δ to construct horizontal and vertical cells. If $r_1 < \cdots < r_l$, where $l = l(\Delta)$ depends on Δ, are the root functions, the horizontal cell $H_j(\Delta)$ is defined by

$$H_j(\Delta) = \{(x, y) \in \Delta \times \mathbb{R} \mid y = r_j(x)\}; \qquad 1 \leqq j \leqq l(\Delta).$$

The vertical cell $V_j(\Delta)$ is defined by

$$V_j(\Delta) = \{(x, y) \in \Delta \times \mathbb{R} \mid r_j(x) < y < r_{j+1}(x)\}, \qquad 0 \leqq j \leqq l(\Delta).$$

Here $r_0 = -\infty$; $r_{l+1} \equiv +\infty$.

For each cell Collins defines a sample point in the cell. If c is the sample point of Δ, then $(c, r_j(c))$ is the sample point of $H_j(\Delta)$. For $1 \leqq j \leqq l(\Delta) - 1$, the sample point of $V_j(\Delta)$ is $(c, (r_j(c) + r_{j+1}(c))/2)$. The sample point of $V_0(\Delta)$ is $(c, -1 + r_1(c))$. The sample point of $V_l(\Delta)$ is $(c, 1 + r_l(c))$. The sample points in the cell decomposition of $\mathbb{R}^{r-j}$ belong to $S(j, P(0, \cdots, 0))$ for each j.

We shall determine incidences in the cell decomposition of $\mathbb{R}^{r-j}$ for $j = 0, \cdots, r-1$. That is, we will find the pairs of cells α, β such that $\alpha \subset \bar{\beta}$ and $\dim \beta = 1 + \dim \alpha$. This

will use the results on the continuity and separation of roots from §§ 3.3 and 3.4. Now let k be as in Definition 3.4.2. Let α and β be incident cells in $\mathbb{R}^{r-j}$. Let a be the sample point of α. We shall construct a point $b = b(\alpha)$ in β such that $b \in S(j, P, k)$ and $\log|a_i - b_i| \leqq -k_{r-i}$ for $i = 1, \cdots, (r-j)$.

It is easy to determine incidence and define such points in the cell decomposition of $\mathbb{R}^1$. (Proceed as in Case 1, below.) Suppose it has been accomplished in $\mathbb{R}^{r-j}$ for some fixed j. We show how to get such points and determine incidence of cells in $\mathbb{R}^{r-j+1}$. Let α, β be cells in $\mathbb{R}^{r-j+1}$ such that $\alpha \subset \bar{\beta}$ and $\dim \beta = 1 + \dim \alpha$. There are three cases to consider.

Case 1. The cell α is horizontal and β is a vertical cell. This occurs precisely when there is a cell Δ in the decomposition of $\mathbb{R}^{r-j}$ and an integer i such that $\alpha = H_i(\Delta)$ and β is $V_i(\Delta)$ or $V_{i+1}(\Delta)$. Let (c, e) be the sample point of α. Then there is an interval I of length 2^{5-k_0} with e as an end point such that except for the point (c, e) the set $\{c\} \times I$ is contained in β. Choose $b(\alpha) = (c, d)$ where $d = a \pm 2^{k_{j-1}}$ and the sign is chosen so $b(\alpha) \in \beta$.

Case 2. Both α and β are horizontal cells. Let

$$\alpha = \{(x, y) \in \alpha' \times \mathbb{R} | y = r(x)\} \quad \text{and}$$

$$\beta = \{(x, y) \in \beta' \times \mathbb{R} | y = \rho(x)\};$$

α' and β' are cells in the decomposition of $\mathbb{R}^{r-j}$ and r and ρ are root functions. The condition $\alpha \subset \bar{\beta}$ is equivalent to the conditions:

1. $\alpha' \subset \bar{\beta}'$.
2. The extension of ρ to $\bar{\beta}'$ equals r on α'.

By induction we know whether $\alpha' \subset \bar{\beta}'$. Suppose that $\alpha' \subset \bar{\beta}'$ and $\dim \beta' = 1 + \dim \alpha'$. Let $a' \in \alpha'$ be the sample point and let $b' = b'(\alpha')$ be the point associated to the incidence of α' and β'.

Order the horizontal cells over β' in accord with the order at each point of values of the defining root functions. Let N be the set of values at b' of real root functions. Then $c \in N$ if and only if (b', c) is a point of a horizontal cell over β'. In this case $(b', c) \in S(j-1, P, k)$. There is an order-preserving bijection from N to the set of horizontal cells over β' so $\rho(b')$ determines β.

Similarly, let A be the set of values at a' of real root functions. Then there is an order-preserving bijection of A with the set of horizontal cells over α' so $r(a')$ and α' determine α. Since $A = \{c \in \mathbb{R} | \exists g \in P(j-1)$ with $g(a', c) = 0\}$ it may be determined by finding roots of polynomials. Likewise for N. Suppose α is specified by giving α' and the place of α in the ordering of horizontal cells over α'. Then $r(a')$ is known. Conversely, when A and $r(a')$ are known then the place of α is known. Similar remarks hold for β.

By induction there is a path $\gamma = [0, 1] \to \mathbb{R}^{r-j}$ with $\gamma(0) = \alpha'$, $\gamma(1) = \beta'$ and for $0 < t \leqq 1$, $\gamma(t)$ is in β' and in $\{x \in \mathbb{R}^{r-j} | \log|x_i - a_i| \leqq -k_{r-i}$ for $i = 1, \cdots, r-j\}$. Let $f \in P(j-1)$ be such that $f(x, \rho(x)) = 0$ for all $x \in \beta'$. For each t the distance from each root of $f(\gamma(t), z)$ to the nearest root of $f(a', z)$ is less than one-sixteenth the distance between distinct elements of A. This follows from Definition 3.4.2 by substituting into Lemma 3.4.1 k for μ, $(0, \cdots, 0)$ for k, a' for b, and $\gamma(t)$ for a.

If condition 2 holds then $|\rho(b') - r(a')| \leqq 2^{-k_{j-1}}$. If not, then $|\rho(b') - r(a')| \geqq 2^{6-C} - 2^{-k_{j-1}} \geqq 2^{5-k_0}$. This determines whether α and β are incident. If they are, define $b(\alpha) = (b', \rho(b'))$.

Case 3. The cells α and β are vertical cells over cells α' and β' with $\dim \beta' = 1 + \dim \alpha'$ and $\alpha' \subset \bar{\beta}'$. In Case 2 we saw how for each root function ρ_j over β' we can determine an integer $i = i(j)$ such that the restriction to α' of the extension to $\bar{\beta}'$ of ρ_j

equals the ith root function over α'. We have seen in § 2 how this allows us to determine whether $\alpha \subset \bar{\beta}$.

Let (a', c) be the sample point of α. Let $b' = b'(\alpha')$ be the point of β' determined inductively since α' and β' are incident, i.e., as above. Then $b = b(\alpha) = (b', c)$ is the desired sample point of β. We must show $b \in \beta$. Suppose the description of β as a vertical cell is

$$\beta = \{(x, y) \in \beta' \times \mathbb{R} | \rho(x) < y < \hat{\rho}(x)\}.$$

For covenience assume $\rho(x)$ and $\hat{\rho}(x)$ are finite. We must show $\rho(b') < c < \hat{\rho}(b')$. We show $(\hat{\rho}(b') - c)$ and $(c - \rho(b'))$ are $\geqq 2^{5-k_0}$.

Let r and $\hat{r}$ be the unique root functions on α' such that

$$|\rho(b') - r(a')| \leqq 2^{-k_{j-1}} \quad \text{and} \quad |\hat{\rho}(b') - \hat{r}(a')| \leqq 2^{-k_{j-1}}.$$

Note that r and $\hat{r}$ may not be consecutive root functions, but $r < \hat{r}$. Since $(a', c) \in \alpha$ and $\alpha \subset \bar{\beta}$, there is an integer i and root functions r_i and r_{i+1} such that

$$r \leqq r_i < r_{i+1} \leqq \hat{r} \quad \text{and} \quad c = (r_i(a') + r_{i+1}(a'))/2.$$

Then

$$\hat{\rho}(b') - c \geqq \hat{r}(a') - c - |\hat{\rho}(b') - \hat{r}(a')| \geqq 2^{6-k_0} - 2^{-k_0} \geqq 2^{5-k_0}.$$

Similarly, $c - \rho(b') \geqq 2^{5-k_0}$. If ρ or $\hat{\rho}$ is infinite, the argument is similar.

Schwartz and Sharir determine incidence by a fractional power series method. They show it determines incidence of r and $(r-1)$-cells in polynomial time. A paper by Arnon, Collins and McCallum [3] gives algorithms for incidence when $r = 2$.

3.6. Approximate computations. Let k, C, μ, etc. be chosen as in § 3.4.4. In § 3.5 we constructed a sample point in each cell. Further, for each pair of cells $\alpha, \beta \in \mathbb{R}^{r-j}$ such that $\alpha \subset \bar{\beta}$ and $\dim \beta = 1 + \dim \alpha$ we constructed a point $b = b(\alpha) \in \beta$ in $S(j, P, k)$ such that the point b and the sample point $a \in \alpha$ are close in the sense that $2^{k_{r-i}} |b_i - a_i| \leqq 1$ for $i = 1, \cdots, (r-j)$. Here we replace computation of such points by a process which approximates them. We show how the cells and their incidences may be recovered from just these approximations. Roughly speaking we will make "μ-close" approximations to all points, i.e. approximate the $(r-i)$th coordinate within $2^{-\mu_i}$ for each i.

We shall construct an abstract cell complex isomorphic to Collins' decomposition of $\mathbb{R}^{r-j}$ for $j = r, \cdots, 0$. This will be done by induction for decreasing values of j. For fixed j we will construct a finite set Σ' in $\mathbb{R}^{r-j}$. The points of Σ' will be in one-to-one correspondence with the cells of the Collins decomposition of $\mathbb{R}^{r-j}$ constructed above so we will refer to the points Σ' as cells. We shall define the notions of incidence and dimension of cells of Σ' and show they agree with these notions for the Collins decomposition. We now define a contraption which is such an abstract complex with such additional properties as are needed to carry out its construction by induction.

DEFINITION. Let $0 \leqq j \leqq r$. A set of approximate sample points in $\mathbb{R}^{r-j}$ is a finite set $\Sigma' \subset \mathbb{R}^{r-j}$ with the following properties:

1. To each $a \in \Sigma'$ is associated a nonnegative integer called its dimension. Each point of Σ' will be called a cell.

2. Each cell has dimension at most $(r-j)$. For each fixed integer l with $1 \leqq l \leqq (r-j)$ there is given an abstract relation called incidence between the set of l-cells and the set of $(l-1)$-cells.

3. If $a, b \in \Sigma'$ and b has dimension one more than a and b are incident, then a point $b(a) \in \mathbb{R}^{r-j}$ is defined.

4. For each $c=(c_1, \cdots, c_{r-j})$ in Σ' or of the form $b(a)$ as in 3 the number $4c_i 2^{\mu_{r-i}}$ is an integer for $i=1, \cdots, r-j$.

5. Let Σ be the set of sample points of cells in the Collins decomposition of $\mathbb{R}^{r-j}$. Give each point of Σ the dimension of the cell in which it lies. There exists a bijection $\Phi: \Sigma \to \Sigma'$ with multitudinous properties as follows:

A. Φ preserves dimensions.

B. For $c \in \Sigma$, the ith components of $\Phi(c)$ and c differ by at most $(\frac{3}{4})2^{-\mu_{r-i}}$ for $i=1, \cdots, r-j$.

C. Φ preserves incidences. More precisely, let α and β be cells in the Collins decomposition having consecutive dimensions with sample points a and b. Then α and β are incident, i.e. $\alpha \subset \bar{\beta}$, if and only if $\Phi(a)$ and $\Phi(b)$ are incident.

D. Let α and β be as in C. Suppose α and β are incident. Denote by $b(\alpha)$ the "incidence point" constructed in § 3.5. Let $a'=\Phi(a)$; $b'=\Phi(b)$. Let $b'(a')$ be the point associated to the incidence of a' and b' according to 3. Then the ith components of $b'(a')$ and the $b(\alpha)$ differ by at most $2^{-\mu_{r-i}}$ for $i=1, \cdots, r-j$.

THEOREM. *Let $0 \leqq j \leqq r$. Then a set of approximate sample points in $\mathbb{R}^{r-j}$ exists.*

Proof. The origin in $\mathbb{R}^0$ is a set of approximate sample points there. Let $0<j \leqq r$. Let $\Sigma' \subset \mathbb{R}^{r-j}$ be a set of approximate sample points in R^{r-j}. We shall use Σ' to construct a set $T' \subset \mathbb{R}^{r-j+1}$ of approximate sample points in $\mathbb{R}^{r-j+1}$. The projection of T' into $\mathbb{R}^{r-j}$ given by neglecting the last coordinate will have image Σ'. This will prove the theorem by induction.

We first construct a set of approximate sample points in $\mathbb{R}^1$. The method is in essence the inductive step for the general case. Let $\mathcal{S}=P(r-1)$, $k=k_{r-1}$, $\mu=\mu_{r-1}$.

Let Z be the union of the real zeros of nonzero members of $\mathcal{S}$. Then Z is a finite set. The points of Z are the 0-cells of the Collins decomposition of $\mathbb{R}^1$. The 1-cells are the open intervals into which Z divides $\mathbb{R}^1$. If $a, b \in Z$ and (a, b) is such a one-cell then $\frac{1}{2}(a+b)$ is the sample point of the one-cell. Distinct points of Z are separated by at least 2^{-C+6}. For each real root of each nonzero member of $\mathcal{S}$ find an approximation within $2^{-\mu-2}$ by a rational number w such that $2^{\mu+2}w$ is an integer. Let W be the set of numbers obtained. If w_1, w_2 approximate the same element of Z then $|w_1-w_2| \leqq 2^{-\mu}$. Otherwise, $|w_1-w_2| \geqq 2^{6-C}-2^{-\mu-1} \geqq 2^{5-C}$. Introduce an equivalence relation on W by calling $w_1, w_2 \in W$ equivalent when $|w_1-w_2| \leqq 2^{-C}$. This is an equivalence relation because members of W are equivalent if and only if they approximate the same element of Z within $2^{-\mu-2}$. Let Z' consist of one representative of each equivalence class in W. Define $\Sigma: Z \to Z'$ by $\Phi(z)=w$ if and only if $|w-z| \leqq 2^{-\mu-2}$. Then Φ preserves order in the real numbers. Extend Φ to the set of sample points of one-cells so that if (a, b) is a one-cell of the Collins decomposition then $\Phi(\frac{1}{2}(a+b))=\frac{1}{2}(\Phi(a)+\Phi(b))$. If $Z \neq \varnothing$, the rightmost one-cell has sample point $a+1$ for some $a \in Z$. Define $\Phi(a+1)=\Phi(a)+1$. Do similarly for the leftmost one-cell. The extension of Φ to all sample points is order-preserving and has image T', a set of approximate sample points. Let (a, b) be a one-cell in the Collins decomposition. Then $a+2^{-k}$ is an incidence point for the incidence of a on (a, b). The points $\Phi(a)$ and $\Phi(b)$ are consecutive points of Z'. Define the incidence point of the "one-cell" $\frac{1}{2}(\Phi(a)+\Phi(b))$ and the "0-cell" $\Phi(a)$ of T' to be $\Phi(a)+2^{-k}$. The points of Z' are the "0-cells" of T'. The remaining points are the one-cells. They may be constructed from Z' by use of the order in the real numbers of points of Z' and do not require a knowledge of Z or of Φ. All the properties of a set of approximate sample points in $\mathbb{R}^1$ are easily checked. For the special case where no nonzero member of $\mathcal{S}$ has real zeros, i.e. $Z=\varnothing$ we take 0 as a sample point for the one-cell and set $\Phi(0)=0$.

Let $0<j\leqq r$. We shall now modify the above procedure to construct a set T' of approximate sample points in $\mathbb{R}^{r-j+1}$ from a set Σ' of approximate sample points in $\mathbb{R}^{r-j}$. Suppose that $\Phi:\Sigma\to\Sigma'$ is a bijection from the set of sample points in the Collins cell decomposition of $\mathbb{R}^{r-j}$ to Σ' such as is required to exist in the definition of a set of approximate sample points.

Let $s\in\Sigma$ be fixed. Let $\mathcal{S}=\{f(s,x)|f\in P(j-1)\}$, a finite set of real polynomials of one variable. Let Z be the union of the real zeros of nonzero members of $\mathcal{S}$. Then $\{(s,z)|z\in Z\}$ is the set of sample points of the horizontal cells of the Collins decomposition over the cell Δ that contains s. If $Z=\emptyset$, then $(s,0)$ is the sample point of the vertical cell over Δ. If $Z\neq\emptyset$, then the sample points of the vertical cells over Δ are obtained in the usual way. That is, they are the points (s,w) where w is the average of successive members of Z or $1+\mu$ or $-1+\nu$, where μ and ν are the largest and smallest members of Z. This shows that the sample points over s can be computed by finding $\mathcal{S}$ and Z.

Let $\Phi(s)=\sigma$. Let $\mathcal{S}'=\{f(\sigma,x)|f\in P(j-1)\}$. We cannot approximate Z by using the union of the real zeros of nonzero members of $\mathcal{S}'$. The discussion of Remark 3.4.3 provides a means of using $\mathcal{S}'$ to get an approximation to Z.

For this paragraph only, let $N=n_{j-1}$ and $m=-(1+\mu_{j-1})$. Let $Z'=\{x\in\mathbb{R}|\exists f\in P(j-1)$ for which $f(\sigma,x)=\pm\frac{1}{2}\cdot(2^{Nm})\}$. By Remark 3.4.3, each element of Z' is within 2^m of some element of Z and each element of Z is within 2^m of some element of Z'. The distance between distinct elements of Z is at least 2^{6-C} and $\mu_{j-1}\geqq C$. For each $f\in P(j-1)$ and each real root of $f(\sigma,x)\pm\frac{1}{2}2^{Nm}$ choose a number w which approximates the root within 2^{m-1} such that $w2^{2+\mu_{j-1}}$ is an integer. Let Z'' be the set of numbers w obtained as f varies through $P(j-1)$ and all real roots of $f(\sigma,x)\pm\frac{1}{2}2^{Nm}$ are taken. Each element of Z'' is within $3(2^{m-1})$ of an element of Z and conversely. Introduce an equivalence relation on Z'' by calling any $w_1, w_2\in Z''$ equivalent if $|w_1-w_2|\leqq 2^{m+2}$. Again, elements $w_1, w_2\in Z''$ are equivalent precisely when they lie within $3(2^{m-1})$ of the same element of Z. Let Z''' consist of one element from each equivalence class in Z''.

Define $\psi:Z\to Z'''$ by $\psi(z)=w$ if and only if $w\in Z'''$ and $|w-z|\leqq\frac{3}{4}2^{-\mu_{j-1}}$. (If $u\in Z'''$ and $\psi(z)\neq u$, then $|u-z|\geqq 2^{5-C}$.) Call $\{(\sigma,w)|w\in Z'''\}$ the set of horizontal cells over σ. The vertical cells over σ are the points of $\{\sigma\}\times\mathbb{R}$ constructed from Z''' exactly as the set of sample points of vertical cells over s was constructed from Z. Define the dimension of cells over σ in the obvious way. Let T'_σ be the set of cells over σ. Let T be the set of sample points in the Collins decomposition of $\mathbb{R}^{r-j+1}$. Let $\pi:\mathbb{R}^{r-j+1}\to\mathbb{R}^{r-j}$ be projection on the first $(r-j)$-coordinates. Let $T_s=\{t\in T|\pi(t)=s\}$. Recall that $\Phi(s)=\sigma$. Then ψ induces an order-preserving isomorphism $\Psi_s:T_s\to T'_\sigma$. Let $T'=U_{\sigma\in\Sigma'}T'_\sigma$. Define $\Psi:T\to T'$ by $\Psi|T_s=\Psi_s$ for each $s\in\Sigma$. Then Ψ is a bijection. For fixed $s\in\Sigma$, the incidence of vertical and horizontal cells over the cell containing s is determined from the order of these cells. This order is the same as the order on the set T_s of sample points of such cells induced by the order in the real numbers of their last components. Define incidences among cells in T'_σ by the use of the order of last components of points in T'_σ. Then $\Psi_s:T_s\to T'_\sigma$ gives an isomorphism of incidence relations, because it preserves order.

We shall show how to define and compute incidences in T' so that Ψ gives isomorphisms of incidence relations. We have already seen how to define incidences in T'_σ for fixed $\sigma\in\Sigma'$ so that when $\Phi(s)=\sigma$ then $\Psi|T_s=\Psi_s$ induces an isomorphism of incidence relations. This corresponds to Case 1 in § 3.5.

Let $c, d\in\Sigma$ be sample points of incident cells γ, δ with $\dim\delta=1+\dim\gamma$. Let $\Phi(c)=c'$, $\Phi(d)=d'$. Let $b\in\mathbb{R}^{r-j}$ be the incidence point associated to the incidence of

γ and δ and let b' be the incidence point associated to the incidence of c' and d' in accordance with the definition of a set of approximate sample points. Then the ith components of b and b' differ by at most $2^{-\mu_{r-i}}$ for $i=1,\cdots,(r-j)$. Moreover, $b\in S(j,P,k)$. For each $f\in P(j-1)$, choose an approximation to within $2^{-2-\mu_{j-1}}$ to each real solution of $f(b',y)=\pm\frac{1}{2}2^{-n_{j-1}(1+\mu_{j-1})}$ by a number z such that $2^{2+\mu_{j-1}}z$ is an integer. Let W be the set of numbers z obtained as f varies through $P(j-1)$ and all real solutions are taken. Introduce an equivalence relation on W by calling points of W equivalent if they are within 2^{-C} of one another. Elements of W are equivalent if they are within $\frac{3}{4}2^{-\mu_{j-1}}$ of the same element of $V=\{y\in\mathbb{R}|\exists f\in P(j-1)$ with $f(b,y)=0\}$. Inequivalent elements of W are at distance at least 2^{4-C} apart. These statements follow from the discussion of Remark 3.4.3. Let W' consist of one element from each equivalence class from W. Distinct elements of V are at least 2^{6-C} apart. An order-preserving bijection $\lambda: V\to W'$ is defined by $\lambda(v)=w$ if and only if $w\in W'$ and $|w-v|\leqq 2^{-C}$. (See Remark 3.4.3 for details.) If $\lambda(v)=w$ then $|w-v|\leqq\frac{3}{4}2^{-\mu_{j-1}}$.

Let $U=\{z\in\mathbb{R}|\exists f\in P(j-1)$ with $f(c,z)=0\}$. Let $u\in U$, $v\in V$. Then u is the value at c of a root function r defined on γ and v is the value at b of a root function defined on δ. In § 3.5 we showed that the horizontal cell over γ defined by r is contained in the closure of the horizontal cell over δ defined by ρ when $|u-v|\leqq 2^{-k_{j-1}}$. Otherwise, $|u-v|\geqq 2^{5-k_0}$. We have constructed an order-preserving bijection $\lambda: V\to W'$. In defining Ψ, we constructed an order-preserving bijection $\psi: U\to U'$, where U' is the set of horizontal cells over c', and ψ moves each point at most $\frac{3}{4}2^{-\mu_{j-1}}$. The map λ also moves each point at most this much. If $|u-v|\leqq 2^{-k_{j-1}}$ then $|\psi(u)-\lambda(v)|\leqq 2^{-k_{j-1}}+\frac{3}{2}2^{-\mu_{j-1}}\leqq 3\cdot 2^{-k_{j-1}}$. Otherwise, $|\psi(u)-\lambda(v)|\geqq 2^{5-k_0}-\frac{3}{2}2^{-\mu_{j-1}}\geqq 30\cdot 2^{-k_{j-1}}$. To specify horizontal cells over γ and δ it suffices to specify where they appear in the ordering of such cells. These cells are in order-preserving bijection with the sets U and V. We may instead use U' and W'. They are also in order-preserving isomorphism with the horizontal cells over γ, resp. δ, and for each $u'\in U'$, $v'\in W'$ the horizontal cell corresponding to v' contains the cell corresponding to u' in its closure if $|u'-v'|\leqq 3\cdot 2^{-k_{j-1}}$. Otherwise, $|u'-v'|\geqq 30\cdot 2^{-k_{j-1}}$. We have seen in § 3.5 how to use the relation between U and V defined by $\{(u,v)||u-v|\leqq 2^{-k_{j-1}}\}$ to determine incidence relations of cells over γ with cells over δ. The relations defined between U' and W' by $\{(u',v')||u'-v'|\leqq 3\cdot 2^{-k_{j-1}}\}$ is equivalent *via* ψ and λ with this relation. Let $\Psi: T\to T'$ be defined as earlier. We now define incidences between cells of $T'_{c'}$ and $T'_{d'}$ as follows: The set of $H'_{d'}$ of horizontal cells in $T'_{d'}$ is bijective in an order-preserving way with W', V, and with the set of horizontal cells in T_d. The set $H_{c'}$ of horizontal cells in T'_c is bijective in an order-preserving way with U'. By use of the order-preserving bijections we get a relation between $H'_{d'}$ and $H'_{c'}$. We define this to be the incidence relation of horizontal cells over d' and c'. By use of the sandwiching of vertical cells between horizontal ones, the order of cells over d' and c' can be used as in § 2 to define incidence of vertical cells over d' and c'. With this definition, Ψ is an incidence-isomorphism. The definition just depends on inequalities involving elements of U' and W' and the order of certain numbers in the reals. It can be computed without knowing U and V.

It remains to show how to define incidence points for T'. Let $\sigma\in\Sigma'$. If $a<b$ and (σ,a) and (σ,b) are consecutive horizontal cells of T'_σ then we define an incidence point for the incidence of (σ,a) with the vertical cell $(\sigma,\frac{1}{2}(a+b))$ to be $(\sigma,a+2^{-k_{j-1}})$. All incidence points for incidences of two different cells over σ are defined similarly. As above in the definition of incidences in T', let $c,d\in\Sigma$ be sample points of incident cells γ, δ, etc.

Suppose the ith element of the ordered set $H'_{c'}$ is incident on the jth element of $H'_{d'}$. Let w' be the jth element of the ordered set W'. Use (b',w') as the incidence

point in the definition of a set of approximate sample points. Suppose that (c', x') and (d', y') are incident vertical cells. As an incidence point, we take (b', x').

The elements of T' which correspond to bounded cells in the Collins decomposition are those elements $(\sigma, c) \in \Sigma' \times \mathbb{R}$ such that σ corresponds to a bounded cell and (σ, c) is not the first or last vertical cell over σ. This enables one to determine immediately by iteration which cells correspond to bounded cells.

4. Computation time. In this section we show the computation time for the algorithm of § 3.6 is polynomial. This extends the work of Collins [6] in that we also compute incidences. Many techniques are the same as he used.

4.1. Regularizing coordinates and subresultants.

LEMMA 1. *Let $g \in \mathbb{Z}[x_1, \cdots, x_r]$ have degree $\leqq n$ and $\log \|g\| \leqq M$. Suppose $a_1, \cdots, a_r$ are integers, $c \geqq 0$, and $\log |a_i| \leqq c$ for all i. Then $g(a_1, \cdots, a_r)$ can be computed in time bounded by*

$$(c+1)(r+1)(M+rc)n^r.$$

Proof. We use induction on r. Let A be the maximum time needed to compute $g(a_1, \cdots, a_r)$ for any such g. Let B be the maximum time needed to compute any polynomial h of $(r-1)$ variables with degree $(h) \leqq n$ and $\log \|h\| \leqq M$. By Horner's rule [13] for polynomials of one variable

$$u = \sum_{j=0}^{n} u_j x^j \quad \text{is computed by} \quad u = (\cdots (u_n x + u_{n-1})x + \cdots)x + u_0.$$

We regard g as a polynomial in x_r. All coefficients can be evaluated at $(a_1, \cdots, a_{r-1})$ in total time at most nB. Once this is done, $g(a_1, \cdots, a_r)$ can be computed by Horner's rule. Each intermediate stage has log bounded by $(M+rc)$. We must perform (at most) n additions of numbers satisfying this bound and multiply by a_r at most n times. Then

$$A \leqq nB + n(M+rc) + n(M+rc)c = n((M+rc)(c+1)+B).$$

The result follows by induction on r.

LEMMA 2. *Let $\mathcal{S} \subset \mathbb{Z}[x_1, \cdots, x_r]$ be a set of at most s nonzero polynomials such that each $g \in \mathcal{S}$ has degree at most n and satisfies $\log \|g\| \leqq M$. Then a regularizing coordinate change φ may be found in time at most*

$$(M+r)(1+ns)^{2r+3}.$$

Proof. Let $ns > 1$. We may suppose $r \geqq 2$. We seek a vector $v \in \mathbb{Z}^r$ with $v_r \neq 0$ and each component bounded by $ns-1$ such that for each $g \in \mathcal{S}$ the degree of g is the same as that of $g(x, v)$. Such vectors exist. We may find one by evaluating the top degree homogeneous parts of the elements of $\mathcal{S}$ for at most $(ns+1)^r$ such vectors. The time is bounded by Lemma 1.

Remark. A modular algorithm for finding a regularizing coordinate change is probably faster, but not needed for present purposes.

Let $\varphi : \mathbb{R}^r \to \mathbb{R}^r$ be a coordinate change with integral matrix which is a regularizing coordinate change of the type considered previously, i.e., $\psi(x_1, \cdots, x_r) = (\varphi(x_1, \cdots, x_k), x_{k+1}, \cdots, x_r)$, where φ either permutes variables or is of the form $\varphi(x_1, \cdots, x_k) = (x_1, \cdots, x_{k-1}, 0) + x_k v$ and $v \in \mathbb{Z}^k$ has components bounded by $d-1$.

LEMMA 3. *Let ψ be as above. Let $g \in \mathbb{Z}[x_1, \cdots, x_r]$ have degree at most n and satisfy $\log \|g\| \leqq M$. Then $g \circ \psi$ can be computed in time bounded by*

$$r(n+1)^r(M + n \log d)(1 + \log d).$$

Proof. If ψ permutes coordinates, we shall neglect the time needed to compute $g \circ \psi$. Consider the second case. The polynomial g can be developed as a polynomial in the last $(r-k)$ variables. As such, it is a sum of at most $(n+1)^{r-k}$ monomials and each coefficient is a polynomial h in $x_1, \cdots, x_k$ of degree at most n with $\log \|h\| \leqq M$. The polynomial $h \circ \varphi$ can be computed by repeated use of Horner's rule. Intermediate results have logs bounded by $M + n \log d$. By induction we get

$$B \leqq k(n+1)^k(M + n \log d)(1 + \log d).$$

The result follows.

The author thanks S. Gonek for a helpful conversation about the next lemma.

LEMMA 4. *For $x \geqq 2^{12}$, let $P = \prod_{p \leqq x} p$, the product running through the primes. Then* $\log_2 P \geqq 10 + \frac{1}{2}x$.

Proof. By refinements of material in Hardy and Wright [10] or otherwise the standard number-theoretical functions

$$\theta(x) = \ln(P),$$

$$\psi(x) = \sum_{p \leqq x} \left[\frac{\log x}{\log p}\right] \ln p,$$

satisfy

$$0 \leqq \psi(x) - \theta(x) < x^{1/2} \ln x \quad \text{for } x \geqq 2^8,$$

$$\psi(x) \geqq \left(\frac{x}{10} - 1\right) \ln 252 \geqq \frac{x}{2} - 6 \quad \text{for } x \geqq 10.$$

Then $\theta(x) > x/2 - 6 - x/7$ for $x \geqq 2^{12}$. Then $\theta(x) > (\frac{1}{2} \ln 2)x + 10$ for $x \geqq 2^{12}$.

We obtain immediately:

LEMMA 5. *Let $M \geqq 2^{11}$. Let Π be the set of primes p for which $p < 6M$. Then the product of the primes in Π exceeds 2^{3M+10}.*

LEMMA 6. *The time to find all primes in the set Π of Lemma 5 and to compute the arithmetic of $\mathbb{Z}/(p)$ for all such primes p is dominated by*

$$M^3(\log M)^2.$$

Proof. By [9] we may find the elements of Π in time dominated by $M(\log M)^2$. For each prime p we may work out arithmetic modulo p in time dominated by $p^2(\log p)^2$. The number of primes in Π is dominated by M. The result follows.

LEMMA 7. *Let $M \geqq 2$. Let $\mathcal{S} \subset \mathbb{Z}[x_1, \cdots, x_r]$ be a set of at most s polynomials each of degree at most n and suppose $\log \|g\| \leqq M$ for every $g \in \mathcal{S} \cup \mathcal{S}'$. Suppose Π is a set of primes $p > n^2$ such that the product of the primes in Π exceeds 2^{3M+10} and the arithmetic modulo each prime of Π is known. Then $\mathcal{S}'$ can be computed in time not exceeding*

$$KrM^3(n^2+1)rns^2, \quad \textit{where } K \textit{ is independent of } r.$$

Proof. This follows *mutatis mutandis* from the proof of Collins' modular algorithm for computing subresultants [7] and the fact that $\mathcal{S}'$ has at most ns^2 elements.

Remark. Several authors have improved algorithms which compute subresultants. See [18].

PROPOSITION 1. *Let $r \geqq 2$. Let $P \subset \mathbb{Z}[x_1, \cdots, x_r]$ be a finite set of s polynomials each of degree at most n. Suppose $M \geqq 2$ and $\log \|g\| \leqq M$ for each $g \in P$. Then one can find regularizing coordinate changes $\psi_0, \cdots, \psi_{r-2}$ and compute $P(0), \cdots, P(r-1)$ as described in* § 3.1 *in time bounded by*

$$Kr^3L^4(1 + \nu\sigma)^{2r+3},$$

where K is a constant independent of all variables. (The quantities L, ν, σ are defined in the proof. They are dominated by polynomials in M, n, s of degree at most a constant times $r^2 2^r$. The coefficients of these polynomials depend on r with similar growth constraints.)

Proof. The computation requires several types of steps.

1. Finding an adequate set of primes for the following computations.
2. Finding a regularizing coordinate change.
3. Computing $\{g \circ \psi | g \in \mathscr{S}\}$ where $\mathscr{S}$ is a finite set of polynomials and ψ is a regularizing coordinate change.
4. Computing $\mathscr{S}'$ for a finite set $\mathscr{S}$ of polynomials.

Each step is applied to a (different) finite set of polynomials in at most r variables consisting of at most $\sigma = s_{r-2}$ polynomials of degrees at most $\nu = n_{r-2}$. For each such finite set $\mathscr{T}$ of polynomials $L = L_{r-1}$ is a bound for $\log \|g\|$ for each $g \in \mathscr{T} \cup \mathscr{T}'$.

Find a set of primes whose product exceeds $2^{3L+10+2\nu^2}$. Discard those less than ν^2. Find the arithmetic modulo each remaining prime. Since $\theta(\nu^2) < 2\nu^2 \ln(2)$, the product of these primes exceeds 2^{3L+10}. According to Lemmata 4–6, all this can be done in time bounded by a constant times L^4.

By Lemma 2, all regularizing coordinate changes can be found in time dominated by $r(L+r)(1+\nu\sigma)^{2r+3}$.

By Lemma 3, each instance of Step 3 takes time dominated by

$$\sigma^2 \nu r(\nu+1)^r (L + \nu \log(\sigma\nu))(1+\log(\sigma\nu)).$$

There are at most r^2 such steps.

By Lemma 7, each occasion of a step of the fourth type takes time dominated by

$$rL^3(\nu^2+1)^r \nu\sigma^2.$$

There are at most r such steps. Thus the total time is dominated with constant independent of r by

$$r^3 L^4 (1+\nu\sigma)^{2r+3}.$$

An estimate of this type in the absence of coordinate changes and finding primes is given by Collins in [6].

4.2. Approximate sample points. Let us now examine the time needed for the computations in the construction of § 3.6. We retain the notation of that section. The set-up is the following.

Basic procedure. Let $0 < j \leqq r$. Let $b' \in \mathbb{R}^{r-j}$ approximate an element b of $\mathbb{R}^{r-j} \cap S(j, P, k)$ so that $|b_i - b_i'| \leqq \frac{3}{4} 2^{-\mu_{r-i}}$ and $4b_i' 2^{\mu_{r-i}}$ is an integer for $i = 1, \cdots, r-j$. (Delete this condition when $j = r$.) Let $\mathscr{S} = \{f(b', x) | f \in P(j-1)\}$. We must perform several steps:

1. Compute an element g of $\mathscr{S}$.
2. If g is not constant, approximate each real root of g or of $g \pm \frac{1}{2} 2^{-n_{j-1}(1+\mu_{j-1})}$ within $2^{-\mu_{j-1}-2}$ by a number z for which $2^{\mu_{j-1}+2} z$ is an integer.
3. Let Z be the set of numbers obtained by 2 for varying g. Order the elements of Z.
4. Call z_1, z_2 equivalent if $|z_1 - z_2| \leqq 2^{2-\mu_{j-1}}$. If z_1 and z_2 are inequivalent then $|z_1 - z_2| \geqq 2^{4-C}$. Form an ordered set Z' from Z by selecting one element from each equivalence class.

As for Step 1, let $f \in P(j-1)$ with b' as above. Let $\nu = n_{j-1}$. Let $(y, x) \in \mathbb{R}^{r-j} \times \mathbb{R}$ and expand $f(y, zx) = \sum_{\alpha \leqq \nu} f_\alpha(y) x^\alpha$ in powers of x. We shall compute $2^{\nu(2+\mu_{j-1})} f(b', x)$ instead of g as it has integer coefficients.

Let $q = 2^{2+\mu_{j-1}}$. Let $f_\alpha(y) = \sum_\beta c_{\alpha\beta} y^\beta$.

$$q^\nu f(y, x) = \sum_{\alpha \leqq \nu} g_\alpha(qy) x^\alpha, \quad \text{where } g_\alpha(z) = \sum_\beta c_{\alpha\beta} q^{\nu - |\beta|} z^\beta.$$

We can compute g_α for all α in time bounded by $(\nu+1)^{r+1} \log \|f\|$. Moreover, $\sum_\alpha \|g_\alpha\| \leqq q^\nu \|f\|$. By Lemma 1 of this section we may compute $g_\alpha(qb')$ for each α in time bounded by

$$r(1+c)(\nu(2+\mu_{j-1}) + \log \|f\| + (r-1)(1+c))\nu^{r-1},$$

where $c = (2+\mu_{j-1}) + \log(R_0+1)$. Thus, we compute $q^\nu f(b', x)$ in time which is polynomial in the basic parameters. Also,

$$\|q^\nu f(b', x)\| \leqq q^\nu (R_0+1))^\nu \|f\|,$$

so that $\log \|q^\nu f(b', x)\|$ is bounded by an easily computed polynomial δ in the basic parameters.

We are concerned in Step 2 with approximating real roots of $q^\nu f(b', x) = G(x)$ or of $G(x) \pm 1$ within $2^{-\mu_{j-1}-3}$. We apply the following

THEOREM (Heindel [11, Cor. 7.1]). *If G is a polynomial with integer coefficients of degree at most $\nu \geqq 1$, $\delta = \log \|G\| \geqq 0$, $\mu \geqq 1$, then the real zeros of G can be found within accuracy $2^{-\mu}$ in time dominated by*

$$\nu^{13}(\delta + \mu)^3.$$

Application to our situation gives a polynomial time bound for Step 2 of the basic procedure. Recall the inequalities of § 3.4.4. Three additional inequalities will apply in the applications of Heindel's theorem.

$$\mu \leqq 3 + \mu_{r-1},$$

$$\nu \leqq n_{r-1},$$

$$\delta \leqq \nu(\mu + rL) + L.$$

For fixed b' we must compute roots of at most $3s_{j-1}$ polynomials.

The time for Steps 3 and 4 is comparatively small.

We see that the estimates obtained for performing 1, 2, 3, 4 for all elements of $\mathcal{S}$ are dominated by an expression such as

$$s_r(\bar{\delta})^3(\bar{\nu})^{13} + s_r(\bar{\nu})^{r-1} r^2 (\bar{\delta})^2,$$

where the quantities $\bar{\nu}$ and $\bar{\delta}$ are defined by making the last three inequalities into equations.

LEMMA 8. *The number of cells in the Collins decomposition of $\mathbb{R}^{r-j}$ associated to P is bounded by $3^r n_r s_r$. The number of cells having dimension one less than a fixed cell and contained in its closure is bounded by $4^r n_{r-1} s_{r-1}$.*

Proof. Induction.

COROLLARY. *The basic procedure must be done at most $(12)^{r+1} r s_{r+1}$ times.*

Proof. We must do the basic procedure once for each cell and once for each pair of incident cells of consecutive dimension. This holds for $j = 1, \cdots, r$. Combine the lemma with the formula for s_{r+1}.

For each incidence point we must compare the numbers of two lists obtained by the basic procedure. For a fixed pair of lists this takes time dominated by $(n_{r-1}s_{r-1})^2(\delta + \mu)^2$, which is less than the bound before Lemma 8. The remaining work in going from a set of approximate sample points in $\mathbb{R}^{r-j}$ to a set of approximate sample points in

$\mathbb{R}^{r-j+1}$ is minor bookkeeping. Naturally we want to identify the members of the set of approximate points in $\mathbb{R}^r$ that correspond to cells in the Collins decomposition which satisfy the given system of d inequalities in members of P. We use the following lemma.

LEMMA 9. *Let $f \in P(0)$. Let a^* be an approximate sample point which corresponds to the cell Δ of the Collins decomposition of $\mathbb{R}^r$. If*

$$-\log |f(a^*)| \geqq (3rn+1)Ln_{r+1}$$

then f is zero on all of Δ. Otherwise, f is positive or negative on Δ according as $f(a^*)$ is positive or negative.

Proof. Let $a \in S(0, P, (0, \cdots, 0))$ be the sample point of Δ. By construction, f is positive, zero, or negative on Δ according as $f(a)$ is.

All conjugates of $\alpha = f(a)$ are bounded by $2^{M_0}R_0^n$. There is a positive integer τ with $\log \tau \leqq \delta_0 = \delta_0(P, (0, \cdots, 0))$ such that $\tau^n\alpha$ is an algebraic integer. The degree over the rationals of α is bounded by n_{r+1}. If α is not zero, then the norm of $\tau^n\alpha$ is a nonzero rational integer. Then

$$\tau^n|\alpha|(\tau^n 2^{M_0}R_0^n)^{n_{r+1}-1} \geqq 1.$$

Thus,

$$-\log|\alpha| \leqq 3rLnn_{r+1} + (L-1)n_{r+1},$$

$$|f(a^*) - f(a)| \leqq 2^{M_0}(2R_0)^{n-1}2 \cdot 2^{-\mu_0}.$$

Thus, if

$$\mu_0 \geqq (3n+1)(n_{r+1}+1)rL,$$

then $|f(a^*) - f(a)| \leqq \frac{1}{2}|\alpha|$. Then $f(a)$ and $f(a^*)$ have the same sign if they are nonzero and $f(a) = 0$ if and only if $-\log|f(a^*)| \geqq (3n+1)rLn_{r+1}$. In Definition 3.4.2, we chose $A > n_{r+1}(5rL)+6$ and defined the k_j accordingly. Then C as defined at the end of § 3.3 is sufficiently so large that μ_0 as in Lemma 3.4.1 satisfies the inequality above. For f and a^* in the above lemma, we may (by Lemma 1 of this section), compute $f(a^*)$ in time bounded by

$$5(3+\mu_{r-1})(r+1)(n+1)^{r+1}.$$

We must do this for all $f \in P(0)$. The time to do so for all approximate sample points is bounded above by the expression before Lemma 8. We must find signs for each approximate sample point. Doing so takes less time than carrying out the Basic Process for all approximate sample points. If a semi-algebraic set is given by a Boolean formula of length a in atoms of the form $\{x|f(x) > 0\}$, $\{x|f(x) = 0\}$, $\{x|f(x) < 0\}$ where in each case $f \in P$, then after we know "signs" of the members of P at approximate sample points, it takes time at most $3^r n,s,a$ to find which approximate sample points correspond to cells of the semi-algebraic set.

By collecting earlier results and manipulating them we have:

THEOREM. *Let X be a semi-algebraic set defined by a formula a-atoms long in members of a set P as throughout this paper. By a coordinate change, it is possible to find a Collins decomposition of $\mathbb{R}^r$ for P such that the boundary of each cell is a union of lower dimensional cells and X is a union of some of these cells. It is possible to construct an "isomorphic abstract cell complex" with the same incidences as the Collins decomposition. This complex, its bounded cells, incidences of its cells, and the cells which correspond*

under the isomorphism to the cells of X can be computed in polynomial time. More explicitly, the time is dominated (for example) by

$$2^{\alpha} a r^{b} s^{c} n^{d} L^{e},$$

where

$$\alpha = 32r + 14, \qquad d = r^2 2^{r+7},$$
$$b = 40, \qquad e = 13.$$
$$c = r2^{r+2} + 13,$$

5. Comments and speculations. We wish to emphasize that the algorithm of § 3.6 and its time analysis are in a crude form, not intended for practical use. The general implications of the time estimates are more important than the particular results. Many improvements of the cylindrical algebraic decomposition of Collins are given in [6], [2], [18] and elsewhere. We felt that to incorporate them would obscure the idea of § 3.6. We could determine locally the accuracy needed for approximate sample points and incidence points. Most points do not interact. We could replace Lemma 3.4.1 by estimates that take the multiplicity at each point into account. Most of the estimates could be sharpened with little effort and these improvements could be incorporated into the tolerance limits of the algorithm. It seems clear that the algorithm described in § 3.5 would also run in polynomial time. The techniques needed for manipulating real algebraic numbers to show this are reviewed in [6] and [18].

The main reason for the time analysis was to get a hold on the exponents and coefficients in the polynomial bound. They are dishearteningly large, as for Collins' original algorithm. This is mainly an artifact of the growth as j increases in the size of the parameters associated to the sets $P(j)$.

To use the cell decomposition to compute homology one should orient the cells. This is discussed by Schwartz and Sharir. The idea is to use induction. If c is an oriented cell in the Collins decomposition of $\mathbb{R}^{r-j}$, then $c \times \mathbb{R}$ is oriented by using the usual orientation on $\mathbb{R}$. The vertical cells of $c \times \mathbb{R}$ are oriented so that inclusion is orientation-preserving. The signs in formula for the boundary of a cell depend on the order of cells over c and the signs in boundary formulae in lower dimensions so they are computable in the context of approximate sample points.

Under certain circumstances a closed semi-algebraic set has the same homotopy type as the union of its bounded cells. This union is a regular cell complex whose homology we can compute. In fact, suppose that each cylinder in the decomposition of $\mathbb{R}^{r-j}$ contains a horizontal cell for $j = 0, \cdots, r-1$. (A horizontal cell in $\mathbb{R}$ is a point.) If necessary, we can arrange this by modifying the definition of $Q(j)$ in § 2 so that $Q(0) = P \cup \{x_r\}$ and

$$Q(j+1) = (Q(j) \circ \psi_j)' \cup \{x_{r-j-1}\}.$$

It can then be shown by induction on r that for each unbounded cell Δ of the Collins decomposition there exists a deformation retraction of $\bar{\Delta}$ to $(\bar{\Delta} - \Delta)$. The assertion at the start of this paragraph follows.

Known bounds for Betti numbers of algebraic sets have exponents which are polynomial in the number of dimensions also. They come from cell decompositions obtained by Morse theory. Possibly such methods could be adapted to algorithmic purposes.

Note. At the SIAM Conference on Geometric Modeling and Robotics, July, 1985, C.-K. Yap announced that he and Kozen had obtained a result on computation of incidences in polynomial time. They and this author have worked independently.

REFERENCES

[1] A. G. AKRITAS, *The fastest exact algorithm for the isolation of the real roots of a polynomial equation*, Computing, 24 (1980), pp. 299-313.

[2] D. S. ARNON, *Algorithms for the geometry of semi-algebraic sets*, Tech. Report 436, Computer Science Dept., Univ. Wisconsin, Madison, WI, 1981.

[3] D. S. ARNON, G. E. COLLINS AND S. MCCALLUM, *Cylindrical algebraic decomposition* II: *An Adjacency algorithm for the plane*, Tech. Report CSDTR-428 Computer Science Department, Purdue U., West Lafayette, IN, December 1982, this Journal, 13 (1984), pp. 878-889.

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

[5] W. S. BROWN AND J. F. TRAUB, *On Euclid's algorithm and the theory of subresultants*, J. Assoc. Comput. Mach., 18, pp. 505-514.

[6] G. E. COLLINS, *Quantifier elimination for real closed fields by cylindrical algebraic decomposition*, in Second G.I. Conference on Automata Theory and Formal Languages, Lecture Notes in Computer Science 33, Springer-Verlag, Berlin, pp. 134-183.

[7] ———, *The calculation of multivariate polynomial resultants*, J. Assoc. Comput. Mach., 18 (1967), pp. 128-142.

[8] B. GIESECKE, *Simpliziale Zerlegung abzählbarer analytischer Räume*, Math. Z., 83 (1964), pp. 177-213.

[9] D. GRIES AND J. MISRA, *A linear sieve algorithm for finding prime numbers*, Comm. ACM, 21 (1978), pp. 999-1003.

[10] G. A. HARDY AND E. M. WRIGHT, *An Introduction to the Theory of Numbers*, Fourth Edition, Oxford Univ. Press, Oxford, 1960.

[11] L. E. HEINDEL, *Integer arithmetic algorithms for polynomial real zero determination*, J. Assoc. Comput. Mach., 22 (1971), pp. 533-549.

[12] H. HIRONAKA, *Triangulations of algebraic sets*, in Proc. Symposia in Pure Math., 29, American Mathematical Society, Providence, RI, pp. 165-185.

[13] D. E. KNUTH, *The Art of Computer Programming, vol.* II, *Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.

[14] B. O. KOOPMAN AND A. B. BROWN, *On the covering of analytic loci by complexes*, Trans. Amer. Math. Soc., 34, pp. 231-251.

[15] J. MILNOR *On the Betti numbers of real varieties*, Proc. Amer. Math. Soc., 15 (1964), pp. 275-280.

[16] O. A. OLEĬNIK, *Estimates of the Betti numbers of real algrebraic hypersurfaces*, Mat. Sb. (N.S.), 28 (1951), pp. 635-640.

[17] J. R. PINKERT, *Algebraic algorithms for computing the complex zeros of Gaussian polynomials*, Ph.D. thesis, Computer Sciences Dept., Univ. Wisconsin, Madison, WI, 1973.

[18] J. T. SCHWARTZ AND M. SHARIR, *On the "Piano Movers Problem"* II. *General techniques for computing topological properties of real algebraic manifolds*, Tech. Report 41, Computer Science Department, New York University, New York, 1982. (Appeared in Adv. Appl. Math. 12 (1983), pp. 298-351.)

[19] R. THOM, *Sur l'homologie des variétés algébriques réelles*, Differential and Combinatorial Topology: A Symposium in Honor of Marston Morse, Princeton Univ. Press, Princeton, NJ, 1965, pp. 255-265.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986

© 1986 Society for Industrial and Applied Mathematics
008

LOG DEPTH CIRCUITS FOR DIVISION AND RELATED PROBLEMS*

PAUL W. BEAME†, STEPHEN A. COOK† AND H. JAMES HOOVER†

Abstract. We present optimal depth Boolean circuits (depth $O(\log n)$) for integer division, powering, and multiple products. We also show that these three problems are of equivalent uniform depth and space complexity. In addition, we describe an algorithm for testing divisibility that is optimal for both depth and space.

Key words. integer division, circuit depth, circuit complexity, depth complexity, space complexity

AMS(MOS) subject classification. 68Q

1. Introduction. It is a well-known fact that addition, subtraction and multiplication on modern computers are significantly faster operations than division. Circuit designers have been unable to match the efficiency of the circuits for addition and multiplication in division circuits. Until recently there seemed to be some theoretical justification for this inability since the best known circuits for the first three problems have $O(\log n)$ depth but division appeared to have only $O((\log n)^2)$ depth circuits.

Reif [8] reduced the division depth to $O(\log n(\log\log n)^2)$ using a circuit for computing the product of $n^{O(1)}$ n-bit integers mod 2^n+1, based on Fourier interpolation and evaluation. This circuit had slightly more than polynomial size, but a revised version of the result [9] yields polynomial size and $O(\log n \log\log n)$ depth circuits for the same problem.

We present simple circuits of depth $O(\log n)$ and polynomial size, using Chinese remaindering, for the division of two n-bit integers and for the product of n n-bit integers. Since the circuits we consider allow gates with fan-in at most two, our division and iterated product circuits are optimal in depth up to a constant factor.

Besides circuit depth complexity we are also interested in the deterministic space complexity of division. Borodin [3] showed that if for all n a problem can be solved for n input bits by a circuit of depth $O(D(n))$ then it can be solved in Turing machine space $O(D(n))$, provided the circuits are "log-space uniform" (i.e. some Turing machine, given any n on its input tape, can generate a description of the circuit for n inputs in $\log n$ space). Since Reif's circuits mentioned above are log-space uniform, it follows that integer division has space complexity $O(\log n \log\log n)$. Unfortunately our circuits for division may not quite be log-space uniform, and it remains an open question whether division has space complexity $\log n$.

Motivated by this question, we prove a number of results. First we show that the three problems division, powering, and iterated product are each strongly reducible to either of the others. Thus all three have the same uniform depth complexity and the same space complexity. Next we give a simple sufficient condition (that some "good modulus sequence" $\{M_n\}$ be log-space generable) for the three problems to have space complexity $\log n$. Finally we show that the problem of testing whether an

* Received by the editors September 12, 1984, and in revised form July 3, 1985. A preliminary version of this paper appeared in the Proceedings of the 25th IEEE Symposium on the Foundations of Computer Science, 1984.

† Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4.

n-bit integer is divisible by another does indeed have uniform depth complexity $O(\log n)$ and hence space complexity $O(\log n)$.

2. Circuits and uniformity. We adopt the usual definition of fan-in two Boolean circuit families in which the nth circuit has $g(n)$ inputs and $h(n)$ outputs where g and h are nondecreasing polynomially bounded functions. With this definition depth $O(\log n)$ implies polynomial size. Using the notion of uniformity (see the Introduction) we can define a basic complexity class:

DEFINITION [10]. The class NC^1 consists of all functions f computable by a log-space uniform circuit family of depth $O(\log n)$.

Thus every function in NC^1 has deterministic space complexity $O(\log n)$ [3]. Using standard methods [11] it is easy to see that multiplication of two n-bit integers and addition of n n-bit integers are each in NC^1. It remains an open question whether division of two n-bit integers is in NC^1.

Although log-space uniformity is desirable for theoretical reasons, there is a weaker kind of uniformity which provides a natural condition on circuit families. The builder of computer hardware may simply want to have fast circuits which are easy to construct. Once a circuit has been constructed, it will be used over and over again. We thus propose the following definition:

DEFINITION. A family $\langle \alpha_n \rangle$ of circuits is *P-uniform* provided some deterministic Turing machine can compute the transformation $1^n \to \bar{\alpha}_n$ in time $n^{O(1)}$ where $\bar{\alpha}_n$ is the standard encoding [10] of α_n.

Some of our circuits require internal constants which are polynomial-time computable but do not appear to be log-space computable, and thus are only P-uniform. However, even though they may not be log-space uniform they almost are, in that the only parts of the circuits which are not log-space constructible may be generated in $O(\log n \log\log n)$ space using Reif's powering algorithm [9].

A useful notion of reducibility for circuits is the following definition [4].

DEFINITION. f is NC^1 *reducible to* g if and only if there exists a log-space uniform circuit family $\langle \alpha_n \rangle$ which computes f with depth $(\alpha_n) = O(\log n)$ where, in addition to the usual nodes, oracle nodes for g are allowed. An *oracle node* is a node which has some sequence $y_1, \cdots, y_r$ of input edges and $z_1, \cdots, z_s$ of output edges with associated function $(z_1, \cdots, z_s) = g(y_1, \cdots, y_r)$. For the purpose of defining depth, the oracle node counts as depth $\lceil \log(r+s) \rceil$.

An important consequence of this definition is that if f is NC^1 reducible to g and g is computable by depth $O(\log^k n)$ uniform circuits then f is also computable by depth $O(\log^k n)$ uniform circuits. This applies whether "uniform" means "log-space uniform" or "P-uniform".

3. Powering and division are equivalent. Let x, y be n-bit positive integers. The DIVISION problem is to compute the n-bit representation of $\lfloor x/y \rfloor$. The POWERING problem is to compute the n^2-bit representation of x^i for $i = 0, \cdots, n$. The following result is adapted from [5].

THEOREM 3.1. DIVISION *is* NC^1 *reducible to* POWERING.

Proof. For integers x, y where $0 < x < 2^n$, $2 \leqq y < 2^n$ we wish to compute $\lfloor x/y \rfloor$. We first compute an under-approximation $\bar{y}^{-1}$ of y^{-1} with error $< 2^{-n}$. Then we compute $t = x\bar{y}^{-1}$ which approximates x/y with error < 1, and determine which one of $\lfloor t \rfloor$ or $\lfloor t \rfloor + 1$ is $\lfloor x/y \rfloor$.

Let $u = 1 - y2^{-j}$ where $j \geqq 2$ is an integer such that $2^{j-1} \leqq y < 2^j$. Thus $|u| \leqq \frac{1}{2}$. Consider the series $y^{-1} = 2^{-j}(1-u)^{-1} = 2^{-j}(1 + u + u^2 + \cdots)$. Set $\bar{y}^{-1} = 2^{-j}(1 + u + \cdots + u^{n-1})$. Then $y^{-1} - \bar{y}^{-1} \leqq 2^{-j} \sum_{i \geqq n} 2^{-i} < 2^{-n}$.

The circuit computes $\lfloor x/y \rfloor$ using scaled arithmetic of n^2 bits of precision as follows:

(1) Determine $j \geqq 2$ such that $2^{j-1} \leqq y < 2^j$ and compute $u = 1 - y2^{-j}$.
(2) Evaluate u^i, $i = 0, \cdots, n-1$ using the n-bit powering circuit.
(3) Compute $\bar{y}^{-1} = 2^{-j}(1 + u + \cdots + u^{n-1})$.
(4) Compute $t = x\bar{y}^{-1}$ and truncate to obtain $\lfloor t \rfloor$. Note that $xy^{-1} \geqq x\bar{y}^{-1} \geqq xy^{-1} - 2^{-n}x$.
(5) Compute $r = x - y\lfloor t \rfloor$ and determine whether $\lfloor x/y \rfloor$ is $\lfloor t \rfloor$ or $\lfloor t \rfloor + 1$.

All of these steps have depth $O(\log n)$ except possibly the powering in step (2). □

THEOREM 3.2. POWERING *is* NC^1 *reducible to* DIVISION.

Proof. Let x be an n-bit integer. We want to compute $x^0, \cdots, x^n$. We use a similar identity to the one in the previous reduction but in reverse, choosing a scaling factor so that none of the powers of x overlaps in the resulting binary representation.

$$\frac{2^{2n^3+2n^2}}{2^{2n^2}-x} = 2^{2n^3}\frac{1}{1-2^{-2n^2}x} = \sum_{i \geqq 0} 2^{2n^2(n-i)}x^i.$$

Note that $\sum_{i>n} 2^{2n^2(n-i)}x^i = 2^{-2n^2}x^{n+1}\sum_{j \geqq 0}(x2^{-2n^2})^j$ which is $\ll \frac{1}{2}\sum_{j \geqq 0}(2^{-n})^j < 1$.

The circuit for computing $x^0, \cdots, x^n$ will implement the following procedure:

(1) Set $u = 2^{2n^3+2n^2}$ and compute $v = 2^{2n^2} - x$.
(2) Evaluate $y = \lfloor u/v \rfloor$ using the $2n^3 + 2n^2$- bit division circuit. From the above identity it follows that $y = \sum_{0 \leqq i \leqq n} 2^{2n^2(n-i)}x^i$.
(3) Read off x^{n-i} as the bits in positions $2n^2i$ to $2n^2(i+1) - 1$ from the right in y (position 0 contains the low order bit).

All of these steps have depth $O(\log n)$ except possibly the division in step (2). □

4. Arithmetic operations modulo small integers. The results of this section are due to McKenzie and Cook [7].

For x and m integers we write $x \bmod m$ for the unique integer y such that $y \equiv x \bmod m$ and $0 \leqq y < m$.

LEMMA 4.1. *For inputs x of n bits and $m \leqq n$ the problems of computing $x \bmod m$, $\lfloor x/m \rfloor$, or $x^{-1} \bmod m$ (if an inverse exists) are all in NC^1.*

Proof. Consider the mod computation first. In space $O(\log n)$ for each $m \leqq n$ we may compute $a_{im} = 2^i \bmod m$ for $i = 0, \cdots, n-1$ and hardwire them into the circuit. Let $x = \sum_{i=0}^{n-1} x_i 2^i$. Then $x \bmod m = \sum_{i=0}^{n-1} x_i a_{im} \bmod m$. The circuit computes $y = \sum_{i=0}^{n-1} x_i a_{im}$ and reduces the result mod m by subtracting off in parallel the multiples of m—$0, m, \cdots, (n-1)m$—and choosing the appropriate difference. Since y has $O(\log n)$ bits the circuit has $O(\log n)$ depth. In order to compute $z = \lfloor x/m \rfloor$ use the above circuit and apply an NC^1 reduction from division to mod computation given by Alt and Blum [2]. Namely, for $i = 0, \cdots, n$ bit z_i is 1 if and only if $2(x_n \cdots x_{i+1} \bmod m) + x_i \geqq m$. To compute $x^{-1} \bmod m$, first compute $y = x \bmod m$ and then in parallel multiply y by each residue z modulo m and find the z for which the result is $\equiv 1 \bmod m$. □

THEOREM 4.2. *Given integers $x_1, \cdots, x_n$ and $p^l \leqq n$ a prime power where $0 \leqq x_1, \cdots, x_n < p^l$ the product $\prod_{i=1}^{n} x_i \bmod p^l$ can be computed in NC^1.*

Proof. It is a known fact of number theory (e.g. [6]) that $\mathbf{Z}_{p^l}^*$ is cyclic except when $p = 2$ and $l > 2$, in which case $\mathbf{Z}_{p^l}^*$ is generated by 5 and $2^l - 1$. The basic idea of the algorithm is to hardwire in a table of discrete logarithms for each prime power $< n$ and then reduce the problem to one of computing iterated addition. In $O(\log n)$ space it is possible to factor any number $\leqq n$ and so determine whether it is a prime power.

For each $p^l \leqq n$ ($p \neq 2$ or $l \leqq 2$) in $O(\log n)$ space one can find a generator g for $\mathbf{Z}_{p^l}^*$ by brute force and then compute all powers of g up to $p^l - p^{l-1}$, the order of $\mathbf{Z}_{p^l}^*$, and hardwire them into the circuit. For each $4 < 2^l \leqq n$ in $O(\log n)$ space one can compute $(-1)^a 5^b \bmod 2^l$ for $a = 0, 1$ and $0 \leqq b < 2^{l-2}$ and hardwire them into the circuit. These tables may be used in either direction as tables of powers or of discrete logarithms.

The algorithm then proceeds as follows:

(1) Compute the largest power, j_i, of p which divides x_i for $i = 1, \cdots, n$ in parallel.
(2) Compute $y_i = x_i / p^{j_i}$ for $i = 1, \cdots, n$.
(3) Compute $j = \sum_{i=1}^n j_i$. Note that the y_i are now in $\mathbf{Z}_{p^l}^*$ and $\prod_{i=1}^n x_i \equiv p^j \prod_{i=1}^n y_i \bmod p^l$.
(4) Test if $p \neq 2$ or $p^l = 2, 4$. If either condition holds, perform A else perform B.

Part A

(5) Find each y_i in the table for p^l and read off its discrete logarithm, a_i.
(6) Compute $a = \sum_{i=1}^n a_i$.
(7) Compute $\bar{a} = a \bmod (p^l - p^{l-1})$.
(8) Read off $\prod_{i=1}^n y_i = g^{\bar{a}} \bmod p^l$ from the table.

Part B

(5) Find each y_i in the table for 2^l and read off its representation as powers of $2^l - 1$ and 5, a_i and b_i.
(6) Compute $a = \sum_{i=1}^n a_i$ and $b = \sum_{i=1}^n b_i$.
(7) Compute $\bar{a} = a \bmod 2$ and $\bar{b} = b \bmod 2^{l-2}$.
(8) Read off $\prod_{i=1}^n y_i = (-1)^{\bar{a}} 5^{\bar{b}} \bmod 2^l$ from the table.

(9) Compute $\prod_{i=1}^n x_i = p^j \prod_{i=1}^n y_i \bmod p^l$.

The table look-ups can be computed in $O(\log n)$ depth using selector trees, the modulo operations are computed as in Lemma 4.1, and the other steps can be computed using fast iterated addition circuits [11] in $O(\log n)$ depth. □

McKenzie and Cook also show how the above circuits may be used to compute iterated products for any small modulus by Chinese remaindering. It is interesting to note the following:

THEOREM 4.3. *For n-bit integers a and b, computing* $a^b \bmod m$ *where* $m \leqq n$ *is in* NC^1.

Proof sketch. By Chinese remaindering the problem can be reduced to computing $a^b \bmod p^l$, for each prime power factor p^l dividing m. This is solved by the same technique as above, taking discrete logarithms, multiplying by b, and then exponentiating mod p^l. □

5. Log depth circuits for division and iterated product. Let $x_1, \cdots, x_n$ be n bit positive integers. The ITERATED PRODUCT problem is to compute $\prod_{i=1}^n x_i$. It is clear that POWERING is reducible to ITERATED PRODUCT (it is little more than a special case) and so POWERING and DIVISION will be computable in small depth if we can find small depth circuits for ITERATED PRODUCT. In order to solve this problem we will make use of Chinese remaindering and the circuits for arithmetic operations modulo small integers.

The Chinese remainder theorem yields a process for determining, given the values of an integer modulo a sequence of relatively prime numbers, the result of taking that integer modulo their product. More formally the CHINESE REMAINDERING problem for pairwise relatively prime integers $c_1, \cdots, c_n$ is: given inputs $c_1, \cdots, c_n$, and $x \bmod c_1, \cdots, x \bmod c_n$, compute $x \bmod \prod_{j=1}^n c_j$.

LEMMA 5.1. CHINESE REMAINDERING *for pairwise relatively prime* $c_1, \cdots, c_n$ *where* $1 < c_1 < \cdots < c_n \leqq n^2$ *is* NC^1 *reducible to the problem of computing* $c = \prod_{i=1}^n c_i$.

Proof. The circuit performs:

(1) Call the oracle to obtain $c = \prod_{i=1}^{n} c_i$.
(2) Compute $\nu_i = \prod_{j \neq i} c_j$ by dividing c by c_i (by Lemma 4.1) for $i = 1, \cdots, n$ in parallel.
(3) Solve $\nu_i w_i \equiv 1 \pmod{c_i}$ for $w_1, \cdots, w_n$ in parallel.
(4) Compute the interpolation constants, $u_i = \nu_i w_i$ for $i = 1, \cdots, n$. Note that $u_i \equiv 1 \bmod c_i$ and $u_i \equiv 0 \bmod c_j$ for $j \neq i$.
(5) Compute $y = \sum_{i=1}^{n} (x \bmod c_i) u_i$ by multiplying in parallel and then computing a series sum. It is necessary to reduce y modulo c to obtain the desired result. The largest multiple of c which is less than y can be estimated since

$$y = \sum_{i=1}^{n} (x \bmod c_i)\nu_i w_i = \sum_{i=1}^{n} \frac{(x \bmod c_i) w_i}{c_i} c.$$

(6) Compute $r_i = m(x \bmod c_i) w_i$ for $i = 1, \cdots, n$ where $m = 2^{\lceil \log_2 n \rceil}$. Thus $y = \sum_{i=1}^{n} (r_i / m c_i) c$.
(7) Compute $s_i = \lfloor r_i / c_i \rfloor$ for $i = 1, \cdots, n$.
(8) Compute $s = \sum_{i=1}^{n} s_i$.
(9) Compute $t = \lfloor s/m \rfloor$, i.e. truncate the right $\lceil \log_2 n \rceil$ bits of s. Note that $0 \leqq y - (s/m)c = \sum_{i=1}^{n} ((r_i / m c_i) - (1/m)\lfloor r_i / c_i \rfloor) c < \sum_{i=1}^{n} (1/m) c \leqq c$. Thus $0 \leqq y - tc < 2c$.
(10) Set $x \bmod c$ to $y - tc$ if $0 \leqq y - tc < c$; otherwise set it to $y - (t+1)c$.

Since each c_i is small, representable in $O(\log n)$ bits, step (2) may be computed in depth $O(\log n)$; similarly step (3) can be computed by brute force. Steps (4), (5), (6), and (8) can be computed by multiplying in parallel and then using multiple addition. Step (7) involves divisions of $O(\log n)$-bit integers by $O(\log n)$-bit integers and can be done using any reasonable division circuit (even linear depth would not hurt here). Step (10) requires simple multiplication, comparison and subtraction in parallel. Each of these steps is of depth $O(\log n)$. □

If we can compute $\prod_{i=1}^{n} x_i \bmod c_j$ for a set of relatively prime $c_1, \cdots, c_s$ such that $\prod_{j=1}^{s} c_j > \prod_{i=1}^{n} x_i$, then the result of the interpolation process of Chinese remaindering will give the value of $\prod_{i=1}^{n} x_i$ exactly. This fact and the above lemma motivate the following definition.

DEFINITION. A sequence $M_1, M_2, \cdots$ is a *good modulus sequence* if and only if there are polynomials $q(n)$ and $r(n)$ such that for all n:

(i) $2^n \leqq M_n \leqq 2^{q(n)}$.

(ii) For any prime p, $p^l \mid M_n$ implies that $p^l \leqq r(n)$.

THEOREM 5.2. ITERATED PRODUCT *is* NC^1 *reducible to the problem of computing any good modulus sequence* $\{M_n\}$.

Proof. From the definition of good modulus sequence it is clear that $M_{n^2} \geqq 2^{n^2} > \prod_{i=1}^{n} x_n$.

We obtain the following algorithm:

(1) Call the good modulus sequence oracle to obtain M_{n^2}.
(2) Factor M_{n^2} to obtain prime power factors $c_i = p_i^{l_i}$ for $i = 1, \cdots, s$.
(3) Compute in parallel $b_{ij} = x_i \bmod c_j$ for $i = 1, \cdots, n$ and $j = 1, \cdots, s$.
(4) Compute $b_j = \prod_{i=1}^{n} b_{ij} \bmod c_j$ for $j = 1, \cdots, s$. Note that $b_j = \prod_{i=1}^{n} x_i \bmod c_j$.
(5) Compute $\prod_{i=1}^{n} x_i \bmod M_{n^2}$ using the Chinese remaindering circuit for $c_1, \cdots, c_s$ to obtain the iterated product exactly.

Step (2) is brute force because the prime power factors are small and step (3) follows from Lemma 4.1. Using Theorem 4.2 for step (4) the entire circuit has depth $O(\log n)$. □

The computational problem is now reduced to finding a good modulus sequence efficiently. The next theorem shows how this can be done.

THEOREM 5.3. ITERATED PRODUCT *is computable by P-uniform Boolean circuits of depth* $O(\log n)$.

Proof. In polynomial time we can find the first n primes, $p_1, \cdots, p_n$ and compute their product. By the prime number theorem, $p_n = O(n \log n)$, so $\prod_{i=1}^{n} p_i = 2^{O(n \log n)}$. Also trivially $2^n \leqq \prod_{i=1}^{n} p_i$. Thus $\prod_{i=1}^{n} p_i$ for $n = 1, 2, \cdots$ forms a good modulus sequence. We can compute this good modulus sequence in polynomial time, hardwire the values into the circuit and then apply Theorem 5.2 to get the desired result. □

Using the previous reductions, we have:

COROLLARY 5.4. DIVISION *and* POWERING *are computable by P-uniform Boolean circuits of depth* $O(\log n)$.

6. Iterated product and powering are equivalent. As was previously stated POWERING is easily NC^1 reducible to ITERATED PRODUCT but the reducibility in reverse is far from obvious.

THEOREM 6.1. ITERATED PRODUCT *is* NC^1 *reducible to* POWERING.

Proof. We use the reduction of ITERATED PRODUCT to computing a good modulus sequence.

The algorithm proceeds as follows:

(1) Set $x = 2^{2n} + 1$.
(2) Use the powering circuit to compute $y = x^{2n}$. Note that $y = \sum_{i=0}^{2n} \binom{2n}{i} 2^{2ni}$.
(3) Read off $\binom{2n}{n}$ as bits in positions $2n^2$ to $2n^2 + 2n - 1$ from the right in y (position 0 contains the low order bit). Note that $2^{2n} > \binom{2n}{n} \geqq 2^n$.

By elementary arithmetic (e.g. [6]) the exponent of the largest power of prime p dividing $n!$ is $\sum_{i>0} \lfloor n/p^i \rfloor$. Thus the largest power dividing $\binom{2n}{n}$ is

$$\sum_{i>0} \left\{ \left\lfloor \frac{2n}{p^i} \right\rfloor - 2 \left\lfloor \frac{n}{p^i} \right\rfloor \right\}.$$

Now each of these terms is $\leqq 1$ and the terms vanish when $p^i > 2n$ so that the largest power p^l dividing $\binom{2n}{n}$ satisfies $p^l < 2n$. From this we see that $\binom{2n}{n}$ for $n = 1, 2, \cdots$ forms a good modulus sequence and so the reduction is correct. □

COROLLARY 6.2. DIVISION, POWERING, *and* ITERATED PRODUCT *are all* NC^1 *equivalent.*

7. Divisibility. Although the DIVISION problem has P-uniform $O(\log n)$ depth circuits, it is still unclear whether or not it has log-space uniform $O(\log n)$ depth circuits. Despite the fact that we are unable to answer this question it is possible to find such circuits for a closely related problem, DIVISIBILITY.

Let x, y be n-bit integers. The output of the DIVISIBILITY problem is 1 if $y|x$, 0 otherwise.

THEOREM 7.1. DIVISIBILITY *is in* NC^1, *and hence has deterministic space complexity* $O(\log n)$.

Proof. For each of n primes $p_1 < \cdots < p_n$ not dividing y we can solve $yz \equiv x \bmod p_i$ to obtain z_i. If we could compute $M = \prod_{i=1}^{n} p_i$ then, as in Lemma 5.1, we could find the unique z such that $0 \leqq x < M$ and $z \equiv z_i \bmod p_i$ for each i. Such a z would be the

only possible candidate for a solution to $yz = x$. If

$$u_i \equiv \begin{cases} 1 \bmod p_i, \\ 0 \bmod p_j, \end{cases} \qquad j \neq i,$$

then $z \equiv \sum_{i=1}^{n} u_i z_i \bmod M$. If, in addition, $0 \leqq u_i < M$ then $z = z^{(t)}$ for some t, $0 \leqq t < np_n$ where $z^{(t)} = \sum_{i=1}^{n} u_i z_i - tM$. It follows that $y|x$ if and only if $\exists t$, $0 \leqq t < np_n$ such that $yz^{(t)} = x$. It is not necessary, however, to compute $z^{(t)}$ explicitly. We merely need to test the condition modulo sufficiently many primes. Since for any t, $|yz^{(t)} - x| < np_n 2^{n+1} M$, it suffices for the product of these primes to exceed $np_n 2^{n+1} M$. Note that the equation always holds modulo each of the primes $p_1, \cdots, p_n$, so that it suffices to choose additional primes whose product exceeds $np_n 2^{n+1}$.

The resulting algorithm is:

(1) Find the first $3n$ primes.
(2) Compute $y_i = y \bmod p_i$ for each of these primes.
(3) Select the first n primes from those found in (1) such that $y_i \neq 0$. Note that since $y \leqq 2^n$ it cannot have more than n different prime factors. In the remainder of the algorithm we designate these primes as $p_1, \cdots, p_n$ and the remaining primes among the first $3n$ as $q_1, \cdots, q_{2n}$.
(4) Compute $z_i = xy_i^{-1} \bmod p_i$ for each $i = 1, \cdots, n$.
(5) Compute $M_k = \prod_{i=1}^{n} p_i \bmod q_k$ for each $k = 1, \cdots, 2n$. Note that $M_k \equiv M \bmod q_k$.
(6) Compute $\nu_{ik} = \prod_{j \neq i} p_j \bmod q_k$ $(= M_k p_i^{-1} \bmod q_k)$ for each $i = 1, \cdots, n$ and $k = 1, \cdots, 2n$.
(7) Compute $w_i = \prod_{j \neq i} p_j^{-1} \bmod p_i$ for each $i = 1, \cdots, n$.
(8) Compute $w_{ik} = w_i \bmod q_k$ for each $i = 1, \cdots, n$ and $k = 1, \cdots, 2n$.
(9) Compute $u_{ik} = \nu_{ik} w_{ik} \bmod q_k$ for each $i = 1, \cdots, n$ and $k = 1, \cdots, 2n$. Note that $u_{ik} \equiv u_i \bmod q_k$.
(10) Compute $z_{ik} = z_i \bmod q_k$ for each $i = 1, \cdots, n$ and $k = 1, \cdots, 2n$.
(11) Compute $z_k^{(t)} = \sum_{i=1}^{n} u_{ik} z_{ik} - tM_k \bmod q_k$ for each $k = 1, \cdots, 2n$ and $t = 1, \cdots, np_n$.
(12) Check if there exists a t such that for all k, $y_k z_k^{(t)} \equiv x \bmod q_k$. If such a t exists output 1 else output 0.

All the operations are computed modulo small primes in $O(\log n)$ depth and the remaining computations are simple tests in parallel which also have $O(\log n)$ depth. □

8. *P*-uniform size bounds. In obtaining the $O(\log n)$ depth circuits for the problems of the previous sections we have avoided using the full power of P-uniformity as much as possible. This permitted us to focus on constructing "good modulus sequences" in attempting to produce log-space uniform circuits for these problems. However, this has made our circuits larger than necessary.

By making fuller use of polynomial time constructibility, the $O(\log n)$ depth circuits can be simplified somewhat, yielding a reduction in their size. Since this simplification is most dramatic for the POWERING circuits, we describe this case in detail.

THEOREM 8.1. POWERING *can be computed by P-uniform circuits of depth* $O(\log n)$ *and size* $O(n^5 \log^2 n)$.

Proof. Precompute and hardwire into the circuit:

(a) Primes $p_1, \cdots, p_s$ such that $\prod_{j=1}^{s} p_j > 2^{n^2}$.
(b) The $n+1$-bit under-approximations of the inverses of p_j, $\bar{p}_j^{-1}$, for $j = 1, \cdots, s$.

(c) For $i = 1, \cdots, n$, $j = 1, \cdots, s$, tables of $a^i \bmod p_j$ for each a, $0 \leqq a < p_j$.
(d) $M = \prod_{i=1}^{s} p_j$.
(e) The $2n^2$-bit under-approximation of the inverse of M, $\bar{M}^{-1}$.
(f) Interpolation constants $0 \leqq u_j < M$ such that

$$u_j \equiv \begin{cases} 1 \bmod p_j, \\ 0 \bmod p_i, \end{cases} \quad i \neq j \quad \text{for } j = 1, \cdots, s.$$

The circuit performs:

(1) Compute $t_j = x\bar{p}_j^{-1}$ and truncate to obtain $\lfloor t_j \rfloor$ for $j = 1, \cdots, s$.
(2) Compute $x \bmod p_j$ for $j = 1, \cdots, n$ as either $z_j = x - p_j \lfloor t_j \rfloor$ or $z_j - p_j$ whichever is between 0 and p_j.
(3) Read $y_{ij} = x^i \bmod p_j$ from the tables for $i = 1, \cdots, n$ and $j = 1, \cdots, s$.
(4) Compute $y_i = \sum_{j=1}^{s} y_{ij} u_j$ for $i = 1, \cdots, n$.
(5) Compute $t_i' = y_i \bar{M}^{-1}$ and truncate to obtain $\lfloor t_i' \rfloor$ for $i = 1, \cdots, n$.
(6) Compute x^i for $i = 1, \cdots, n$ as either $z_i' = y_i - M \lfloor t_i' \rfloor$ or $z_i' - M$ whichever is between 0 and M.

Certainly $s < n^2$ and thus by the Prime Number Theorem $p_s = O(n^2 \log n)$. Since multiplication of n-bit integers can be done in size $O(n \log n \log\log n)$ [11] and $O(\log n)$ depth, steps (1) and (2) can be computed in size $O(n^3 \log n \log\log n)$ as can steps (5) and (6). Step (4) computes n sums of $O(n^2)$ terms which are each multiplications of $O(\log n)$-bit integers by $O(n^2)$-bit integers. The multiplications in (4) cost a total of $O(n^5 \log n)$ size and the summations cost $O(n^5)$ in size. The table look-ups in step (3) are computed separately for each value of i and for each value of j. It follows that the circuit size for step (3) is dominated by the size of the tables, which contain $O(nsp_s) = O(n^5 \log n)$ entries of $O(\log n)$ bits each. Thus the total size of the circuit is $O(n^5 \log^2 n)$, as stated. □

The circuits for ITERATED PRODUCT are somewhat more complicated but are still the same size as the POWERING circuits.

THEOREM 8.2. ITERATED PRODUCT *can be computed by P-uniform circuits of depth* $O(\log n)$ *and size* $O(n^5 \log^2 n)$.

Proof sketch. These circuits are similar to the circuits in Theorem 8.1 above except for the following:

(i) The steps corresponding to (1) and (2) are performed on each x_i for $i = 1, \cdots, n$.

(ii) The steps corresponding to (4), (5) and (6) are performed so as to compute only one n^2-bit output.

(iii) The table look-ups in step (3) are replaced by circuits with tables of discrete logarithms for each p_j like the ones described in steps (5)-(8) of Part A in Theorem 4.2.

Applying changes (i) and (ii) to the arguments in Theorem 8.1, those portions of the circuit may be computed in size $O(n^4 \log n \log\log n)$ and $O(\log n)$ depth. The tables of discrete logarithms required by this circuit have only $O(sp_s)$ entries of $O(\log n)$ bits each, for a total size of $O(n^4 \log^2 n)$. However, the table for each p_j is accessed n different times in parallel (once for each x_i) so the accessing hardware is of size $O(n^5 \log^2 n)$. The other computations are easily within this size bound and the theorem follows. □

DIVISION circuits may be constructed, using the reduction circuits of Theorem 3.1 and the POWERING circuits in Theorem 8.1, of the same asymptotic size as those for POWERING. By applying a trick suggested by Reif we obtain smaller, although more complicated, DIVISION circuits.

THEOREM 8.3. DIVISION *can be computed by P-uniform circuits of depth* $O(\log n)$ *and size* $O(n^4 \log^3 n)$.

Proof sketch. In computing DIVISION, the hard part is computing the series, $1+u+u^2+\cdots$, to at least n terms for a scaled n-bit u. Instead of computing each term separately, apply the identity

$$\prod_{i=0}^{k} (1+u^{2^i}) = 1+u+u^2+\cdots+u^{2^{k+1}}-1$$

with $k = \lfloor \log_2 n \rfloor$. The circuit then computes only $O(\log n)$ powers from the POWERING circuit, adds 1 to each of the scaled results, and then computes the ITERATED PRODUCT of the $O(\log n)$ resulting terms. The powering portion of the circuit uses only $O(n^4 \log^3 n)$ size because there are fewer powers to be computed, resulting in smaller tables, and the iterated product portion of the circuit needs fewer simultaneous table accesses and also uses size $O(n^4 \log^3 n)$. □

9. Summary and open problems. From the $O(\log n)$ depth P-uniform circuits for the problems presented here, using the results of Alt [1], a large class of natural problems can now be shown to have $O(\log n)$ depth circuits. It is unknown whether any of these circuits may be made log-space uniform, which would imply that the problems are computable in deterministic log space.

An interesting problem related to powering is the conversion of integers from one fixed base to another, e.g. base 3 to base 5, when the digits of integers in bases other than 2 are represented by groups of bits. This problem can easily be seen to have P-uniform $O(\log n)$ depth circuits even without the machinery presented here. In our example all that is required is to precompute $3^0, \cdots, 3^{n-1}$ in base 5, hardwire them into the circuit, and on input $(b_{n-1} \cdots b_0)_3$ compute $\sum_{i=0}^{n-1} \sum_{j=1}^{b_i} 3^i$ in base 5. The base 5 summation circuits are simple modifications of standard fast binary summation circuits [11]. It is an open question whether this problem, which is reducible to powering for a fixed base, has $O(\log n)$ depth log-space uniform circuits when one base is not a power of the other.

The class of problems which are reducible to the decision problem, DIVISIBILITY, may be worth investigating since our results imply that such problems would have log-space uniform $O(\log n)$ depth Boolean circuits.

Finally, there is a stronger and in some ways more natural definition of uniform than log-space uniform. This stronger form was introduced by Ruzzo [10] and called U_{E^*}-uniform (see [4]). If this condition is used to define NC^1, then NC^1 can be characterized simply as the class of problems computable in time $O(\log n)$ on an alternating Turing machine. Unfortunately, it is not clear whether all the results shown here still hold with the stronger condition. In particular, it would be interesting to know whether DIVISIBILITY, iterated product modulo small prime powers, and the reduction of iterated product to powering, have NC^1 circuits in this stronger sense.

REFERENCES

[1] H. ALT, *Comparison of arithmetic functions with respect to Boolean circuit depth*, Proc. 16th ACM Symposium on Theory of Computing (1984), pp. 466–470.

[2] H. ALT AND N. BLUM, *On the Boolean circuit depth of division related functions*, Dept. Computer Science, Pennsylvania State University, State College, PA, 1983.

[3] A. BORODIN, *On relating time and space to size and depth*, this Journal, 6 (1977), pp. 733–744.

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

[5] H. J. HOOVER, *Some topics in circuit complexity*, M.Sc. thesis and TR-139/80, Dept. Computer Science, University of Toronto, 1979.
[6] L. K. HUA, *Introduction to Number Theory*, Springer-Verlag, New York, 1982.
[7] P. MCKENZIE AND S. A. COOK, *The parallel complexity of Abelian permutation group problems*, TR-181/85, Dept. Computer Science, University of Toronto, 1985.
[8] J. REIF, *Logarithmic depth circuits for algebraic functions*, Proc. 24th IEEE Symposium on Foundations of Computer Science (1983), pp. 138–145.
[9] ———, *Logarithmic depth circuits for algebraic functions*, this Journal, 15 (1986), pp. 231–242.
[10] W. L. RUZZO, *On uniform circuit complexity*, J. Comput. System Sci., 22 (1981), pp. 365–383.
[11] J. E. SAVAGE, *The Complexity of Computing*, John Wiley, New York, 1976.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
009

ON THE VALIDITY OF THE DIRECT SUM CONJECTURE*

JOSEPH JA'JA'† AND JEAN TAKCHE‡

Abstract. The direct sum conjecture states that the multiplicative complexity of disjoint sets of bilinear computations is the sum of their separate multiplicative complexities. This conjecture is known to hold for only a few specialized cases. In this paper, we establish its validity for large classes of computations. One such class can be defined as follows. Let S_1 be a set of r $m \times n$ bilinear forms, and let S_2 be a different set of s $p \times q$ bilinear forms. Then, if $2 \in \{r, m, n, s, p, q\}$, we show that the direct sum conjecture holds over any field. The proof involves some nontrivial facts from linear algebra and relies on the theory of invariant polynomials. This result also settles the multiplicative complexity of pairs of bilinear forms over any field with large enough cardinality. It is also shown that the direct sum conjecture is true for the case when $r = mn - 2$.

Key words. algebraic complexity, bilinear forms, direct sum conjecture

AMS(MOS) subject classification. 68C

1. Introduction. Suppose that two sets of bilinear forms $S_1 = \{B_i = x^T G_i y\colon 1 \leqq i \leqq r, \dim x = m, \dim y = n\}$ and $S_2 = \{\bar{B}_j = \bar{x}^T \bar{G}_j \bar{y}\colon 1 \leqq j \leqq s, \dim \bar{x} = p, \dim \bar{y} = q\}$ are to be computed such that the indeterminates x and y are respectively disjoint from $\bar{x}$ and $\bar{y}$. The *direct sum* of S_1 and S_2, denoted by $S_1 \oplus S_2$, is the set of bilinear forms $\{B_i\} \cup \{\bar{B}_j\}$ with indeterminates $\{x_i\}_{1 \leqq i \leqq m} \cup \{\bar{x}_k\}_{1 \leqq k \leqq p}$ and $\{y_1\}_{1 \leqq 1 \leqq n} \cup \{\bar{y}_u\}_{1 \leqq u \leqq q}$. Let $\delta\{S_k\}$ denote the multiplicative complexity of S_k, $k = 1, 2$. It is obvious that $\delta(S_1 + S_2) \leqq \delta(S_1) + \delta(S_2)$, over any field since one may combine optimal algorithms of S_1 and S_2 into an algorithm for $S_1 \oplus S_2$. The *direct sum conjecture*, originally due to Strassen [St], states that $\delta(S_1 \oplus S_2) = \delta(S_1) + \delta(S_2)$, which essentially means that disjoint bilinear problems can be optimally solved separately. Since then, few facts have been established about this conjecture.

Using a class of algorithms called approximation algorithms introduced by [B et al.], Pan [P] and Schönhage [S] showed that there exist certain pairs of matrix multiplication problems with completely disjoint sets of variables which can be evaluated *faster* in one compound computation than separately. Therefore, the direct sum conjecture is not true for approximation algorithms. This also implies that the conjecture for exact algorithms does not hold over rings with divisors of zeros but leaves open its validity over fields.

On the other hand, the validity of the direct sum conjecture over fields has been established for a few very specialized cases [AW], [FW].

In this paper, we show that if $2 \in \{r, m, n, s, p, q\}$, then the direct sum conjecture holds over any field F. This is a large class of bilinear computations since any of five parameters could be arbitrary. Moreover, the proof relies on nontrivial facts from Kronecker's theory of pencils [G]. We extend several results of linear algebra and strengthen the linear combination techniques to establish the lower bound. The same techniques could be used to establish the multiplicative complexity of pairs of bilinear forms over any field, proving a conjecture stated in [J]. On the other hand, we also show that the direct sum conjecture is true if $r = mn - 2$ (or equivalently, $s = pq - 2$).

* Received by the editors January 22, 1985, and in revised form July 18, 1985. Supported in part by the National Science Foundation under grant MCS-83-15890.

† Electrical Engineering Department, University of Maryland, College Park, Maryland 20742.

‡ Department of Mathematics, Penn State University, University Park, Pennsylvania 16802.

We start, in § 2, by proving several preliminary results which will be used in § 3 to establish the direct sum conjecture for the first class. Section 4 is devoted to the solution of the direct sum conjecture for the case when $r = mn-2$.

2. Preliminary results. Several technical results will be shown that will play a crucial role in the proof of the next section. Most of these facts are extensions of well-known results in linear algebra whose proofs are interesting on their own.

We start by introducing some terminology [G]. Let G be an $n \times n$ matrix over a field F. The *invariant polynomials* $p_t(\lambda)$ are defined by

$$p_t(\lambda) = \frac{d_{n-t+1}(\lambda I - G)}{d_{n-t}(\lambda I - G)},$$

where $d_t(I-G)$ is the greatest common divisor of all minors of $\lambda I - G$ of order t, $1 \leqq t \leqq k$, $(p_{k+1}(\lambda) = \cdots = p_n(\lambda) = 1)$. Notice that $p_k(\lambda) | p_{k-1}(\lambda) | \cdots | p_1(\lambda)$. Finally, if $p(\lambda) = \alpha_0 + \alpha_1\lambda + \cdots + \cdots + \alpha_{n-1}\lambda^{n-1} + \lambda^n$ is a monic polynomial, the *companion matrix* of $p(\lambda)$, denoted by $C(p(\lambda))$, is given by

$$C(p(\lambda)) = \begin{bmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \\ -\alpha_0 & -\alpha_1 & & \cdots & -\alpha_{n-1} \end{bmatrix}.$$

We are now ready for the first lemma.

LEMMA 2.1. *Let $p(\lambda) = \alpha_0 + \alpha_1\lambda + \cdots + \alpha_{m-1}\lambda^{m-1} + \lambda^m$ be an irreducible polynomial over a field F and let $C = C(p(\lambda))$ be the corresponding companion matrix. If $q(\lambda) = \beta_0 + \beta_1\lambda + \cdots + \beta_{r-1}\lambda^{r-1} + \lambda^r \neq 0$ is a polynomial over F of degree r less than m, then* $\det(q(C)) \neq 0$.

Proof. If $r = 0$, then $\det(q(C)) = \det(I) = 1 \neq 0$. Hence we can assume that $r > 0$. The proof will be by contradiction. Assume that $\det(q(C)) = 0$. Then there exists a nonzero vector v such that $q(C)v = 0$. Let $\bar{F}$ be the splitting field of $q(\lambda)$. Then $q(\lambda) = \pm(\lambda_1 - \lambda)(\lambda_2 - \lambda)\cdots(\lambda_r - \lambda)$, where $\lambda_i \in \bar{F}$, $1 \leqq i \leqq r$. We thus have $(\lambda_r I - C) \times (\lambda_{r-1} I - C) \cdots (\lambda_1 I - C) \cdot v = 0$. Let r' be the smallest index i such that $(\lambda_i I - C) \times (\lambda_{i-1} I - C) \cdots (\lambda_1 I - C)v = 0$. It follows that $(\lambda_{r'} I - C)(\lambda_{r'-1} I - C) \cdots (\lambda_1 I - C)v = 0$ and $(\lambda_{r'-1} I - C)(\lambda_{r'-2} I - C) \cdots (\lambda_1 I - C)v = \bar{v} \neq 0$. Hence $(\lambda_{r'} I - C)\bar{v} = 0$ which implies that $\lambda_{r'}$ is an eigenvalue of C. Thus $\det(\lambda_{r'} I - C) = 0$. This implies that $p(\lambda_{r'}) = 0$. But $p(\lambda)$ is irreducible over F, i.e. $p(\lambda)$ is the minimal polynomial of $\lambda_{r'}$. Since $q(\lambda_{r'}) = 0$, $p(\lambda)$ must divide $q(\lambda)$ which contradicts the fact that $\deg q(\lambda) < \deg p(\lambda)$. Therefore $\det(q(C)) \neq 0$. □

A *generalized Jordan form* is given by the next lemma.

LEMMA 2.2. *Given a square matrix G with the following set of invariant polynomials:*

$$p_1(\lambda) = q_1^{\tau_{11}} q_2^{\tau_{12}} \cdots q_r^{\tau_{1r}},$$

$$p_2(\lambda) = q_1^{\tau_{21}} q_2^{\tau_{22}} \cdots q_r^{\tau_{2r}},$$

$$p_k(\lambda) = q_1^{\tau_{k1}} q_2^{\tau_{k2}} \cdots q_r^{\tau_{kr}},$$

where $0 \leqq \tau_{kj} \leqq \tau_{k-1,j} \leqq \cdots \leqq \tau_{1j}$, $1 \leqq j \leqq r$. *Then* G *is similar to the matrix*

$$A = \begin{bmatrix} \begin{matrix} C(q_1) & H_1 & & \\ & C(q_1) & \ddots & H_1 \\ & & \ddots & C(q_1) \end{matrix} & & & & \\ & \begin{matrix} C(q_r) & H_r & & \\ & C(q_r) & \ddots & H_r \\ & & \ddots & C(q_r) \end{matrix} & & & \\ & & \begin{matrix} C(q_1) & H_1 & & \\ & C(q_1) & \ddots & H_1 \\ & & \ddots & C(q_1) \end{matrix} & & \\ & & & \begin{matrix} C(q_r) & H_r & & \\ & C(q_r) & \ddots & H_r \\ & & \ddots & C(q_r) \end{matrix} \end{bmatrix}$$

where

$$H_i = \left[\begin{array}{c|c} 0 & 0 \\ \hline 1 & 0 \end{array}\right]$$

with appropriate dimensions and $C(q_i)$ *is the companion matrix of* q_i. *In fact* A *and* G *have the same invariant polynomials.*

Proof. Straightforward (see [G]).

LEMMA 2.3. *Let* $p(\lambda) = \alpha_0 + \alpha_1\lambda + \cdots + \alpha_{m-1}\lambda^{m-1} + \lambda^m \in F[\lambda]$ *be a polynomial which has no roots in* F. *Let* $C(p(\lambda))$ *be its companion matrix, and let* C_i *be the* i*th column of* $I_m s_1 + C(p(\lambda))s_2$, *where* s_1 *and* s_2 *are indeterminates. Then the* m *rows of* $\bar{C} = [C_2 + \beta_1 C_1 \;\; C_3 + \beta_2 C_1 \;\; \cdots \;\; C_m + \beta_{m-1} C_1]$ *are linearly independent for any set of* β_i*'s in* F.

Proof. It is clear that

$$I_m s_1 + C(p(\lambda))s_2 = \begin{bmatrix} s_1 & s_2 & & & \\ & & & 0 & \\ & s_1 & \ddots & & \\ 0 & & & s_2 & \\ & & \ddots & & \\ & & & s_1 & s_2 \\ -\alpha_0 s_2 & -\alpha_1 s_2 & \cdots & -\alpha_{m-2} s_2 & s_1 - \alpha_{m-1} s_2 \end{bmatrix}$$

and

$$\bar{C} = \begin{bmatrix} s_2 + B_1 s_1 & \beta_2 s_1 & \cdots & \beta_{m-1} s_1 \\ s_1 & s_2 & & \\ & s_1 & \ddots & \\ & & \ddots & s_2 \\ (-\alpha_1 - \beta_1\alpha_0)s_2 & (-\alpha_2 - \beta_2\alpha_0)s_2 & \cdots & s_1 + (-\alpha_{m-1} - \beta_{m-1}\alpha_0)s_2 \end{bmatrix}.$$

The number of linearly independent rows does not change by performing elementary row operations on $\bar{C}$. Let R_i be the ith row of $\bar{C}$. Multiply R_{i+1} by $-\beta_i$ and add it to

the 1st row R_1, we get

$$\bar{C} = \begin{bmatrix} (1+\beta_{m-1}(\alpha_1+\beta_1\alpha_0))s_2 & (-\beta_1+\beta_{m-1}(\alpha_2+\beta_2\alpha_0))s_2 & \cdots & (-\beta_{m-2}+\beta_{m-1}(\alpha_{m-1}+\beta_{m-1}\alpha_0))s_2 \\ s_1 & s_2 & & \\ & s_1 & \ddots & \vdots \\ & & \ddots & s_2 \\ (-\alpha_1-\beta_1\alpha_0)s_2 & (-\alpha_2-\beta_2\alpha_0)s_2 & \cdots & s_1+(-\alpha_{m-1}-\beta_{m-1}\alpha_0)s_2 \end{bmatrix}.$$

Claim. The first row of $\bar{C}$ is not identically equal to zero.

Proof of Claim. Assume that the 1st row is equal to zero. Then

$$1+\beta_{m-1}(\alpha_1+\beta_1\alpha_0) = -\beta_1+\beta_{m-1}(\alpha_2+\beta_2\alpha_0) = \cdots = -\beta_{m-2}+\beta_{m-1}(\alpha_{m-1}+\beta_{m-1}\alpha_0) = 0.$$

Let $x = \beta_{m-1}\alpha_0$. Notice that $\alpha_0 \neq 0$ because $p(\lambda)$ has no roots in F. Therefore we can multiply all equations by α_0. We obtain

$$\alpha_0 + x(\alpha_1+\beta_1\alpha_0) = -\alpha_0\beta_1 + x(\alpha_2+\beta_2\alpha_0) = \cdots = -\alpha_0\beta_{m-2} + x(\alpha_{m-1}+x) = 0.$$

We thus have:

$$\begin{aligned}
&-\alpha_0\beta_{m-2} + x(\alpha_{m-1}+x) = 0 \Rightarrow \alpha_0\beta_{m-2} = \alpha_{m-1}x + x^2, \\
&-\alpha_0\beta_{m-3} + x(\alpha_{m-2}+\beta_{m-2}\alpha_0) = 0 \Rightarrow \alpha_0\beta_{m-3} = \alpha_{m-2}x + x(\alpha_{m-1}x + x^2) \\
&\qquad\qquad = \alpha_{m-2}x + \alpha_{m-1}x^2 + x^3, \\
&\quad\vdots \\
&-\alpha_0\beta_1 + x(\alpha_2+\beta_2\alpha_0) = 0 \Rightarrow \alpha_0\beta_1 = \alpha_2 x + \alpha_3 x^2 + \cdots + \alpha_{m-1}x^{m-2} + x^{m-1}, \\
&\alpha_0 + x(\alpha_1+\beta_1\alpha_0) = 0 \Rightarrow \alpha_0 + \alpha_1 x + x(\alpha_2 x + \alpha_3 x^2 + \cdots + \alpha_{m-1}x^{m-2} + x^{m-1}) = 0 \\
&\qquad\qquad \Rightarrow \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_{m-1}x^{m-1} + x^m = 0.
\end{aligned}$$

Therefore $x = \beta_{m-1}\alpha_0$ is a root of the polynomial $p(\lambda)$ which contradict the hypothesis. Therefore the 1st row of $\bar{C}$ is $\neq 0$. It is easy to see that $\bar{C}$ has m linearly independent rows iff the 1st row is $\neq 0$. There $\bar{C}$ has m linearly independent rows. □

We end this section by making one observation about the direct sum conjecture when one of the dimensions is equal to 2. The direct sum conjecture can in general be restated as follows: Let $\{G_i\}_{1\leq i\leq r}$ be a set of $m\times n$ matrices and let $\{\bar{G}_j\}_{1\leq j\leq s}$ be a set of $p\times q$ matrices over a field F. If $\{s_i\}_{1\leq i\leq r}$ and $\{\bar{s}_j\}_{1\leq j\leq s}$ are 2 disjoint sets of indeterminates, then

$$\delta\left[\begin{array}{c|c} \sum_{i=1}^{r} G_i s_i & 0 \\ \hline 0 & \sum_{j=1}^{s} \bar{G}_j \bar{s}_j \end{array}\right] = \delta\left(\sum_{i=1}^{r} G_i s_i\right) + \delta\left(\sum_{j=1}^{s} \bar{G}_j \bar{s}_j\right).$$

We will completely settle the case when $r=2$ over any field F. It is not hard to see that this will imply that the direct sum conjecture is true if any one of the dimensions is equal to 2. Let $\bar{G}(\bar{s})$ denote $\sum_{j=1}^{s} \bar{G}_j\bar{s}_j$. We have to show that

$$\delta((G_1s_1+G_2s_2)\oplus\bar{G}(\bar{s})) = \delta(G_1s_1+G_2s_2) + \delta(\bar{G}(\bar{s})).$$

We can assume that the pencil $G_1s_1+G_2s_2$ is regular, i.e., G_1 and G_2 are square matrices and $\det(G_1s_1+G_2s_2) \not\equiv 0$ (in all other cases, the pencil is called singular). In fact, if

$G_1s_1+G_2s_2$ is singular [J], then $G_1s_1+G_2s_2$ is equivalent to

$$\begin{bmatrix} L_{\varepsilon_1} & & & & & \\ & \ddots & & & & \\ & & L_{\varepsilon_\alpha} & & & \\ & & & L^T_{\eta_1} & & \\ & & & & \ddots & \\ & & & & & L^T_{\eta_\beta} \\ & & & & & & G'_1s_1+G'_2s_2 \end{bmatrix}$$

where $G'_1s_1+G'_2s_2$ is a regular pencil and

$$L_{\varepsilon_i}=\underbrace{\begin{bmatrix} s_1 & s_2 & & & \\ & s_1 & s_2 & & \\ & & \ddots & \ddots & \\ & & & s_1 & s_2 \end{bmatrix}}_{\varepsilon_i+1}\Bigg\}\varepsilon_i.$$

It was shown in [J] that if Card $(F)\geqq\varepsilon$, then

$$\delta\left[\begin{array}{c|c} L_\varepsilon & 0 \\ \hline 0 & G(s) \end{array}\right]=\delta(L_\varepsilon)+\delta(G(s))=\varepsilon+1+\delta(G(S)).$$

Therefore, if Card $(F)\geqq\max(\varepsilon_i,\eta_i)$ then

$$\delta((G_1s_1+G_2s_2)\oplus\bar{G}(\bar{s}))=\sum_{i=1}^{\alpha}(\varepsilon_i+1)+\sum_{i=1}^{\beta}(\eta_i+1)+\delta((G'_1s_1+G'_2s_2)\oplus\bar{G}(\bar{s})).$$

In particular if $\bar{G}(\bar{s})=0$ we have

$$\delta(G_1s_1+G_2s_2)=\sum_{i=1}^{\alpha}(\varepsilon_i+1)+\sum_{i=1}^{\beta}(\eta_i+1)+\delta(G'_1s_1+G'_2s_2).$$

If the direct sum conjecture is true for the regular pencil $G'_1s_1+G'_2s_2$, then $\delta((G'_1s_1+G'_2s_2)\oplus\bar{G}(\bar{s}))=\delta(G'_1s_1+G'_2s_2)+\delta(\bar{G}(\bar{s}))$. Hence

$$\delta((G_1s_1+G_2s_2)\oplus\bar{G}(\bar{s}))=\delta(G_1s_1+G_2s_2)+\delta(\bar{G}(\bar{s})).$$

We will show in the next section that this is indeed the case.

3. A solution to the direct sum conjecture for a large class of computations. Any regular pencil $G_1s_1+G_2s_2$ can be reduced into an equivalent pencil of the form $I_ns_1+Gs_2$, where G is an $n\times n$ matrix. Therefore, the problem is reduced to showing that

$$\delta((I_ns_1+Gs_2)\oplus\bar{G}(\bar{s}))=\delta(I_ns_1+Gs_2)+\delta(\bar{G}(\bar{s})).$$

Let $p_k(\lambda)|p_{k-1}(\lambda)|\cdots|P_1(\lambda)$ be the invariant polynomials of an $n\times n$ matrix G over a field F. We say that F is a field of *large enough cardinality* if Card $(F)\geqq\deg(p_1(\lambda))$.

DEFINITION. A polynomial is called *simple* if it splits into a product of distinct linear factors over the field F, otherwise it is called *nonsimple.*

LEMMA 3.1. *If F is a field of large enough cardinality, then $\delta((I_ns_1+Gs_2)\oplus\bar{G}(\bar{s}))\leqq\delta(\bar{G}(\bar{s}))+n+$ number of nonsimple invariant polynomials of G.*

Proof. We know that $\delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s})) \leqq \delta(I_n s_1 + Gs_2) + \delta(\bar{G}(\bar{s}))$. By [J, Thm. 3.3], $\delta(I_n s_1 + Gs_2) \leqq n +$ number of nonsimple invariant polynomials of G. Therefore $\delta((I_n S_1 + Gs_2) \oplus \bar{G}(\bar{s})) \leqq \delta(\bar{G}(\bar{s})) + n +$ number of nonsimple invariant polynomials of G. □

We now state the main theorem of this section.

THEOREM 3.2. *If F is a field of large enough cardinality, then $\delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s})) = \delta(\bar{G}(\bar{s})) + n +$ number of nonsimple invariant polynomials of G.*

Before we prove this theorem in the general case, we start by settling the particular case where all invariant polynomials are equal and irreducible.

LEMMA 3.3. *Let G be an $n \times n$ matrix with k invariant polynomials $p_1(\lambda) = p_2(\lambda) = \cdots = p_k(\lambda)$. Assume that F is a field of large enough cardinality. If $p_k(\lambda)$ is irreducible over F and if* $\deg(p_k(\lambda)) \geqq 2$, *then $\delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s})) = \delta(\bar{G}(\bar{s})) + n + k$.*

Proof of Lemma 3.3. If $p_1(\lambda) = \alpha_0 + \alpha_1 \lambda + \cdots + \alpha_{m-1}\lambda^{m-1} + \lambda^m$, then $m \geqq 2$ and $n = mk$.

$$\delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s})) = \delta\begin{bmatrix} \left.\begin{matrix} I_m s_1 + C(p_1(\lambda))s_2 & & \\ & \ddots & \\ & & I_m s_1 + C(p_1(\lambda))s_2 \end{matrix}\right\} k \text{ copies} & \\ & \bar{G}(\bar{s}) \end{bmatrix},$$

where

$$I_m s_1 + C(p_1(\lambda))s_2 = \begin{bmatrix} s_1 & s_2 & & & \\ & s_1 & s_2 & & \\ & & \ddots & \ddots & \\ & & & s_1 & s_2 \\ -\alpha_0 s_2 & -\alpha_1 s_2 & \cdots & -\alpha_{m-2}s_2 & s_1 - \alpha_{m-1}s_2 \end{bmatrix}.$$

Let $C = C(p_1(\lambda))$ and let C_i be the ith column of $I_m s_1 + Cs_2$. Using a substitution argument [BD], we can delete the 1st column of the 1st block and add linear combinations to the remaining columns to get $\delta \geqq 1 +$

$$\delta\begin{bmatrix} C_2 + \beta_1 C_1 & C_3 + \beta_2 C_1 & \cdots & C_m + \beta_{m-1}C_1 & r_{11}C_1 & r_{12}C_1 & \cdots & r_{1m}C_1 & \cdots & r_{k1}C_1 & r_{k2}C_1 & \cdots & r_{km}C_1 & H_1(s_1, s_2) \\ & & & & C_1 & C_2 & \cdots & C_m & & & & & & \\ & & & & & & & & & 0 & & & 0 & \\ 0 & & & & & & & & & C_1 & C_2 & \cdots & C_m & \\ & & & & & & & & & & & & & \bar{G}(\bar{s}) \end{bmatrix}$$

for some matrix $H_1(s_1, s_2)$ and some constants β_i and r_{ij} in F. Let $B_{22} = \cdots = B_{kk} = I_m s_1 + Cs_2$ ($m \times m$ matrices), and let $\bar{B}_{11} = [C_2 + \beta_1 C_1 \;\; C_3 + \beta_2 C_k \cdots C_m + \beta_{m-1}C_1]$, and $m \times (m-1)$ matrix, and $B_{1,i+1} = [r_{i1}C_1 \;\; r_{i2}C_1 \;\cdots\; r_{im}C_1] = C_1[r_{i1} \;\; r_{i2} \;\cdots\; r_{im}] = C_1 r_i$, for $i = 1, 2, \cdots, k-1$.

Then

$$\delta \geqq 1 + \delta\begin{bmatrix} \bar{B}_{11} & B_{12} & B_{13} & \cdots & B_{1k} & H_1(s_1, s_2) \\ & B_{22} & & & & \\ & & B_{33} & & & 0 \\ & & & \ddots & & \\ & 0 & & & B_{kk} & \bar{G}(\bar{s}) \end{bmatrix}.$$

Using row and column operations we want to eliminate the blocks B_{1i} for $i \geqq 2$. We will eliminate the block B_{12} without changing the other blocks, and the same will work for the other blocks B_{1i}. Let

$$\bar{P} = \begin{bmatrix} a_{21} & a_{22} & \cdots & a_{2m} \\ a_{31} & a_{32} & \cdots & a_{3m} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{bmatrix}$$

be an arbitrary $(m-1)\times m$ matrix to be determined later. Multiplying $\bar{B}_{11}$ by $\bar{P}$ and adding it to B_{12}, we get $\bar{B}_{12} = B_{12} + \bar{B}_{11}\bar{P}$, i.e.,

$$\bar{B}_{12} = [C_1 r_{11} \cdots C_1 r_{1m}] + [C_2 + \beta_1 C_1 \cdots C_m + \beta_{m-1} C_1] \begin{bmatrix} a_{21} & \cdots & a_{2m} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mm} \end{bmatrix}$$

$$= [C_1 r_{11} \cdots C_1 r_{1m}] + \left[C_1 \sum_{i=1}^{m-1} \beta_i a_{i+1,1} \cdots C_1 \sum_{i=1}^{m-1} \beta_i a_{i+1,m} \right] + [C_2 \cdots C_m]\bar{P}$$

$$= [C_1(r_{11} + \textstyle\sum \beta_i a_{i+1,1}) \cdots C_1(r_{1m} + \textstyle\sum \beta_i a_{i+1,m})] + [C_2 \cdots C_m]\bar{P}.$$

Therefore $\bar{B}_{12} = [C_1 C_2 \cdots C_m]P$ where

$$P = \begin{bmatrix} r_{11} + \sum \beta_i a_{i+1,1} & \cdots & r_{1m} + \sum \beta_i a_{i+1,m} \\ a_{21} & \cdots & a_{2m} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mm} \end{bmatrix}.$$

Multiplying B_{22} on the left by $-P$ and adding it to $\bar{B}_{12}$, we get

$$\begin{aligned} \bar{\bar{B}}_{12} &= \bar{B}_{12} - PB_{22} = (I_m s_1 + Cs_2)P - P(I_m s_1 + Cs_2) \\ &= Ps_1 + CPs_2 - Ps_1 - PCs_2 \\ &= (CP - PC)s. \end{aligned}$$

We want to find $\bar{P}$ such that $CP - PC = 0$.

Claim. There exists an $(m-1)\times m$ matrix $\bar{P}$ such that $CP = PC$.

Proof of Claim. Let $\bar{P}$ be arbitrary and let R_{i+1} be the ith row of $\bar{P}$. Then

$$P = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_m \end{bmatrix}$$

where $R_1 = r_1 + \sum_{i=1}^{m-1} \beta_i R_{i+1}$ and $r_1 = (r_{11}\ r_{12}\ \cdots\ r_{1m})$ as shown before.

Now

$$PC = CP \Leftrightarrow \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_m \end{bmatrix} \quad C = \begin{bmatrix} 0 & 1 & & \\ & 0 & 1 & \\ & & \ddots & 1 \\ -\alpha_0 & -\alpha_1 & \cdots & -\alpha_{m-1} \end{bmatrix} \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ R_m \end{bmatrix}$$

$$\Leftrightarrow\begin{cases} R_1C = R_2 \\ R_2C = R_3 \\ \vdots \\ R_{m-1}C = R_m \\ R_mC = -\alpha_0R_1 - \alpha_1R_2 - \cdots - \alpha_{m-1}R_m \end{cases}$$

$$\Leftrightarrow\begin{cases} R_2 = R_1C \\ R_3 = R_1C^2 \\ \vdots \\ R_m = R_1C^{m-1} \\ R_1C^m = -\alpha_0R_1 - \alpha_1R_2 - \cdots - \alpha_{m-1}R_m \end{cases}$$

$$\Leftrightarrow\begin{cases} R_2 = R_1C \\ R_3 = R_1C^2 \\ \vdots \\ R_m = R_1C^{m-1} \\ R_1C^m = R_1(-\alpha_0 - \alpha_1C - \cdots - \alpha_{m-1}C^{m-1}). \end{cases}$$

C is a root of its minimal polynomial $p_1(\lambda) = \alpha_0 + \alpha_1\lambda + \cdots + \alpha_{m-1}\lambda^{m-1} + \lambda^m$. Therefore $\alpha_0 + \alpha_1C + \cdots + \alpha_{m-1}C^{m-1} + C^m = 0$, i.e., $C^m = -\alpha_0 - \alpha_1C - \cdots - \alpha_{m-1}C^{m-1}$. So the last equation of the system of equations is always true. Therefore, $PC = CP$ is equivalent to

$$\begin{cases} R_2 = R_1C \\ R_3 = R_1C^2 \\ \vdots \\ R_m = R_1C^{m-1}. \end{cases}$$

But $R_1 = r_1 + \sum_{i=1}^{m-1} \beta_iR_{i+1}$. Therefore,

$$R_1 = r_1 + \sum_{i=1}^{m-1} \beta_iR_1C^i = r_1 + R_1 \sum_{i=1}^{m-1} \beta_iC^i,$$

i.e.,

$$R_1(I_m - \beta_1C - \beta_2C^2 - \cdots - \beta_{m-1}C^{m-1}) = r_1.$$

By Lemma 2.1 the matrix $I_m - \beta_1C - \cdots - \beta_{m-1}C^{m-1}$ is invertible. Therefore, we can take $R_1 = r_1(I_m - \beta_1C - \cdots - \beta_{m-1}C^{m-1})^{-1}$,

$$R_2 = R_1C = r_1(I_m - \beta_1C - \cdots - \beta_{m-1}C^{m-1})C^{-1},$$

$$\vdots$$

$$R_m = R_1C^{m-1} = r_1(I_m - \beta_1C - \cdots - \beta_{m-1}C^{m-1})^{-1}C^{m-1}.$$

Then

$$\bar{P} = \begin{bmatrix} R_2 \\ R_3 \\ \vdots \\ R_m \end{bmatrix}.$$

will be such that $PC = CP$. □

It follows that we can choose $\bar{P}$ such that $\bar{\bar{B}}_{12}=0$. Similarly, we delete the blocks $B_{13}, \cdots, B_{1k}$. Therefore,

$$\delta \geqq 1+\delta \left[\begin{array}{ccc|c} \bar{B}_{11} & & & H_1(s_2, s_2) \\ & B_{22} & & \\ & & \ddots & 0 \\ & & B_{kk} & \\ \hline & 0 & & \bar{G}(\bar{s}) \end{array}\right].$$

Since $P_1(\lambda)=\alpha_0+\alpha_1\lambda+\cdots+\alpha_{m-1}\lambda^{m-1}+\lambda^m$ has no roots in F, then by Lemma 2.3, the m rows of $\bar{B}_{11}$ are linearly independent. Hence, we get

$$\delta \geqq m+1+\delta \left[\begin{array}{ccc|c} B_{22} & & & \\ & \ddots & 0 & H_2(s_1, s_2) \\ 0 & & B_{kk} & \\ \hline & 0 & & \bar{G}(\bar{s})+H_3(s_1, s_2) \end{array}\right].$$

By induction $\delta \geqq k(m+1)+\delta(\bar{G}(\bar{s})+\bar{H}(s_1, s_2))$. Putting $s_1=s_2=0$ we get $\delta \geqq k(m+1)+\delta(\bar{G}(\bar{s}))=n+k+\delta(\bar{G}(\bar{s}))$. By Lemma 2.4 we have

$$\delta \leqq \delta(\bar{G}(\bar{s})+n+\text{number of nonsimple invariant polynomials of } G.$$

Since all k invariant polynomials of G are nonsimple then

$$\delta=\delta(\bar{G}(\bar{s}))+n+k. \qquad \square$$

Before we prove Theorem 3.2, we claim that $p_k(\lambda)$ can be assumed to be nonsimple. In fact, if $p_k(\lambda)$ is simple, then let $n_i=\deg(p_i(\lambda))$. Hence $p_k(\lambda)=(\lambda-\lambda_1)(\lambda-\lambda_2)\cdots(\lambda-\lambda_{n_k})$, where the λ_i's are pairwise distinct.

Let

$$G'=\begin{bmatrix} C(p_{k-1}(\lambda)) & & & \\ & C(p_{k-2}(\lambda)) & & 0 \\ 0 & & \ddots & \\ & & & C(p_1(\lambda)) \end{bmatrix}.$$

Then G is similar to the matrix

$$\begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & 0 \\ & & \ddots & \\ & & & \lambda_{n_k} \\ 0 & & & G' \end{bmatrix}.$$

Therefore,

$$\delta=\delta((I_n s_1+G s_2)\oplus \bar{G}(\bar{s}))$$

$$=\delta \begin{bmatrix} s_1+\lambda_1 s_2 & & & & \\ & s_1+\lambda_2 s_2 & & 0 & \\ & & \ddots & & \\ & & & s_1+\lambda_{n_k} s_2 & \\ & 0 & & & I_{n-n_k} s_1+G' s_2 \\ & & & & & \bar{G}(\bar{s}) \end{bmatrix}.$$

Hence $\delta = n_k + \delta((I_{n-n_k}s_1 + G's_2) \oplus \bar{G}(\bar{s}))$. By induction on k we can assume that

$$\delta((I_{n-n_k}s_1 + G's_2) \oplus \bar{G}(\bar{s})) = \delta(\bar{G}(\bar{s})) + n - n_k + \text{number of nonsimple invariant polynomials of } G'.$$

The invariant polynomials of G' are $p_{k-1}(\lambda) | p_{k-2}(\lambda) | \cdots | p_1(\lambda)$. Since $p_k(\lambda)$ is simple then G and G' have the same number of nonsimple invariant polynomials. Therefore,

$$\delta = n_k + \delta(\bar{G}(\bar{s})) + n - n_k + \text{number of nonsimple invariant polynomials of } G',$$

i.e.

$$\delta = \delta(\bar{G}(\bar{s})) + n + \text{number of nonsimple invariant polynomials of } G.$$

We now restate the main theorem and prove it.

THEOREM 3.4. *If F is a field of large enough cardinality, then $\delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s})) = \delta(\bar{G}(\bar{s})) + n +$ number of nonsimple invariant polynomials of G.*

Proof. As we have seen before, it is enough to show the theorem in the case where $p_k(\lambda)$ is nonsimple. If $p_k(\lambda)$ is nonsimple, then one of the following must be true:

1. $p_k(\lambda)$ has no roots in F.
2. $p_k(\lambda) = (\lambda - \lambda_1)(\lambda - \lambda_2) \cdots (\lambda - \lambda_m) q(\lambda)$ where the λ_i's are pairwise distinct and $q(\lambda)$ has no roots in F.
3. $p_k(\lambda)$ has a multiple root λ_0 in F.

We will settle each case separately.

Case 1. $p_k(\kappa)$ has no roots in F.

Write the invariant polynomials as product of irreducible factors

$$\begin{aligned} p_1(\lambda) &= q_1^{\tau_{11}} q_2^{\tau_{12}} \cdots q_r^{\tau_{1r}}, \\ p_2(\lambda) &= q_1^{\tau_{21}} q_2^{\tau_{22}} \cdots q_r^{\tau_{2r}}, \\ &\vdots \\ p_k(\lambda) &= q_1^{\tau_{k1}} q_2^{\tau_{k2}} \cdots q_r^{\tau_{kr}}. \end{aligned}$$

Since $p_k(\lambda)$ has no roots in F, then there exists an i such that $\deg(q_i) \geqq 2$ and $\tau_{ki} > 0$. For the sake of clarity, let us assume that $i = 1$, i.e. $\deg(q_1) \geqq 2$ and $\tau_{k1} \geqq 1$. Since $p_k(\lambda) | p_i(\lambda)$, for $i = 1, 2, \cdots, k-1$, $\tau_{i1} \neq 0$, for $i = 1, 2, \cdots, k$.

Let A be the generalized Jordan form of G of Lemma 2.2. Then

$$A = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_k \end{bmatrix},$$

where

$$A_i = \left[\begin{array}{cccc:c} C(q_1) & H_1 & & & & \\ & C(q_1) & \ddots & & & \\ & & \ddots & H_1 & & 0 \\ & & & C(q_1) & & \\ \hdashline & & & & \multicolumn{2}{c}{\ddots} \\ & & & & C(q_r) & H_r \\ & & & & & C(q_r) \ddots \\ 0 & & & & & \ddots \; H_r \\ & & & & & C(q_r) \end{array}\right],$$

with τ_{i1} copies of the $C(q_1)$ block and τ_{ir} copies of the $C(q_r)$ block.

$C(q_i)$ being the companion matrix of the polynomial q_i, and

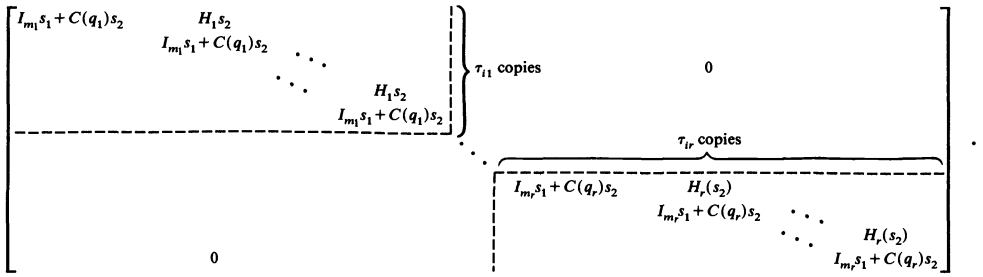
$$H_i = \left[\begin{array}{c|c} 0 & 0 \\ \hline 1 & 0 \end{array}\right].$$

Let $\delta = \delta((I_n s_1 + G s_2) \oplus \bar{G}(\bar{s}))$. Then $\delta = \delta((I_n s_1 + A s_2) \oplus \bar{G}(\bar{s}))$, i.e.,

$$(*) \qquad \delta = \delta \begin{bmatrix} I_{n_1}s_1 + A_1 s_2 & & & \\ & I_{n_2}s_1 + A_2 s_2 & & 0 \\ & & \ddots & \\ 0 & & & I_{n_k}s_1 + A_k s_2 \\ & & & & \bar{G}(\bar{s}) \end{bmatrix},$$

where $n_i = \deg(p_i(\lambda))$, $i = 1, 2, \cdots, k$.

Let $m_j = \deg(q_j)$, $j = 1, 2, \cdots, r$. Then, $I_{n_i}s_1 + A_i s_2 =$



Consider the matrix in equation (*). By row transformations, we can move the last $n_i - m_1$ rows of each block $I_{n_i}s_1 + A_i s_2$ to the bottom. By column transformations, we can move the last $n_i - m_1$ columns of each block $I_{n_i}s_1 + A_i s_2$ to the right-hand side. since the complexity does not change under these transformations, we get

$$\delta = \delta \left[\begin{array}{ccc|c} I_{m_1}s_1 + C(q_1)s_2 & & & \\ & I_{m_1}s_1 + C(q_1)s_2 & & \\ & & \ddots & Hs_2 \\ & & I_{m_1}s_1 + C(q_1)s_2\bar{G}(\bar{s}) & \\ \hline & 0 & & I_{n-km_1}s_1 + G''s_2 \end{array}\right].$$

Since $I_{n-km_1}s_1 + G''s_2$ has $n - km_1$ linearly independent rows, then, we have $\delta \geqq n - km_1 +$

$$\min_N \delta \left[\begin{array}{ccc|c} I_{m_1}s_1 + C(q_1)s_2 & & & \\ & \ddots & 0 & \\ & & I_{m_1 s_1} + C(q_1)s_{2\bar{G}(\bar{s})} & Hs_2 + N(I_{n-km1}s_1 + G''s_2) \\ 0 & & & \end{array}\right].$$

Eliminate the last $n - km_1$ columns to get:

$$\delta \geqq n - km_1 + \delta((I_{km_1}s_1 + Bs_2) \oplus \bar{G}(\bar{s})),$$

where

$$B = \begin{bmatrix} C(q_1) & & & 0 \\ & C(q_1) & & \\ & & \ddots & \\ 0 & & & C(q_1) \end{bmatrix}, \quad k \text{ copies.}$$

It is easy to see that B has k invariant polynomials equal to q_1. Then, since q_1 is irreducible and $m_1 = \deg(q_1) \geqq 2$, we can apply Lemma 3.3. Therefore $\delta((I_{km_1}s_1 + Bs_2) \oplus \bar{G}(\bar{s})) = \delta(\bar{G}(\bar{s})) + km_1 + k$. Hence

$$\delta \geqq n - km_1 + \delta(\bar{G}(\bar{s})) + km_1 + k = \delta(\bar{G}(\bar{s})) + n + k.$$

Since $p_k(\lambda)$ has no roots in F and $p_k(\lambda) | p_i(\lambda)$ for $i = 1, 2, \cdots, k-1$, then all k invariant polynomials of G are nonsimple. Therefore, $\delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s})) \geqq \delta(\bar{G}(\bar{s})) + n +$ number of nonsimple invariant polynomials of G. Since Lemma 2.4 settled the other inequality, then Theorem 3.2 is true for the case where $p_k(\lambda)$ has no roots in F.

Case 2. $p_k(\lambda) = (\lambda - \lambda_1)(\lambda - \lambda_2) \cdots (\lambda - \lambda_m) q(\lambda)$, where the λ_i's are pairwise distinct and $q(\lambda)$ has no roots in F.

We know that G is similar to its rational canonical form:

$$\begin{bmatrix} C(p_k(\lambda)) & & & \\ & C(p_{k-1}(\lambda)) & & 0 \\ & & \ddots & \\ 0 & & & C(p_1(\lambda)) \end{bmatrix}.$$

On the other hand, it is easy to check that $C(p_k(\lambda))$ is similar to the matrix

$$\begin{bmatrix} \lambda_1 & & & & \\ & \lambda_2 & & & \\ & & \ddots & & 0 \\ 0 & & & \lambda_m & \\ & & & & C(q(\lambda)) \end{bmatrix}.$$

In fact, they have the same unique invariant polynomial $p_k(\lambda)$. Therefore, G is similar to the matrix:

$$\begin{bmatrix} \lambda_1 & & & & \\ & \lambda_2 & & & \\ & & \ddots & & \\ & & & \lambda_m & \\ & & & & A \end{bmatrix},$$

where A has the following invariant polynomials:

$$q(\lambda) | p_{k-1}(\lambda) | \cdots | p_1(\lambda).$$

Let $\delta = \delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s}))$. Then,

$$\delta = \delta \begin{bmatrix} s_1 + \lambda_1 s_2 & & & & & \\ & s_1 + \lambda_2 s_2 & & & 0 & \\ & & \ddots & & & \\ & & & s_1 + \lambda_m s_2 & & \\ & 0 & & & I_{n-m}s_1 + As_2 & \\ & & & & & \bar{G}(\bar{s}) \end{bmatrix}.$$

Therefore,

$$\delta = m + \delta \begin{bmatrix} I_{n-m}s_1 + As_2 & 0 \\ 0 & \bar{G}(\bar{s}) \end{bmatrix} = m + \delta((I_{n-m}s_1 + As_2) \oplus \bar{G}(\bar{s})).$$

Since $q(\lambda)$, the first invariant polynomial of A, is irreducible, then by Case 1 we have:

$$\delta((I_{n-m}s_1 + As_2) \oplus \bar{G}(\bar{s})) = \delta(\bar{G}(\bar{s})) + n - m + k.$$

Therefore,

$$\delta = \delta(\bar{G}(\bar{s})) + n + \text{number of nonsimple invariant polynomials of } G.$$

Case 3. $p_k(\lambda)$ has multiple root λ_0.

Since $p_k(\lambda)|p_{k-1}(\lambda)|\cdots|p_1(\lambda)$ then λ_0 is a multiple root of $p_i(\lambda)$ of multiplicity $m_i \geqq 2$, $i = 1, 2, \cdots, k$, i.e. $p_i(\lambda) = (\lambda - \lambda_0)^{m_i} q_i(\lambda)$ where $q_i(\lambda_0) \neq 0$ and $q_k(\lambda)|q_{k-1}(\lambda)|\cdots|q_1(\lambda)$. Therefore, G has k nonsimple invariant polynomials. By Lemma 2.4 we have

$$\delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s})) \leqq \delta(\bar{G}(\bar{s})) + n + k.$$

The proof of the theorem will be complete if we can show that $\delta((I_n s_1 + Gs_2) \oplus \bar{G}(\bar{s})) \geqq \delta(\bar{G}(\bar{s})) + n + k$.

G is similar to its rational canonical form

$$\begin{bmatrix} C(p_k(\lambda)) & & & \\ & C(p_{k-1}(\lambda)) & & 0 \\ & & \ddots & \\ & 0 & & C(p_1(\lambda)) \end{bmatrix}.$$

It is easy to check that the matrix

$$H_i = \begin{bmatrix} \left.\begin{matrix} \lambda_0 & 1 & & \\ & \lambda_0 & 1 & \\ & & \ddots & \ddots \\ & & & 1 \\ & & & \lambda_0 \end{matrix}\right\} m_i & \\ & C(q_i(\lambda)) \end{bmatrix}$$

has only one invariant polynomial equal to $(\lambda - \lambda_0)^{m_i} q_i(\lambda) = p_i(\lambda)$. Therefore it is similar to the matrix $C(p_i(\lambda))$. Hence, G is similar to

$$\begin{bmatrix} H_k & & & \\ & H_{k-1} & & 0 \\ & & \ddots & \\ & 0 & & H_1 \end{bmatrix}.$$

Therefore,

$$\delta = \delta \begin{bmatrix} I_{n_k}s_1 + H_k s_2 & & & & \\ & I_{n_{k-1}}s_1 + H_{k-1}s_2 & & 0 & \\ & & \ddots & & \\ & 0 & & I_{n_1}s_1 + H_1 s_1 & \\ & & & & \bar{G}(\bar{s}) \end{bmatrix}$$

where

$$I_{n_i}s_1 + H_i s_2 = \begin{bmatrix} s_1+\lambda_0 s_2 & s_2 & & & \\ & s_1+\lambda_0 s_2 & s_2 & & \\ & & \ddots & \ddots & \\ & & & s_2 & \\ & & & s_1+\lambda_0 s_2 & \\ & & & & I_{n_i-m_i}s_1 + C(q_i(\lambda))s_2 \end{bmatrix}.$$

We can use techniques similar to Case 1 to show that

$$\delta \geqq \sum_{i=1}^{k}(n_i-m_i)+\delta \begin{bmatrix} \begin{matrix} s_1+\lambda_0 s_2 & s_2 & & \\ & s_1+\lambda_0 s_2 & \ddots & \\ & & \ddots & s_2 \\ & & & s_1+\lambda_0 s_2 \end{matrix} & \Big\} m_k & \\ & \cdots & \begin{matrix} \overbrace{\begin{matrix} s_1+\lambda_0 s_2 & s_2 & & \\ & s_1+\lambda_0 s_2 & \ddots & \\ & & \ddots & s_2 \\ & & & s_1+\lambda_0 s_2 \end{matrix}}^{m_1} & \\ & \bar{G}(\bar{s}) \end{matrix} \end{bmatrix}$$

Now apply the transformation

$$\begin{cases} s_1 \to s_1' - \lambda_0 s_2' \\ s_2 \to s_2'. \end{cases}$$

We get

$$\delta \geqq n - \sum_{i=1}^{k} m_i + \delta \begin{bmatrix} \begin{matrix} s_1' & s_2' & & \\ & s_1' & \ddots & \\ & & \ddots & s_2' \\ & & & s_1' \end{matrix} & \Big\} m_k & \\ & \ddots & \begin{matrix} \overbrace{\begin{matrix} s_2' & s_2' & & \\ & s_1' & \ddots & \\ & & \ddots & s_2' \\ & & & s_1' \end{matrix}}^{m_1} & \\ & \bar{G}(\bar{s}) \end{matrix} \end{bmatrix}.$$

Since $m_i \geqq 2$, then each block has at least 2 rows and 2 columns. Then move the first row of each block to the right-hand side and use the basic linear independence and substitution argument [BD] to get

$$\delta \geqq n - \sum_{i=1}^{k} m_i + 2k + \delta \begin{bmatrix} I_{\sum(m_i-1)}s_2' + G_1 s_1' & \\ & \bar{G}(\bar{s}) \end{bmatrix},$$

for some matrix G_1.

Putting $s_1' = 0$, we get

$$\delta \geqq n - \sum_{i=1}^{k} m_i + 2k + \delta \begin{bmatrix} I_{\sum (m_i - 1)} s_2' & \\ & \bar{G}(\bar{s}) \end{bmatrix}.$$

Therefore,

$$\delta \geqq n - \sum_{i=1}^{k} m_i + 2k + \sum_{i=1}^{k} (m_i - 1) + \delta(\bar{G}(\bar{s})),$$

i.e., $\delta \geqq n + k + \delta(\bar{G}(\bar{s}))$.

This ends the proof of Theorem 3.4. □

In [J], Ja'Ja' showed that if $p_k(\lambda) | \cdots | p_1(\lambda)$ are the invariant polynomials of an $n \times n$ matrix G and if F is a field of large enough cardinality which contains the roots of $p_1(\lambda)$, then $\delta(I_n s_1 + G s_2) = n +$ number of nonsimple invariant polynomials of G. He conjectured that the condition "F contains the roots of $p_1(\lambda)$" can be omitted.

The conjecture is true as the following corollary shows.

COROLLARY 1. *Let F be a field with "large enough cardinality." Then $\delta(I_n s_1 + G s_2) = n +$ number of nonsimple invariant polynomials of G.*

COROLLARY 2. *If $G_1 s_1 + G_2 s_2$ is a regular pencil, then $\delta((G_1 s_1 + G_2 s_2) \oplus \bar{G}(\bar{s})) = \delta(G_1 s_1 + G_2 s_2) + \delta(\bar{G}(\bar{s}))$.*

COROLLARY 3. *If $G_1 s_1 + G_2 s_2$ is any pencil of $n \times m$ matrices, then*

$$\delta((G_1 s_1 + G_2 s_2) \oplus \bar{G}(\bar{s})) = \delta(G_1 s_1 + G_2 s_2) + \delta(\bar{G}(\bar{s})).$$

COROLLARY 4 (Direct Sum Conjecture). *If $G_1, G_2, \cdots, G_r$ are $n \times m$ matrices and $\bar{G}_1, \bar{G}_2, \cdots, \bar{G}_s$ are $p \times q$ matrices over any field of cardinality $\geqq \max\{r, s, n, m, p, q\}$, and if $2 \in \{r, s, n, m, p, q\}$. Then*

$$\delta\left(\left(\sum_{i=1}^{r} G_i s_i\right) \oplus \left(\sum_{j=1}^{s} \bar{G}_j \bar{s}_j\right)\right) = \delta\left(\sum_{i=1}^{r} G_i s_i\right) + \delta\left(\sum_{j=1}^{s} \bar{G}_j \bar{s}_j\right).$$

4. Direct sum conjecture for the case when $r = mn - 2$. When working with some unknown $m \times n \times r$ tensors defined by $m \times n$ matrices $G_1, G_2, \cdots, G_r$ it is common to assume that the tensor is nondegenerate, i.e., the matrices $G_1, \cdots, G_r$ are linearly independent. If the tensor is degenerate, then it can be reduced to one of smaller size. In this section, we will show the Direct Sum conjecture in the case where one of the summands is a nondegenerate $m \times n \times (mn - 2)$ tensor.

Let us start by stating a couple of definitions and a proposition from [AL].

DEFINITION. A space of $m \times n$ matrices is said to be *perfect* if it is generated (as a vector space) by rank one matrices.

DEFINITION. Two spaces V and W 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

$$W = PVQ = \{PXQ : X \in V\}.$$

PROPOSITION. *If V is a vector space of $m \times n$ matrices of dimension $mn - 2$, then V is perfect unless it is equivalent to the space of all matrices $X = (x_{ij})$ for which $x_{11} + x_{22} = 0$ and $x_{12} = 0$.*

THEOREM 4.1. *Let $\{G_i\}_{1 \leqq i \leqq r}$ be a set of $m \times n$ matrices and $\{\bar{G}_j\}_{1 \leqq j \leqq t}$ be a set of $p \times q$ matrices.*

If $r = mn-2$ and if $\sum_{i=1}^{mn-2} G_i s_i$ is nondegenerate, then

$$\delta\left(\left(\sum_{i=1}^{r} G_i s_i\right)\oplus\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right)\right) = \delta\left(\sum_{i=1}^{r} G_i s_i\right) + \delta\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right).$$

Proof. Let V be the vector space generated by the $m \times n$ matrices $G_1, G_2, \cdots, G_r$.

Assume first that V is not equivalent to the space of all matrices $X = (x_{ij})$ for which $x_{11} + x_{22} = 0$ and $x_{12} = 0$. Then, by the previous proposition, V is perfect. Therefore, V is generated (as a vector space) by rank one matrices. Hence, there exist $mn-2$ rank one matrices whose span contains the matrices $G_1, G_2, \cdots, G_r$. Therefore, $\delta(\sum_{i=1}^{mn-2} G_i s_i) \leqq mn-2$. But $\delta(\sum_i G_i s_i)\oplus(\sum_j \bar{G}_j \bar{s}_j) \leqq \delta(\sum_i G_i s_i) + \delta(\sum_j \bar{G}_j \bar{s}_j)$, therefore $\delta((\sum_i G_i s_i)\oplus(\sum_j \bar{G}_j \bar{s}_j)) \leqq mn-2+\delta(\sum_j \bar{G}_j \bar{s}_j)$. On the other hand, since $\sum_{i=1}^{mn-2} G_i s_i$ is nondegenerate, then $\dim(s) = mn-2$. Applying Theorem 1.2, we get

$$\delta\left(\left(\sum_{i=1}^{mn-2} G_i s_i\right)\oplus\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right)\right) \geqq mn-2+\delta\left(H(\bar{s})\oplus\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right)\right).$$

Setting $x_i = 0$, we get

$$\delta \geqq mn-2+\delta\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right).$$

Therefore

$$\delta = mn+2+\delta\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right) = \delta\left(\sum_{i=1}^{r} G_i s_i\right) + \delta\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right).$$

Assume now that V is equivalent to the space of all matrices $X = (x_{ij})$ for which $x_{11} + x_{22} = 0$ and $x_{12} = 0$. Then there exist nonsingular matrices P and Q such that

$$G_i' = PG_iQ = \begin{bmatrix} x_{11}^{(i)} & 0 & x_{13}^{(i)} & \cdots & x_{1n}^{(i)} \\ x_{21}^{(i)} & -x_{11}^{(i)} & x_{23}^{(i)} & \cdots & x_{2n}^{(i)} \\ \vdots & \vdots & & & \\ x_{m1}^{(i)} & x_{m2}^{(i)} & x_{m3}^{(i)} & \cdots & x_{mn}^{(i)} \end{bmatrix} \quad \text{for } i = 1, 2, \cdots, mn-2.$$

Therefore,

$$\delta\left(\left(\sum_{i=1}^{r} G_i s_i\right)\oplus\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right)\right) = \delta\left(\left(\sum_{i=1}^{r} G_i' s_i\right)\oplus\left(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j\right)\right).$$

Not all entries $x_{11}^{(i)}$ are 0; in fact if this were the case then the space generated by $G_1', \cdots, G_s'$ would have dimension $\leqq mn-3 < r$. Therefore, the summand $\sum_{i=1}^{r} G_i' s_i$, and hence $\sum_{i=1}^{r} G_i s_i$ is degenerate, which contradicts the hypothesis.

The $(1, 1)$ entry of $\sum_{i=1}^{r} G_i' s_i$ is $\sum_{i=1}^{r} x_{11}^{(i)} s_i$. By performing transformation on the s_i's, we can assume that the $(1, 1)$ entry is s_1. Hence,

$$\delta = \delta\left(\left(\sum_i G_i s_i\right)\oplus\left(\sum_j \bar{G}_j \bar{s}_j\right)\right)$$

$$= \delta \left[\begin{array}{ccccc|c} s_1 & 0 & l_{13}(s) & \cdots & l_{1n}(s) & \\ l_{21}(s) & s_1 & l_{23}(s) & \cdots & l_{2n}(s) & 0 \\ \vdots & & & & \vdots & \\ l_{m1}(s) & l_{m2}(s) & l_{m3}(s) & \cdots & l_{mn}(s) & \\ \hline & & 0 & & & \sum_{j=1}^{t} \bar{G}_j \bar{s}_j \end{array}\right].$$

Since $\sum_{i=1}^{r} G_i s_i$ is nondegenerate, then we get

$$\delta \geqq r-1+\delta \left[\begin{array}{cccc|c} s_1 & 0 & \cdots & l_{1n}(s_1, \bar{s}) & \\ l_{21}(s_1, \bar{s}) & s_1 & \cdots & l_{2n}(s_1, \bar{s}) & 0 \\ \vdots & \vdots & & \vdots & \\ l_{m1}(s_1, \bar{s}) & l_{m2}(s_1, \bar{s}) & \cdots & l_{mn}(s_1, \bar{s}) & \\ \hline & 0 & & & \sum_{i=1}^{t} \bar{G}_j \bar{s}_j \end{array}\right].$$

Delete the last $m-2$ rows and the last $n-2$ columns of the 1st block, we get

$$\delta \geqq r-1+\delta \left[\begin{array}{cc|c} s_1 & 0 & 0 \\ l_{21}(s_1, \bar{s}) & s_1 & \\ \hline 0 & & \sum_{j=1}^{t} (\bar{G}_j \bar{s}_j) \end{array}\right].$$

Therefore, $\delta \geqq r-1+2+(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j)$, i.e. $\delta \geqq mn-1+\delta(\sum_j \bar{G}_j \bar{s}_j)$. But we know that $\delta \leqq \delta(\sum_{i=1}^{mn-2} G_i s_i)+\delta(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j)$ and $\delta(\sum_{i=1}^{mn-2} G_i s_i) \leqq mn-1$. Therefore $\delta = mn-1+\delta(\sum_j \bar{G}_j \bar{s}_j) = \delta(\sum_{u=1}^{r} G_i s_i)+\delta(\sum_{j=1}^{t} \bar{G}_j \bar{s}_j)$. □

REFERENCES

[AL] M. D. ATKINSON AND S. LLOYD, *The ranks of $m \times n \times (mn-2)$ tensors*, this Journal, 12 (1983), pp. 611-615.

[AS] A. ALDER AND V. STRASSEN, *On the algorithmic complexity of associative algebras*, Theoret. Comput. Sci., 15 (1981), pp. 201-211.

[AW] L. AUSLANDER AND S. WINOGRAD, *Direct sums of bilinear algorithms*, IBM TWRC, Yorktown Heights, NY, 1979.

[B et al.] D. BINI, G. LOTTI AND F. ROMANI, *Approximate solutions for the bilinear form computational problem*, this Journal, 4(9) (1980), pp. 692-697.

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

[BD] 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.

[FW] E. FEIG AND S. WINOGRAD, *On the Direct Sum Conjecture*, Proceedings 22nd Annual Symposium on Foundations of Computer Science, Nashville, TN, 1981, pp. 91-94.

[G] F. R. GANTMACHER, *The Theory of Matrices, Vols. 1 and 2*, Chelsea, New York, 1959.

[J] J. JA'JA', *Optimal evaluation of pairs of linear forms*, this Journal, 8 (1979), pp. 443-462.

[P] V. PAN, *How to multiply matrices faster*, Lecture Notes Comp. Sci. 179, G. Goos and J. Hartmanis, eds, Springer-Verlag, 1984, pp. 65-66.

[S] A. SCHÖNHAGE, *Partial and total matrix multiplication*, this Journal, 10 (1981), pp. 434-455.

[St] V. STRASSEN, *Vermeidung von Divisionen*, J. Reine Angew. Math., 204 (1973), pp. 184-202.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
010

ON THE SINGLE-OPERATION WORST-CASE TIME COMPLEXITY OF THE DISJOINT SET UNION PROBLEM*

NORBERT BLUM†

Abstract. We give an algorithm for the disjoint set union problem, within the class of algorithms defined by Tarjan, which has $O(\log n/\log\log n)$ single-operation time complexity in the worst case. Also we define a class of algorithms for the disjoint set union problem, which includes the class of algorithms defined by Tarjan. We prove that any algorithm from this class has at least $\Omega(\log n/\log\log n)$ single-operation time complexity in the worst case.

Key words. disjoint set union, data structure, time complexity, lower bound

AMS(MOS) subject classification. 68C25

1. Introduction. Let $S_1, \cdots, S_n$ be n disjoint sets, each containing a single element. The *disjoint set union problem* is to carry out a sequence of operations of the following two types:

FIND (x): determine the name of the set containing x.
UNION (A, B, C): combine the disjoint sets A and B into a new set named C.

The operations must be carried out on-line, that is, each instruction must be completed before the next one is known.

Let m denote the total number of unions and finds. The fastest known algorithm for the disjoint set union problem amortized over m FIND- and UNION-operations runs in time $O(m\alpha(m+n, n)+n)$ and uses $O(n)$ space, where α is a functional inverse of Ackermann's function [3], [4].

Tarjan [4], [5] defines a class of algorithms for the disjoint set union problem, which is general enough to include all known algorithms for this problem. In [4] he also defines a machine model, the pointer machine, on which all this algorithms can be implemented. Tarjan proves that any algorithm from this class has at least $\Omega(m\alpha(m+n, n)+n)$ amortized time complexity in the worst case.

If the structure of the UNION-operations is known in advance (i.e., the UNION-operations are not carried out on-line) then a sequence of m FIND- and UNION-operations can be executed in time $O(m+n)$ [1].

Kurt Mehlhorn [2] asks for upper and lower bounds on the single-operation worst-case time complexity of the disjoint set union problem. All the algorithms known to the author have single-operation worst-case time complexity at least $\Omega(\log n)$.

In § 2 we give an algorithm which is in Tarjan's class and has single-operation worst-case time complexity $O(\log n/\log\log n)$.

In § 3 we define a class of algorithms for the disjoint set union problem, which includes Tarjan's class, and prove that any algorithm from this class has at least $\Omega(\log n/\log\log n)$ single-operation time complexity in the worst case.

2. The upper bound. The data structure used by the algorithm is based on the following tree structure:

Let $k \geqq 2$ be an integer. A tree T is called a *k-UF tree* exactly if
(i) all leaves of T have the same depth,
(ii) each node v of T except the root has $\geqq k$ sons,
(iii) the root has $\geqq 2$ sons.

* Received by the editors October 29, 1984, and in revised form June 15, 1985.
† Fachbereich 10, Universität des Saarlandes, D-6600 Saarbrücken, West Germany.

The following lemma is easy to prove.

LEMMA 1. *Let T be a k-UF tree with n leaves. Then T has height* $\leqq \lceil \log_k n \rceil$.

Now we describe the algorithm in detail.

Sets are represented by *k-UF* trees. The elements of a set are stored in the leaves and the name of a set is stored in the root of the tree. Every node has a pointer to its father. The root has pointers to its sons and to the leftmost leaf. Additionally, the root contains the height of the tree, and the information if the number of its sons is $\leqq k$ or not. As usual we assume that the algorithm obtains with constant cost the leaf containing the element x when it performs the operation FIND (x) and the nodes containing A and B, respectively, when it performs the operation UNION (A, B, C).

The operation FIND (x) is performed by following the path from the leaf containing x to the root of the tree. It is easy to see that the worst-case running time of FIND (x) is $O(\log_k n)$.

The procedure UNION (A, B, C) is performed by inserting the tree with smaller height into the other tree as follows.

Without loss of generality let height $(B) \leqq$ height (A). Let r be the root of B and v be the root of the leftmost subtree of A with the same height as B. Note that v is easy to find. If height (A) = height (B), then v is the root of A. If height (A) > height (B) then follow the pointer to the leftmost leaf and then the path upward from that leaf to the root until v is found. We distinguish between two cases.

Case 1. The number of sons of $r \leqq k$. Then for all sons of r we change the pointer to r into a pointer to v.

Case 2. The number of sons of $r > k$. If $v \neq$ root of A we introduce a pointer from r to the father of v. If $v =$ root of A and the number of sons of $v \leqq k$ then for all sons of v we change the pointer to v into a pointer to r. If $v =$ root of A and the number of sons of $v > k$ then we create a new root to which the two nodes v and r point.

It is easy to see how to efficiently give the root of the new tree a pointer to the leftmost leaf, the right value for the height of the tree, and the information if the number of sons of the root is $\leqq k$ or not. Note that we can count the sons of the root up to k. It is clear that the new tree is a *k-UF* tree. It is also easy to see that the worst-case running time of UNION (A, B, C) is $O(k + \log_k n)$.

Note that the algorithm has space complexity of $O(n)$.

THEOREM 1. *Disjoint set union can be performed with single-operation worst-case time complexity* $O(\log n / \log \log n)$.

Proof. Use the algorithm described above with $k = \lceil \log n / \log \log n \rceil$. □

3. The lower bound. Next we sketch the class B of algorithms for which we prove the lower bound. The data structures used by these algorithms are linked structures which can be considered as directed graphs. The algorithms solve the disjoint set union problem with respect to the following rules.

(i) To each set and to each element exactly one distinct node in the data structure is associated containing the element or the name of the set, respectively.

(ii) The data structure can be partitioned into subgraphs such that each subgraph corresponds exactly to a current set. There exists no edge from a node in such a subgraph to a node outside the subgraph.

(iii) For executing FIND (x) the algorithm obtains the node v containing x. The algorithm follows paths with start node v until it reaches the node which contains the name of the corresponding set.

(iv) For executing FIND (x) or UNION (A, B, C) the algorithm may insert or delete *any* edge as long as rule (ii) is satisfied.

Note that we allow that the algorithm can insert or delete edges when it performs a FIND-operation. Our class B includes properly the class of pointer-machine solutions defined by Tarjan [4]. The main difference is that with respect to rule (ii) we allow any insertion or deletion of an edge in the data structure. What we forbid is that the algorithms build data structures not motivated by UNION-operations.

LEMMA 2. *Let A be an algorithm from the class B. If in every UNION-operation A inserts at most k edges into the data structure then there is a FIND-operation which needs at least time*

$$\Omega\left(\frac{\log n}{\log k+\log\log n}\right).$$

Proof. Given n disjoint sets $S_1, \cdots, S_n$, each containing a single element, we define a sequence of UNION-operations and then one FIND-operation which needs time at least $\Omega(\log n/(\log k+\log\log n))$. The sequence of UNION-operations is partitioned into $\frac{1}{2}\cdot\log n$ levels: level $1, \cdots,$ level $\frac{1}{2}\cdot\log n$.

For $1\leqq i\leqq\frac{1}{2}\cdot\log n$ the following hold.

(a) All unions at level $i-1$ are performed before the first one at level i, for $i>1$.

(b) A UNION-operation at level i combines two disjoint sets of size 2^{i-1} to one set of size 2^i.

Let S be a set which is constructed by the algorithm at level i. Then $G_i(S)$ denotes the subgraph in the data structure corresponding to S after the last UNION at level i.

We define

$$A_0 := \{S_1, \cdots, S_n\},$$

$$A_i := \{S \mid S \text{ is constructed at level } i \text{ and } \forall \text{ nodes } v\in G_i(S)\text{: degree}(v)\leqq ik\cdot\log n\},$$

$$i=1, \cdots, \tfrac{1}{2}\cdot\log n.$$

The UNION-operations which are performed at level i are the following. While $|A_{i-1}|>1$ the algorithm takes two sets B and C from A_{i-1} and performs UNION (B, C, B').

CLAIM. $|\{S_j \mid j\in\{1, \cdots, n\} \text{ and } \exists S\in A_{1/2\log n}\text{: } S_j\subseteq S\}|\geqq n/2$.

The claim can be used to prove the lemma in the following way: Note that $\forall S\in A_{1/2\cdot\log n}$: $|S|=\sqrt{n}$. Hence from the claim it follows that $|A_{1/2\cdot\log n}|\geqq 1/2\sqrt{n}$. Let $S\in A_{1/2\cdot\log n}$. Since $|S|=\sqrt{n}$ and $\forall v\in G_{1/2\cdot\log n}(S)$: degree $(v)\leqq 1/2k\cdot\log^2 n$ there exists an element x with the following property:

The length of any path in $G_{1/2\cdot\log n}(S)$ from the node which contains x to the node which contains the name of S is at least

$$\log_{1/2\cdot k\cdot\log^2 n}\sqrt{n}=\frac{\log n}{2(\log k+2\cdot\log\log n)}.$$

Hence the operation FIND (x) needs time of at least $\log n/(2(\log k+\log\log n))$. This proves the lemma.

Proof of the claim. Since A_{i-1} contains at most $n/2^{i-1}$ sets, at most $n/2^i$ UNION-operations are performed at level i. Hence at most $nk/2^i$ edges are inserted into the data structure at level i.

Let the operation UNION (B, C, B') be performed at level i. By the definition of A_{i-1} it is clear that

$$\forall v\in G_{i-1}(B)\cup G_{i-1}(C)\text{: degree}(v)\leqq(i-1)k\cdot\log n.$$

Hence if there exists a node $v \in G_i(B')$ with degree $(v) > ik \cdot \log n$, at least $k \log n$ edges are inserted into $G_i(B')$ at level i. Hence there exist at most $n \cdot k/(2^i k \cdot \log n) = n/(2^i \log n)$ such sets. Each set which is constructed at level i has 2^i elements. Hence

$$|\{S_j | j \in \{1, \cdots, n\} \text{ with } \exists S \in A_{i-1} \text{ and } S_j \subseteq S\}|$$

$$-|\{S_j | j \in \{1, \cdots, n\} \text{ with } \exists S \in A_i \text{ and } S_j \subseteq S\}| \leqq \frac{n}{\log n}.$$

Hence

$$|\{S_j | j \in \{1, \cdots, n\} \text{ with } \exists S \in A_{1/2 \cdot \log n} \text{ and } S_j \subset S\}|$$

$$\geqq n - \frac{1}{2} \cdot \log n \cdot \frac{n}{\log n} = \frac{n}{2}.$$

This proves the claim. □

THEOREM 2. *Every algorithm from the class B has single-operation time complexity at least* $\Omega(\log n / \log \log n)$ *in the worst case.*

Proof. Apply Lemma 2 with $k \leqq \log n / \log \log n$. □

Acknowledgment. I thank Kurt Mehlhorn for calling my attention to the single-operation worst-case time complexity of the disjoint set union problem.

REFERENCES

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

[2] K. MEHLHORN, *Data Structures and Algorithms* 1, *Sorting and Searching*, Springer-Verlag, Berlin, 1984.

[3] R. E. TARJAN, *Efficiency of a good but not linear set union algorithm*, J. Assoc. Comput. Mach., 22 (1975), pp. 215–225.

[4] ———, *A class of algorithms which require nonlinear time to maintain disjoint sets*, J. Comput. System Sci., 18 (1979), pp. 110–127.

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

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
011

RANKING AND UNRANKING OF AVL-TREES*

LIWU LI†

Abstract. In this paper, we consider the problem of generating, ranking, and unranking of AVL-trees with n leaves. We represent AVL-trees by integer-pair sequences, called LDP-sequences. Then we propose a linear ordering among these sequences, i.e., among the AVL-trees. The problem of ranking is to determine the order number (rank) of a given tree in this ordering, unranking means constructing the tree of a given rank. The main result is that ranking and unranking can be done in $O(n \log^2 n)$ and $O(n \log^3 n)$ time, respectively, after a preprocessing step that takes $O(n^2 \log n)$ time.

Key words. algorithm, procedure, function, complexity, AVL-trees, partial preorder, dynamic programming, dividing and conquering, ranking and unranking

1. Introduction. The problem of generating, ranking, and unranking of some classes of trees has received considerable attention in the recent past. In [3], [4], [5], [6] and [9], algorithms are given for binary trees and k-ary trees. Zaks and Richards, in [7], present algorithms for generating, ranking, and unranking all trees with n_i nodes having k_i sons each, $i=1, 2, \cdots, t$, and n_0+1 leaves ($n_0=\sum_{1\le i\le t}(k_i-1)n_i$). In [10] and [11], the problem of ranking, unranking, and generating of 2-3-trees with n leaves and ordered B-trees with n leaves, respectively, is solved. As pointed out in [10], these algorithms mentioned above have obvious use in the generation of random data to test and predict the behavior of algorithms that manipulate these classes of trees.

AVL-trees are one kind of balanced-tree schemes, proposed for the organization of information so as to generate worst case logarithmic search time. AVL-trees are introduced by Adel'son-Vel'skii and Landis in [12], and are studied by some computer scientists recently. The *height* of a tree, which is also the height of the root, is the length of a longest path from the root to a leaf. An *AVL-tree* is a binary tree such that at each vertex v the heights of the left and right subtrees of v differ by at most one. If a subtree is missing, it is deemed of "height" -1.

In the papers mentioned above, typically, a one-to-one correspondence is established between a class of trees and certain integer sequences. A linear ordering is established among these sequences. It is then shown how to determine the position (rank) of a given sequence in this ordering (ranking), and vice versa (unranking). In [3] and [4], a binary tree is represented by the sequence listing the level numbers of the leaves from left to right. The sequences representing binary trees are called feasible sequences. The fact which is made use of in [3] and [4] to alter a feasible sequence to another feasible sequence is that deleting the two leaf-sons of a vertex in a binary tree results in another binary tree. The fact generally does not hold in AVL-trees. In [5], a permutation σ on $\{1, 2, \cdots, n\}$ which is generated by labeling the vertices of a binary tree T in preorder and reading off the labels in inorder, represents the binary tree T. By inserting number $n+1$ in one of some positions, another permutation σ' on $\{1, 2, \cdots, n+1\}$ will be resulted in and represents a binary tree with $n+1$ vertices. In [6], a binary tree with n internal vertices and $n+1$ leaves is represented by the 0-1-sequence generated by visiting the vertices in preorder and replacing the internal and external vertices by 1 and 0, respectively. Other equivalent representations are also studied in this paper. In [7], a one-to-one correspondence between all the ordered trees, which have n_0+1 leaves and n_i internal vertices with k_i sons each for $i=1, 2, \cdots, t$, and all the lattice paths in the $(t+1)$-dimensional space from the point

* Received by the editors June 24, 1983, and in final revised form August 26, 1985.

† Department of Computer Science, Nankai University, Tianjin, People's Republic of China. Present address, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada.

$(n_0, n_1, \cdots, n_t)$ to the origin, which do not go below the hyperplane $x_0 = \sum_1^t (k_i - 1)x_i$. In [8], by listing all level numbers of the vertices in preorder, the sequence represents a unique rooted tree. In [9], Knott shows a correspondence between binary trees and "tree permutations." The ordering he defines on binary trees corresponds to a lexicographic ordering of tree permutations. The representations mentioned above do not provide a mechanism which regards, directly or indirectly, the height of a binary tree and the height-balance requirement of an AVL-tree. In other words, the methods apply to those trees each vertex of which can have two subtrees of any different heights provided they satisfy other specified requirement. On the other hand, the problem of generating, ranking, and unranking of such trees that all of the leaves of one tree are on the same level is studied in [10] and [11]. The authors represent a 2-3-tree or a B-tree by listing the numbers of sons of internal vertices, level by level, from left to right on the same level. Obviously, because the leaves of an AVL-tree are generally on different levels, two different AVL-trees may result in the same sequence according to the above method.

Before introducing our representations of AVL-trees, we state an important property of AVL-trees in the following lemma, which is an easy corollary of [2, Thm. A, p. 453].

LEMMA 1. *The maximum of the heights over all AVL-trees with n leaves is* $O(\log n)$.

In the following sections, we use $UN(n)$, or simply UN where n is apparent from the context, to denote the maximum of the heights over all AVL-trees with n leaves. In fact, the calculation of value $UN(n)$ can be included into algorithm PREPROCESSING described in § 3 without affecting its time complexity.

2. LDP-sequences of AVL-trees and their linear ordering. In this paper, the set of vertices of an AVL-tree with m vertices will be represented by integer set $\{1, 2, \cdots, m\}$, and the tree itself, which is a binary tree, is represented by two arrays, LEFTSON [i] and RIGHTSON [i], where $1 \leqq i \leqq m$, and LEFTSON [i] (RIGHTSON [i]) points at the left (right, respectively) son of vertex i, if the son exists, otherwise at empty symbol $\varnothing$. Therefore, we sometimes do not differentiate the two data types, integer and vertex. Because we are interested in the number of leaves of an AVL-tree and there are three different AVL-trees with one leaf, we treat each of them as a single element. In other words, if the father of a leaf has exactly one son, the leaf and its father will not be treated with separately. The following concept reflects this point of view. The *partial preorder* traversal of a binary tree, denoted as P-preorder traversal, is defined recursively as follows:

1. Visit the root;

2. If the root has both a left and a right son, visit the left subtree in P-preorder, and then visit the right subtree in P-preorder.

Figure 1 shows three AVL-trees; the vertices which can be visited in P-order traversal are darkened.

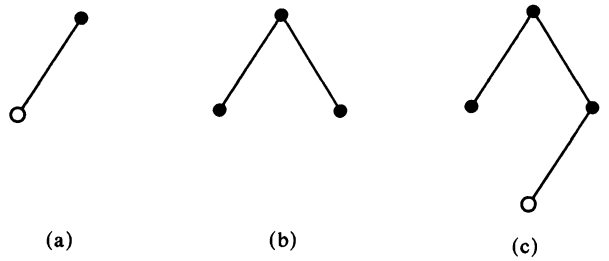


FIG. 1

For a given AVL-tree T, we associate an integer pair $\langle l_i, r_i \rangle$, called *leaf-distribution-pair* or LDP, to each vertex i which can be visited in the P-preorder traversal of T, where $l_i(r_i)$ is the number of leaves in the left (right, respectively) subtree of vertex i. The *LDP-sequence* of an AVL-tree T is obtained by listing the LDP of each vertex visited in the P-preorder traversal of T. Therefore, the LDP-sequence of the AVL-tree in Fig. 1c is $\langle 1, 1\rangle\langle 0, 0\rangle\langle 1, 0\rangle$. We should notice that the vertex with LDP $\langle 0, 0\rangle$ has no proper descendents which are leaves, but it is a leaf itself.

Obviously, not all of integer pair sequences are LDP-sequences. The following algorithm tests whether a given nonnegative integer pair sequence is an LDP-sequence, and if the answer is positive, gives the corresponding AVL-tree described by arrays LEFTSON and RIGHTSON.

We use a Pascal-like language to describe algorithms.

ALGORITHM 1. VERIFYING

Input. A sequence of nonnegative integer pairs $\langle l_1, r_1\rangle\langle l_2, r_2\rangle \cdots \langle l_k, r_k\rangle$. For simplicity, we assume that no pair has the form $\langle l, r\rangle$, where $l+r>1$ and ($l=0$ or $r=0$), since such a pair is inhibited from appearing in an LDP-sequence.

Output. "YES," if the sequence is an LDP-sequence, "NO," otherwise. If the answer is "YES," the corresponding AVL-tree is described by arrays LEFTSON [i] and RIGHTSON [i], $1 \leqq i \leqq m$. Array HEIGHT [i], $1 \leqq i \leqq m$, is the height of vertex i in the AVL-tree.

Method. We follow the input sequence to construct an AVL-tree whose vertices possess the pairs as their LDP's.

We use a stack ANC to store the vertices i whose subtrees are being constructed. When the left(right) subtree is being constructed, integer $-i$ (i, respectively) is stored in ANC.

We define a two-operand logic function $\ominus$. The result of $\ominus(op1, op2)$, where the first operand $op1$ is an integer and the second operand $op2$ is a nonnegative integer pair $\langle l, r\rangle$, is true if and only if ($op1=1$ and $l=r=0$) or $op1=l+r$. In other words, $\ominus(op1, op2)$ is true if and only if the number of leaves of the AVL-tree with root LDP $op2$ is $op1$.

```
var top, m: integer;
begin (* the main program *)
  top := 0; (* make stack ANC empty *)
  m := k; (* m points at the maximum existing vertex k *)
  VERIFY (1)
end;
procedure VERIFY (i: integer);
var p: vertex; H: integer;
begin
  if ⊖(1, ⟨l_i, r_i⟩)
  then (* the subtree rooted at i has only one leaf *)
  begin
    if l_i = 0 and r_i = 0
    then (* the subtree consists of one vertex *)
    begin
      LEFTSON [i] := RIGHTSON [i] := ∅;
      HEIGHT [i] := H := 0
    end
    else (* vertex i is the father of a leaf *)
```

```
        begin
          m := m+1; (* m will be the son of vertex i *)
          LEFTSON [m] := RIGHTSON [m] := ∅; (* m is a leaf *);
          HEIGHT [m] := 0;
          if l_i = 0
          then (* m is the right son *)
          begin
            LEFTSON [i] := ∅; RIGHTSON [i] := m
          end
          else
          begin
          LEFTSON [i] := m; RIGHTSON [i] := ∅
          end;
          HEIGHT [i] := H := 1
        end;
        while top > 0 and ANC [top] > do(* we come back from the right son of vertex
ANC [top] *)
          if |HEIGHT [ANC [top]+1] − H| > 1  (* left son of ANC [top] is
ANC [top]+1 *)
          then return "NO" (* lose balance here *)
          else
          begin
            HEIGHT [ANC [top]] := H := max {HEIGHT [ANC [top]+1], H}+1;
            top := top − 1
          end;
        if top > 0
        then (* right subtree of vertex − ANC [top] is to be constructed *)
        begin
          ANC [top] := p := − ANC [top];
          if not ⊖(r_p, ⟨l_i+1, r_{i+1}⟩)
          then return "NO"
          else VERIFY (i+1); RIGHTSON [p] := i+1
        end
        else
          if k = i
          then return "YES" (* input is exactly an LDP-sequence *)
          else return "NO" (* input is too long *)
        end
        else (* the subtree rooted at i has more than one leaf *)
        begin
          top := top+1;
          ANC [top] := −i; (* we are to construct the left subtree of vertex i *)
          if i = k
          then return "NO" (* input is too short *)
          else
        if not ⊖(l_i, ⟨l_{i+1}, r_{i+1}⟩)
          then return "NO"
          else VERIFY (i+1); LEFTSON [i] := i+1
      end
    end.
```

Since for each input integer pair, we call procedure VERIFY only once and put the subscript of the pair onto stack ANC only once, the time complexity of algorithm VERIFYING is $O(k)$, linear to the length of input.

The following lemma is an easy result.

LEMMA 2. *Two AVL-trees are isomorphic if and only if they have the same LDP-sequences.*

We shall define a linear ordering among AVL-trees with the same number of leaves, in fact, among the LDP-sequences. After defining a priority, which is a partial relation, among LDP's, we define the linear ordering as the lexicographic ordering among LDP-sequences, where each LDP is deemed to be a symbol.

DEFINITION 1 (*Priority* among LDP's). We say LDP $\langle l_1, r_1\rangle$ *is prior to* LDP $\langle l_2, r_2\rangle$, denoted by $\langle l_1, r_1\rangle \prec \langle l_2, r_2\rangle$, if and only if:

$$\ominus(l_2 + r_2, \langle l_1, r_1\rangle) \quad \text{and} \quad ((l_1 = l_2 = r_1 = 0 \text{ and } r_2 - 1) \text{ or } l_1 < 2).$$

DEFINITION 2. (*Linear ordering* $\prec$ among LDP-sequences). Among the LDP-sequences whose AVL-trees have the same number of leaves, we say LDP-sequence $L1$ *is prior to* LDP-sequence $L2$, denoted by $L1 \prec L2$, if and only if in lexicographic ordering on the basic relation $\prec$ of LDPs, $L1$ is precedent to $L2$.

To prove the ordering $\prec$ among LDP-sequences whose AVL-trees have the same number of leaves is a linear ordering, we need the following lemma.

LEMMA 3. *Let $L1$, $L2$ and $L3$ be LDP-sequences whose AVL-trees have the same number of leaves. If $L1 \prec L2$ and $L2 \prec L3$, then $L1 \prec L3$.*

Proof. Let $Li = \langle l_1^{(i)}, r_1^{(i)}\rangle\langle l_2^{(i)}, r_2^{(i)}\rangle \cdots \langle l_{ki}^{(i)}, r_{ki}^{(i)}\rangle$, $i = 1, 2, 3$. Since $L1 \prec L2$ and $L2 \prec L3$, there are two integers J and K such that

$$\langle l_i^{(1)}, r_i^{(1)}\rangle = \langle l_i^{(2)}, r_i^{(2)}\rangle, \quad \text{for } 1 \leqq i \leqq K-1,$$

$$\langle l_K^{(1)}, r_K^{(1)}\rangle \prec \langle l_K^{(2)}, r_K^{(2)}\rangle,$$

$$\langle l_j^{(2)}, r_j^{(2)}\rangle = \langle l_j^{(3)}, r_j^{(3)}\rangle, \quad \text{for } 1 \leqq j \leqq J-1,$$

$$\langle l_J^{(2)}, r_J^{(2)}\rangle \prec \langle l_J^{(3)}, r_J^{(3)}\rangle.$$

We prove the lemma in three cases.

Case 1. $K < J$. Since $\langle l_i^{(1)}, r_i^{(1)}\rangle = \langle l_i^{(2)}, r_i^{(2)}\rangle = \langle l_i^{(3)}, r_i^{(3)}\rangle$, $1 \leqq i \leqq K-1$, and $\langle l_K^{(1)}, r_K^{(1)}\rangle \prec \langle l_K^{(2)}, r_K^{(2)}\rangle = \langle l_K^{(3)}, r_K^{(3)}\rangle$, we can deduce that $L1 \prec L3$.

Case 2. $K > J$. Since $\langle l_j^{(1)}, r_j^{(1)}\rangle = \langle l_j^{(2)}, r_j^{(2)}\rangle = \langle l_j^{(3)}, r_j^{(3)}\rangle$, $1 \leqq j \leqq J-1$, and $\langle l_J^{(1)}, r_J^{(1)}\rangle = \langle l_J^{(2)}, r_J^{(2)}\rangle \prec \langle l_J^{(3)}, r_J^{(3)}\rangle$, we can deduce $L1 \prec L3$.

Case 3. $K = J$. By the constructing of LDP-sequences from AVL-trees, it is easy to see that if $l_K^{(2)} + r_K^{(2)} = 1$, the three LDP's $\langle l_K^{(1)}, r_K^{(1)}\rangle$, $\langle l_K^{(2)}, r_K^{(2)}\rangle$, and $\langle l_K^{(3)}, r_K^{(3)}\rangle$ must be

$$\langle 0, 0\rangle, \langle 0, 1\rangle, \langle 1, 0\rangle,$$

respectively. Since $\langle 0, 0\rangle \prec \langle 1, 0\rangle$, we can deduce $L1 \prec L3$ when $l_K^{(2)} + r_K^{(2)} = 1$. If $l_K^{(2)} + r_K^{(2)} > 1$, by Definitions 1 and 2, we know that

$$l_K^{(1)} < l_K^{(2)} < l_K^{(3)}.$$

The relation $L1 \prec L3$ follows from $\langle l_K^{(1)}, r_K^{(1)}\rangle \prec \langle l_K^{(3)}, r_K^{(3)}\rangle$. □

Now we can say that the relation $\prec$ among LDP-sequences whose AVL-trees have the same number of leaves is indeed a linear ordering. In the following sections, the number of leaves is fixed as n.

3. Ranking and unranking. In this section, we first present a preprocessing algorithm to assign values to some arrays with the help of which we can rank and

unrank AVL-trees with n leaves in $O(n \log^2 n)$ and $O(n \log^3 n)$ time, respectively. This idea is also used in the ranking and unranking of a 2-3-trees and B-trees in [10] and [11], respectively.

The two-dimension array $K1[\langle l, r\rangle, h]$, where $\langle l, r\rangle$ is an LDP with $l+r \leqslant n$ and h is an integer, contains the number of AVL-trees of height h with root LDP $\langle l, r\rangle$. Array $K2[k, h]$ contains the number of AVL-trees of height h with k leaves. Another useful array is $K3[\langle l, r\rangle, h]$, which contains the number of AVL-trees of height h with root LDP prior or equal to $\langle l, r\rangle$. For the simplicity of the algorithms, we follow a convention that when an item of an array which has not been assigned any values is used, the content of the item is understood to be integer zero.

ALGORITHM 2. PREPROCESSING

```
var l, k, h: integer;
begin
  K1[⟨0, 0⟩, 0] := K3[⟨0, 0⟩, 0] := 1;
  K1[⟨0, 1⟩, 1] := K3[⟨0, 1⟩, 0] := K3[⟨0, 1⟩, 1] := 1;
  K1[⟨1, 0⟩, 1] := K3[⟨1, 0⟩, 0] := 1; K3[⟨1, 0⟩, 1] := 2; (* there are three AVL-trees
        with one leaf whose LDP-sequences are ⟨0, 0⟩, ⟨0, 1⟩ and ⟨1, 0⟩ respec-
        tively *)
  K2[1, 0] := 1; K2[1, 1] := 2;
  for k := 2 to n do
    for h := 1 to UN do
    begin
      K2[k, h] := K3[⟨0, k⟩, h] := 0; (* initiation *)
      for l := 1 to k − 1 do
      begin
        K1[⟨l, k − 1⟩, h] := K2[l, h − 2] * K2[k − l, h − 1]
                           + K2[l, h − 1] * K2[k − l, h − 2]
                           + K2[l, h − 1] * K2[k − l, h − 1];
        K2[k, h] := K2[k, h] + K1[⟨l, k − l⟩, h];
        K3[⟨l, k − l⟩, h] := K3[⟨l − 1, k − (l − 1)⟩, h] + K1[⟨l, k − l⟩, h]
      end
    end
end.
```

Since UN is $O(\log n)$, it is easy to prove that the time complexity of algorithm PREPROCESSING is $O(n^2 \log n)$.

For any given LDP-sequence whose AVL-tree has n leaves, the following algorithm will output its order number in the linear ordering $\prec$. The difficulty involved in ranking AVL-trees lies on the height-balance requirement. In other words, though $\langle l_1, r_1\rangle \cdots \langle l_{p-1}, r_{p-1}\rangle\langle l_p, r_p\rangle$ is the prefix of an LDP-sequence and $\langle l_p, r_p\rangle \cdots \langle l_q, r_q\rangle$ is an LDP-sequence, $\langle l_1, r_1\rangle \cdots \langle l_{p-1}, r_{p-1}\rangle\langle l_p, r_p\rangle \cdots \langle l_q, r_q\rangle$ need not be the prefix of any LDP-sequences. Following the input sequence $\langle l_1, r_1\rangle\langle l_2, r_2\rangle \cdots \langle l_k, r_k\rangle$, when we meet an LDP $\langle l_i, r_i\rangle$ we count only those LDP-sequences $\langle l'_1, r'_1\rangle \cdots \langle l'_s, r'_s\rangle$, where $\langle l'_1, r'_1\rangle \prec \langle l_i, r_i\rangle$ and $\langle l_1, r_1\rangle \cdots \langle l_{i-1}, r_{i-1}\rangle\langle l'_1, r'_1\rangle \cdots \langle l'_s, r'_s\rangle$ is the prefix of an LDP-sequence. For each i, $1 \leqslant i \leqslant k$, we actually calculate the height of the vertex corresponding to $\langle l_i, r_i\rangle$.

ALGORITHM 3. RANKING

Input. An LDP-sequence $\langle l_1, r_1\rangle\langle l_2, r_2\rangle \cdots \langle l_k, r_k\rangle$ whose AVL-tree has n leaves.

Output. The rank of input LDP-sequence in the linear ordering $\prec$.

Method. Since the linear ordering $\prec$ is essentially a lexicographic ordering, we proceed to treat with the LDP's $\langle l_1, r_1\rangle\langle l_2, r_2\rangle, \cdots, \langle l_k, r_k\rangle$ in this order. For each LDP

$\langle l_i, r_i\rangle$, we make use of function ACCUMULATE to calculate the number of LDP-sequences $\langle l_1, r_1\rangle\langle l_2, r_2\rangle \cdots \langle l_{i-1}, r_{i-1}\rangle\langle l'_i, r'_i\rangle \cdots$, where $\langle l'_i, r'_i\rangle \prec \langle l_i, r_i\rangle$.

```
begin (* RANKING *)
  top := 0; (* make stack ANC empty *)
  sum := 1; (* variable sum will contain the rank upon completion of
RANKING *)
  RANK (1);
  write sum
end;
procedure RANK (i: integer);
begin
  case ⟨l_i, r_i⟩ of
    ⟨0, 0⟩: HEIGHT[i] := H := 0; (* no LDP is prior to ⟨0, 0⟩ *)
    ⟨0, 1⟩: begin (* only one LDP ⟨0, 0⟩ ≺ ⟨0, 1⟩ *)
              HEIGHT[i] := H := 1;
              L := 0;
              U := 0;
              K[0] := 1
            end;
    ⟨1, 0⟩: begin (* the LDPs ⟨0, 0⟩, ⟨0, 1⟩ are prior to ⟨1, 0⟩ *)
              HEIGHT[i] := H := 1;
              L := 0;
              U := 1;
              K[0] := K[1] := 1
            end;
    others: begin
              L := 1;
              U := UN;
              for h := 1 to UN do
                K[h] := K3[⟨l_i − 1, r_i + 1⟩, h]
            end
  end; (* case end *)
  if ⟨l_i, r_i⟩ ≠ ⟨0, 0⟩ then sum := sum + ACCUMULATE; (* the number of LDP-
       sequences with prefixes ⟨l_1, r_1⟩ ··· ⟨l_{i-1}, r_{i-1}⟩⟨l'_i, r'_i⟩, where ⟨l'_i, r'_i⟩ ≺ ⟨l_i, r_i⟩,
       is added to the variable sum *)
  if ⟨l_i, r_i⟩ ∉ {⟨0, 0⟩, ⟨0, 1⟩, ⟨1, 0⟩}
  then (* we must continue ranking the descendants of vertex i *)
  begin
    top := top + 1;
    ANC[top] := −i; (* left son of vertex i is to be treated with *)
    RANK (i + 1)
  end
  else (* we will go back to the lowest ancestor of i whose right subtree has not
       been treated with *)
  begin
    while top > 0 and ANC[top] > 0 do
      begin
        HEIGHT[ANC[top]] := H := max {HEIGHT[ANC[top] + 1], H};
            (* note that the left son of vertex ANC[top] is ANC[top] + 1 *)
        top := top − 1
      end;
```

```
        if top > 0
        then (* ANC [top] < 0 and the right son, which is i+1, of vertex −ANC [top]
                has not been treated with *)
        begin
        ANC [top] := −ANC [top];
          RANK (i+1)
        end
      end
    end (* procedure RANK end *)
    function ACCUMULATE;
    var p, h, x: integer; j: vertex;
    begin
      for p := top downto 1 do
      if ANC [p] > 0
      then (* the LDP being considered is in the right subtree of vertex ANC [p] *)
      begin
        L = HEIGHT [ANC [p]+1]+1; (* the left son of ANC [p] is vertex
ANC [p]+1 *)
        U := HEIGHT [ANC [p]+1]+2;
        K[U] := K[L];
        K[L] := K[L−2]+K[L−1]
      end
        else
        begin
          j := −ANC [p]; (* the LDP being considered is in the left subtree of j *)
          K[U+2] := K[U] * K2[r_j, U+1];
          for h := U+1 downto L+2 do
            K[h] := K[h−2] * K2[r_j, h−1]
                    + K[h−1] * K2[r_j, h−2]
                    + K[h−1] * K2[r_j, h−1];
          K[L+1] := K[L] * (K2[r_j, L]+K2[r_j, L−1]):
          L := L+1;
          U := U+2
        end; (* now, K[h], L ≦ h ≦ U, is the number of LDP-sequences with prefix
⟨l_1, r_1⟩ · · · ⟨l_{i−1}, r_{i−1}⟩⟨l'_i, r'_i⟩, where ⟨l'_i, r'_i⟩ < ⟨l_i, r_i⟩, whose AVL-trees are of height h *)
        x := 0;
        for h := L to U do x := x+K[h]:
        ACCUMULATE := x
    end.
```

Because U in the function ACCUMULATE is always less than $2 * \text{top} + \text{UN}$, and top is at most UN, by Lemma 1, the time spent for each call of function ACCUMULATE is $O(\log^2 n)$. For each LDP $\langle l_i, r_i\rangle$, we call ACCUMULATE once and push the vertex i on stack ANC at most once in procedure RANK, it is easy to prove the time complexity of algorithm RANKING is $O(n \log^2 n)$.

We will discuss in the following the problem of unranking, i.e., given an integer r, where $1 \leqq r \leqq \sum_h K2[n, h]$, how we can construct an LDP-sequence which has the order number (rank) r in the linear ordering $\prec$. As linear ordering $\prec$ is a lexicographic ordering, we produce LDP's $\langle l_1, r_1\rangle$, $\langle l_2, r_2\rangle, \ldots$, one by one, such that for each i there is an LDP-sequence with prefix $\langle l_1, r_1\rangle\langle l_2, r_2\rangle \cdots \langle l_i, r_i\rangle$ which has rank r in the linear ordering $\prec$.

ALGORITHM 4. UNRANKING

Input. Integers n and r, where $1 \leqq r \leqq \sum_h K2[n, h]$.

Output. An LDP-sequence $\langle l_1, r_1\rangle\langle l_2, r_2\rangle \cdots \langle l_k, r_k\rangle$, whose AVL-tree has n leaves and the rank of which in the linear ordering $\prec$ is r.

Method. We produce the LDP's $\langle l_1, r_1\rangle, \langle l_2, r_2\rangle, \cdots, \langle l_k, r_k\rangle$ in this order. To determine $\langle li, r_i\rangle$, in fact, to determine l_i, such that there is an LDP-sequence with prefix $\langle l_1, r_1\rangle\langle l_2, r_2\rangle \cdots \langle l_i, r_i\rangle$ which has rank r, the binary search technique and function ACCUMULATE defined in algorithm RANKING are used.

```
begin (* UNRANKING *)
  top := 0; (* make stack ANC empty *)
  i := 1; (* integer i is used as a pointer pointing at the LDP being constructed *)
  UNRANK (r, n)
end;
procedure UNRANK (s, m: integer);
begin (* we are to determine an LDP ⟨l_i, r_i⟩ such that ⊖(m, ⟨l_i, r_i⟩) is true and the
number of all the LDP-sequences with prefixes ⟨l_1, r_1⟩ ··· ⟨l_{i-1}, r_{i-1}⟩⟨l'_i, r'_i⟩, where
⟨l'_i, r'_i⟩ ≺ ⟨l_i, r_i⟩ (where ⟨l'_i, r'_i⟩ ≺ ⟨l_i, r_i⟩ or ⟨l'_i, r'_i⟩ = ⟨l_i, r_i⟩), is less than (greater than or equal
to, respectively) s *)
    if m = 1
    then
    begin
      L := 0;
      U := 0;
      K[0] := 1;
      k1 := ACCUMULATE;
      L := 0;
      U := 1;
      K[0] := K[1] := 1;
      k2 := ACCUMULATE;
      if S ≦ k1
      then
      begin
        ⟨l_i, r_i⟩ := ⟨0, 0⟩;
        HEIGHT [i] := H := 0
      end
      else
        if s ≦ k2
        then
        begin
          ⟨l_i, r_i⟩ := ⟨0, 1⟩;
          HEIGHT [i] := H := 1; s := s − k1
      end
      else
      begin
        ⟨l_i, r_i⟩ := ⟨1, 0⟩;
        HEIGHT [i] := H := 1; s := s − k2
      end;
    while top > 0 and ANC [top] > 0 do
      begin
        HEIGHT [ANC [top]] := H := max {HEIGHT [ANC [top] + 1], H} + 1;
```

```
      top := top − 1
    end;
  if top > 0
    then (* right subtree of vertex −ANC [top] has not been constructed *)
    begin
      ANC [top] := j := −ANC [top];
      i := i + 1; UNRANK (s, r_j)
      end
      else k := i
  end
  else (* m > 1 *)
  begin
    I := 1; J := m − 1;
    while J − I > 0 do
      begin
        K := ⌊(I + J)/2⌋;
        L := 0; U := UN;
        for h := L to U do
          K[h] := K3[⟨K, m − K⟩, h];
        if ACCUMULATE < s
        then I := K + 1
        else J := K
      end; (* binary search end *)
      ⟨l_i, r_i⟩ := ⟨I, m − I⟩;
      L := 0; U := UN;
      for h := L to U do K[h] := K3[⟨I − 1, m − (I − 1)⟩, h];
      s := s − ACCUMULATE;
      top := top + 1;
      ANC [top] := −i; i := i + 1;
      UNRANK (s, I) (* constructing the left subtree *)
  end
end.
```

For each call of UNRANK, the most costly part is the second while statement which will repeat $O(\log m)$ tiles. Since ACCUMULATE is of $O(\log^2 n)$ time complexity, UNRANK is of $O(\log^3 n)$ time complexity. UNRANK is to be called $O(n)$ times for each execution of the main program, and therefore, the time complexity of algorithm UNRANKING is $O(n \log^3 n)$.

To generate all the AVL-trees with n leaves represented by LDP-sequences in the order $<$, we can make use of algorithm UNRANKING. Let $s = \sum_h K2[n, h]$. The following algorithm will generate all AVL-trees with n leaves.

ALGORITHM 5. GENERATING

```
begin
  for u := 1 to s do
    begin
      call UNRANKING with integers n and u;
      write ⟨l_1, r_1⟩⟨l_2, r_2⟩ · · · ⟨l_k, r_k⟩
    end
end.
```

In the algorithms described in this paper, we often make use of the dividing-and-conquering technique by putting onto the stack ANC a negative integer, indicating that we are ranking or unranking the left subtree, or a positive integer, indicating that we are ranking or unranking the right subtree. Meanwhile, we keep the balance between the left and right subtrees.

Acknowledgment. The author wishes to thank the referees for their patience in carefully reading an earlier version of this paper. Their comments and suggestions were quite constructive and helpful.

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

[3] F. RUSKEY AND T. C. HU, *Generating binary trees lexicographically*, this Journal, 6 (1987), pp. 745–758.

[4] F. RUSKEY, *Generating t-ary trees lexicographically*, this Journal, 7 (1978), pp. 706–712.

[5] A. E. TROJANOWSKI, *Ranking and listing algorithms for k-ary trees*, this Journal, 7 (1978), pp. 492–509.

[6] S. ZAKS, *Lexicographic generation of ordered trees*, Theoret. Comput. Sci., 10 (1980), pp. 63–82.

[7] S. ZAKS AND D. RICHARDS, *Generating trees and other combinatorial objects lexicographically*, this Journal, 8 (1979), pp. 73–81.

[8] T. BEYER AND S. M. HEDETNIEMI, *Constant time generation of rooted trees*, this Journal, 9 (1980), pp. 706–712.

[9] G. D. KNOTT, *A numbering system for binary trees*, Comm. ACM, 20(2) (1977), pp. 113–115.

[10] U. GUPTA, D. T. LEE AND C. K. WONG, *Ranking and unranking of 2-J trees*, this Journal, 11 (1982), pp. 582–590.

[11] U. J. GUPTA AND D. T. LEE, *Ranking and unranking of B-trees*, J. Algorithms, 4 (1983), pp. 51–60.

[12] G. M. ADEL'SON-VEL'SKII AND Y. M. LANDIS, *An algorithm for the organization of information*, Dokl. Akad. Nauk SSSR, 146 (1962), pp. 263–266; Soviet Math. Dokl., 3 (1962), pp. 1259–1262.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
012

A SIMPLE PARALLEL ALGORITHM FOR THE MAXIMAL INDEPENDENT SET PROBLEM*

MICHAEL LUBY†

Abstract. Two basic design strategies are used to develop a very simple and fast parallel algorithms for the maximal independent set (MIS) problem. The first strategy consists of assigning identical copies of a simple algorithm to small local portions of the problem input. The algorithm is designed so that when the copies are executed in parallel the correct problem output is produced very quickly. A very simple Monte Carlo algorithm for the MIS problem is presented which is based upon this strategy. The second strategy is a general and powerful technique for removing randomization from algorithms. This strategy is used to convert the Monte Carlo algorithm for this MIS problem into a simple deterministic algorithm with the same parallel running time.

Key words. parallel computations, NC, maximal independent set, randomizing algorithms, pairwise independences

AMS(MOS) subject classifications. 05B99, 05C99, 62E25, 62K99, 68O0, 68E10, 68G99

Introduction. A maximal independent set (MIS) in an undirected graph is a maximal collection of vertices I subject to the restriction that no pair of vertices in I are adjacent. The MIS problem is to find a MIS. In this paper, fast parallel algorithms are presented for the MIS problem. All of the algorithms are especially noteworthy for their simplicity.

One of the key properties behind any successful parallel algorithm design is that the algorithm can be productively subdivided into a number of almost identical simple algorithms, which when executed in parallel produce a correct problem output very quickly. Monte Carlo Algorithm A for the MIS problem (§ 3.2) has an even stronger property: the subdivision into almost identical simple algorithms respects the inherent subdivision in the problem input. More specifically, the problem input is a local description of the graph in terms of vertices and edges. Algorithm A is described in terms of two algorithm templates called *ALGVERTEX* and *ALGEDGE*. A copy of *ALGVERTEX* is assigned to each vertex in the graph and a copy of *ALGEDGE* is assigned to each edge in the graph. The algorithm runs in phases. During each phase all copies of *ALGVERTEX* are executed in parallel followed by the execution of all copies of *ALGEDGE* in parallel. The algorithm has the property that after a very small number of phases the output is a MIS. This property of the Monte Carlo algorithm may make it a useful protocol design tool in distributed computation.

One of the main contributions of this paper is the development of a powerful and general technique for converting parallel Monte Carlo algorithms into deterministic algorithms (§ 4.1). Monte Carlo Algorithm B (§ 3.3) is very similar to, but slightly more complicated than, Algorithm A. The random variables in Algorithm B are mutually independent. The general technique is used to convert Algorithm B into a deterministic algorithm with the same running time. The first major step in the conversion process is a more sophisticated analysis of the algorithm (§ 4.3), which shows that if the random variables are only *pairwise independent* [Fr] then the algorithm has essentially the same expected running time as when the random variables are mutually independent. The second major step is a method for generating a probability space over n random variables containing $O(n^2)$ sample points, where the n random variables are pairwise

* Received by the editors July 1, 1985, and in final form September 18, 1985.
† Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4.

independent (§ 4.2). Algorithm C, which is almost exactly the same as Algorithm B, chooses values for the random variables by randomly choosing one of the sample points in this probability space (§ 4.4). The number of random bits needed to choose a random sample point is $O(\log n)$. Algorithm D tests in parallel all of the sample points and uses the best (§ 4.4). This algorithm is deterministic.

For purposes of analysis the P-RAM parallel computer is used [FW]. Two models of a P-RAM are considered: the CRCW P-RAM, in which concurrent reads and writes to the same memory location are allowed; and the less powerful but perhaps more realistic EREW P-RAM, in which concurrent reads and writes to the same memory location are disallowed. Algorithm A is the simplest Monte Carlo algorithm and has the best running time on a CRCW P-RAM. Algorithm B is slightly more complicated, but it is presented because it is the basis for Algorithms C and D. Algorithm D can be implemented by a logspace-uniform circuit family, where the circuit which accepts inputs of length k has depth $O((\log k)^2)$ and polynomial in k gates. (All logs are base 2 in this paper.) This establishes that the MIS problem is in NC^2 (see [Co] for a discussion of the complexity class NC). Let n be the number of vertices and m be the number of edges in the graph. Let $EO(k)$ denote "the expected values is $O(k)$." Table 1 summarizes the features of each algorithm. The analysis for Algorithm A assuming implementation on a EREW P-RAM is the same as for Algorithm B. The column labelled "Random bits" indicates the number of unbiased random bits consumed during the execution of the algorithm.

1. History of the MIS problem. The obvious sequential algorithm for the MIS problem can be simply stated as: Initialize I to the empty set; for $i = 1, \cdots, n$, if vertex i is not adjacent to any vertex in I then add vertex i to I. The MIS output by this algorithm is called the lexicographically first maximal independent set (LFMIS). Valiant [Va] noted that the MIS problem, which has such an easy sequential algorithm, may be one of the problems for which there is no fast parallel algorithm. Cook [Co] strengthened this belief by proving that outputting the LFMIS is NC^1-complete for P. This gave strong evidence that there is no NC algorithm which outputs the LFMIS. Thus, it became clear that either there was no fast parallel algorithm for the MIS problem or else the fast parallel algorithm had to have a completely different design than the sequential algorithm.

TABLE 1

Algorithm	P-RAM Type	Processors	Time	Random bits
A	CRCW	$O(m)$	$EO(\log n)$	$EO(n(\log n)^2$
B	EREW	$O(m)$	$EO((\log n)^2)$	$EO(n \log n)$
C	EREW	$O(m)$	$EO((\log n)^2)$	$EO((\log n)^2)$
D	EREW	$O(n^2 \cdot m)$	$O((\log n)^2)$	none

Surprisingly, Karp and Wigderson [KW] did develop a fast parallel algorithm for the MIS problem. They presented a randomized algorithm with expected running time $O((\log n)^4)$ using $O(n^2)$ processors, and a deterministic algorithm with running time $O((\log n)^4)$ using $O(n^3/(\log n)^3)$ processors on a EREW P-RAM, also establishing the result that the MIS problem is in NC^4. This paper describes algorithms which are substantially simpler than their algorithm, and establishes that the MIS problem is in NC^2.

Alon, Babai and Itai [ABI] independently found a Monte Carlo algorithm for the MIS problem, which is similar to Algorithm B, shortly after the present author found Algorithms B, C and D [Lu]. Their algorithm can be implemented on a EREW P-RAM with the same efficiency as is shown for Algorithm B in Table 1. They have an implementation of their algorithm on a CRCW P-RAM where the running time is $EO(\log n)$ using $O(\Delta m)$ processors, where Δ is the maximum degree of any vertex in the graph. Algorithm A was developed after seeing a preliminary version of [ABI], inspired by the CRCW P-RAM parallel computation model they consider. Algorithm A is the simplest Monte Carlo algorithm for the MIS problem. It has a more processor efficient implementation then the algorithm described in [ABI], using $O(m)$ processors versus $O(\Delta m)$.

2. Applications of the MIS algorithm. A growing number of parallel algorithms use the MIS algorithm as a subroutine. Karp and Wigderson [KW] gave logspace reductions from the *Maximal Set Packing* and the *Maximal Matching* problems to the MIS problem, and a nondeterministic logspace reduction from the 2-*Satisfiability* problem to the MIS problem. In this paper, it is shown that there is a logspace reduction from the *Maximal Coloring* problem (§ 6.1) to the MIS problem. Thus, using the results of this paper, all of these problems are now known to be in NC^2. However, Cook and Luby [CL] previously showed that 2-*Satisfiability* is in NC^2.

Lev [Le], previous to [KW], designed an algorithm for the *Maximal Matching* problem with running time $O((\log n)^4)$ on a P-RAM (and also established that the problem is in NC^5). Subsequently, Israeli and Shiloach [IS] found an algorithm for *Maximal Matching*, implemented on a CRCW P-RAM, where the running time is $O((\log n)^3)$. More recently and independently of this paper, Israeli and Itai [II] found a Monte Carlo algorithm for the *Maximal Matching* problem. The running time of their algorithm implemented on a EREW P-RAM is $EO((\log n)^2)$ and implemented on a CRCW P-RAM is $EO(\log n)$. The reduction of [KW] from the *Maximal Matching* problem to the MIS problem together with the results in this paper establishes the stronger results that there is a deterministic algorithm for *Maximal Matching* which can be implemented on a EREW P-RAM with running time $O((\log n)^2)$, and that the *Maximal Matching* problem is in NC^2.

Karloff [Kf1] uses the MIS algorithm as a subroutine for the *Odd Set Cover* problem. This algorithm can be used to convert the Monte Carlo algorithm for *Maximum Matching* [KUW] into a Las Vegas algorithm. Also, [KSS] use both the MIS algorithm and the Maximal Coloring algorithm to find a Δ vertex coloring of a graph when Δ is $O((\log n)^c)$ for some constant c, where Δ is the maximum degree of any vertex in the graph.

3. Monte Carlo MIS algorithms.

3.1. A high level description of the algorithm. All of the MIS algorithms in this paper adhere to the outline described below. The input to the MIS algorithm is an undirected graph $G=(V, E)$. The output is a maximal independent set $I \subseteq V$. Let $G'=(V', E')$ be a subgraph of G. For all $W \subseteq V'$, define the neighborhood of W to be $N(W)=\{i \in V': \exists j \in W, (i, j) \in E'\}$.

begin
$I \leftarrow \varnothing$
$G'=(V', E') \leftarrow G=(V, E)$
while $G' \neq \varnothing$ **do**
  begin
    select a set $I' \subseteq V'$ which is independent in G'

$I \leftarrow I \cup I'$
$Y \leftarrow I' \cup N(I')$
$G' = (V', E')$ is the induced subgraph on $V' - Y$.
end
end

It is easy to show that I is a maximal independent set in G at the termination of the algorithm. The crux of the algorithm is the design of the **select step**, which must satisfy two properties:

1. The **select step** can be implemented on a P-RAM so that its execution time is very small.

2. The number of executions of the body of the **while** loop before G' is empty is very small.

For each MIS algorithm presented in this paper only the description of the **select step** will be given.

The analysis of the algorithms assumes that $n = |V|$ and $m = |E|$ are stored in the first two common memory locations of the P-RAM, and that the edge descriptions follow in consecutive memory locations. The body of the **while** loop, excluding the **select step**, can be implemented on a CRCW P-RAM using $O(m)$ processors, where each execution takes time $O(1)$. The same portion of the loop can be implemented on a EREW P-RAM using $O(m)$ processors, where each execution takes time $O(\log n)$. Here and throughout the rest of this paper the low level implementation details are omitted.

3.2. Monte Carlo Algorithm A description. The simplest Monte Carlo algorithm for the MIS problem is described in this section. Without loss of generality, assume that $V' = \{1, \cdots, n'\}$. For all $i \in V'$, define adj $(i) = \{j \in V' | (i, j) \in E'\}$. A very high level description of the **select step** is:

1. Choose a random reordering (permutation) π of V',
2. $I' \leftarrow \{i \in V' | \pi(i) < \min \{\pi(j) | j \in \text{adj } (i)\}\}$.

Here is a more detailed description of the **select step.** Define an algorithm to be executed by each vertex $i \in V'$ as follows.

ALGVERTEX (i)
begin
$\pi(i) \rightarrow$ a number randomly chosen from $\{1, \cdots, n^4\}$.
end

Define an algorithm to be executed by each edge $(i, j) \in E'$ as follows.

ALGEDGE (i, j)
begin
if $\pi(i) \geqq \pi(j)$ **then** $I' \leftarrow I' - \{i\}$
else $I' \leftarrow I' - \{j\}$
end

The **select step** is:

begin
In **parallel**, $\forall i \in V'$, **execute** *ALGVERTEX* (i)
$I' \leftarrow V'$
In **parallel**, $\forall (i, j) \in E'$, **execute** *ALGEDGE* (i, j)
end

The random choices of the values of $\pi(i)$ in *ALGVERTEX* (i) are mutually independent. This is not literally an implementation of the high level description of the algorithm because there is some chance that for some $(i,j) \in E'$, $\pi(i) = \pi(j)$. However, since each pair of vertices receives the same π value with probability $1/n^4$, and there are at most $\binom{n}{2}$ pairs of vertices from V', π is a random reordering of the vertices with probability at least $1 - 1/2n^2$.

The **select step** can be implemented on a CRCW P-RAM using $O(m)$ processors, where each execution takes time $O(1)$ and consumes $O(n \log n)$ random bits. An implementation on a EREW P-RAM uses $O(m)$ processors, where each execution takes time $O(\log n)$. In § 3.4, the number of executions of the **while** loop before termination of the algorithm is proven to be $EO(\log n)$. Thus, an implementation of the algorithm on a CRCW (EREW) P-RAM uses $O(m)$ processors, where the total execution time is $EO(\log n)$ $(EO((\log n)^2))$.

3.3. Monte Carlo Algorithm B description. In this section a Monte Carlo algorithm for the MIS problem is described which is slightly more complicated than Algorithm A. Algorithm B is the basis for the deterministic algorithm presented in the following sections.

The notation of § 3.2 is retained. For each $i \in V'$, define $d(i)$, the degree of i, to be $|\text{adj}(i)|$. Define a collection of mutually independent $\{0, 1\}$ valued random variables $\{\text{coin}(i) \mid i \in V'\}$ such that if $d(i) \geqq 1$ then coin (i) takes on value 1 with probability $1/2d(i)$ and if $d(i) = 0$ then coin (i) is always 1. The **select step** is:

```
begin
  In parallel, ∀i ∈ V', compute d(i)
  X ← ∅
  In parallel, ∀i ∈ V' {choice step}
    randomly choose a value for coin (i)
    if coin (i) = 1 then X ← X ∪ {i}
  In parallel, ∀(i, j) ∈ E'
    if i ∈ X and j ∈ X then
      if d(i) ≦ d(j) then I' ← I' − {i}
      else I' ← I' − {j}
end
```

The **select step** can be implemented on a EREW P-RAM using $O(m)$ processors, where each execution takes time $O(\log n)$ and consumes $EO(n)$ random bits. In § 3.4, the number of executions of the **while** loop before termination of the algorithm is proven to be $EO(\log n)$. Thus, an implementation of the algorithm on a EREW P-RAM uses $O(m)$ processors, where the total execution time is $O((\log n)^2)$.

3.4. Analysis of Algorithms A and B. Let t_A and t_B be the number of executions of the body of the **while** loop before $G' = \varnothing$ for Algorithms A and B, respectively. In this section it is shown that $E(t_A) = O(\log n)$ and $E(t_B) = O(\log n)$. The proof is very similar for both algorithms. It is shown that on the average each execution of the body of the **while** loop eliminates a constant fraction of the edges in E'.

Let Y_k^A and Y_k^B be the number of edges in E' before the kth execution of the body of the **while** loop for Algorithms A and B, respectively. The number of edges eliminated from E' due to the kth execution of the body of the **while** loop for A and B is $Y_k^A - Y_{k+1}^A$ and $Y_k^B - Y_{k+1}^B$, respectively.

THEOREM 1.

(1) $E[Y_k^A - Y_{k+1}^A] \geqq \frac{1}{8} \cdot Y_k^A - \frac{1}{16}$.

(2) $E[Y_k^B - Y_{k+1}^B] \geqq \frac{1}{8} \cdot Y_k^B$.

From Theorem 1 it is easily shown that $E(t_A) = O(\log n)$ and that $E(t_B) = O(\log n)$. One can make even stronger claims. For example,

$$\Pr[Y_{k+1}^B \leqq \tfrac{15}{16} Y_k^B] \geqq \tfrac{1}{15}.$$

Thus, $\Pr[t_B \geqq 700 \log n] \leqq 2n^{-2000}$. This can be shown using an inequality due to Bernstein which is cited in Rényi [R, p. 387]. From this it is easy to see that the bounds on the running time and on the number of random bits used by Algorithms A and B hold with high probability. These details are omitted from this paper.

For all $i \in V'$ such that $d(i) \geqq 1$, let

$$\text{sum}(i) = \sum_{j \in \text{adj}(i)} \frac{1}{d(j)}.$$

Proof of Theorem 1. Let $G' = (V', E')$ be the graph before the kth execution of the body of the **while** loop. The edges eliminated due to the kth execution of the body of the **while** loop are the edges with at least one endpoint in the set $I' \cup N(I')$, i.e., each edge (i, j) is eliminated either because $i \in I' \cup N(I')$ or because $j \in I' \cup N(I')$. Thus,

$$E[Y_k^B - Y_{k+1}^B] \geqq \frac{1}{2} \cdot \sum_{i \in V'} d(i) \cdot \Pr[i \in I' \cup N(I')]$$

$$\geqq \frac{1}{2} \cdot \sum_{i \in V'} d(i) \cdot \Pr[i \in N(I')].$$

The remaining portion of the proofs for part (1) and part (2) of Theorem 1 are based upon Lemmas A and B, respectively. Here, only the proof for part (2) is given. (The proof for part (1) is a consequence since Lemma A is strictly stronger than Lemma B except for the $1 - 1/2n^2$ multiplicative factor.) Lemma B states that $\Pr[i \in N(I')] \geqq \frac{1}{4} \cdot \min\{\text{sum}(i)/2, 1\}$. Thus,

$$E[Y_k^B - Y_{k+1}^B] \geqq \frac{1}{8} \cdot \left(\frac{1}{2} \cdot \sum_{\substack{i \in V' \\ \text{sum}(i) \leqq 2}} d(i) \cdot \text{sum}(i) + \sum_{\substack{i \in V' \\ \text{sum}(i) > 2}} d(i) \right)$$

$$\geqq \frac{1}{8} \cdot \left(\sum_{\substack{i \in V' \\ \text{sum}(i) \leqq 2}} \sum_{j \in \text{adj}(i)} \frac{d(i)}{2 \cdot d(j)} + \sum_{\substack{i \in V' \\ \text{sum}(i) > 2}} \sum_{j \in \text{adj}(i)} 1 \right)$$

$$\geqq \frac{1}{8} \cdot \left(\sum_{\substack{(i,j) \in E' \\ \text{sum}(i) \leqq 2 \\ \text{sum}(j) \leqq 2}} \frac{1}{2} \cdot \left(\frac{d(i)}{d(j)} + \frac{d(j)}{d(i)} \right) + \sum_{\substack{(i,j) \in E' \\ \text{sum}(i) \leqq 2 \\ \text{sum}(j) > 2}} \left(\frac{d(i)}{2 \cdot d(j)} + 1 \right) + \sum_{\substack{(i,j) \in E' \\ \text{sum}(i) > 2 \\ \text{sum}(j) > 2}} 2 \right)$$

$$\geqq \frac{1}{8} \cdot |E'| = \frac{1}{8} \cdot Y_k^B. \qquad \square$$

Lemmas A and B are crucial to the proof of Theorem 1. Lemma A, due to Paul Beame, was proved after Lemma B and uses some of the same ideas as Lemma B. For expository reasons, it is presented first.

LEMMA A (Beame). *For Algorithm* A, $\forall i \in V'$ *such that* $d(i) \geqq 1$,

$$\Pr[i \in N(I')] \geqq [\tfrac{1}{4} \cdot \min\{\text{sum}(i), 1\}] \cdot \left(1 - \frac{1}{2n^2}\right).$$

Proof. With probability at least $1-1/2n^2$, π is a random reordering of V'. Assume that π is a random reordering of V'. $\forall j \in V'$, let E_j be the event that

$$\pi(j) < \min\{\pi(k) \mid k \in \text{adj}(j)\}$$

and let

$$p_j = \Pr[E_j] = \frac{1}{d(j)+1}.$$

Without loss of generality let

$$\text{adj}(i) = \{1, \cdots, d(i)\}$$

and let

$$p_1 \geqq \cdots \geqq p_{d(i)}.$$

Then, by the principle of inclusion-exclusion, for $1 \leqq l \leqq d(i)$,

$$\Pr[i \in N(I')] \geqq \Pr\left[\bigcup_{j=1}^{l} E_j\right] \geqq \sum_{j=1}^{l} p_j - \sum_{j=1}^{l} \sum_{k=j+1}^{l} \Pr[E_j \cap E_k].$$

For fixed j, k such that $1 \leqq j < k \leqq l$, let E_j' be the event that

$$\pi(j) < \min\{\pi(v) \mid v \in \text{adj}(j) \cup \text{adj}(k)\}$$

and let E_k' be the event that

$$\pi(k) < \min\{\pi(v) \mid v \in \text{adj}(j) \cup \text{adj}(k)\}.$$

Let

$$d(j,k) = |\text{adj}(j) \cup \text{adj}(k)|.$$

Then,

$$\Pr[E_j \cap E_k] \leqq \Pr[E_j'] \cdot \Pr[E_k \mid E_j'] + \Pr[E_k'] \cdot \Pr[E_j \mid E_k']$$

$$\leqq \frac{1}{d(j,k)+1} \cdot \left(\frac{1}{d(k)+1} + \frac{1}{d(j)+1}\right) \leqq 2 \cdot p_j \cdot p_k.$$

Let $\alpha = \sum_{j=1}^{d(i)} p_j$. By the technical lemma which follows,

$$\Pr[i \in N(I')] \geqq \tfrac{1}{2} \cdot \min\{\alpha, \tfrac{1}{2}\} \geqq \tfrac{1}{4} \cdot \min\{\text{sum}(i), 1\}$$

when π is a random reordering of V'. □

LEMMA B. *For Algorithm* B, $\forall i \in V'$ *such that* $d(i) \geqq 1$,

$$Pr[i \in N(I')] \geqq \tfrac{1}{4} \cdot \min\left\{\frac{\text{sum}(i)}{2}, 1\right\}.$$

Proof. $\forall j \in V'$, let E_j be the event that $\text{coin}(j) = 1$ and let

$$p_j = \Pr[E_j] = \frac{1}{2 \cdot d(j)}.$$

Without loss of generality let

$$\text{adj}(i) = \{1, \cdots, d(i)\}$$

and let

$$p_1 \geqq \cdots \geqq p_{d(i)}.$$

Let E_1' be the event E_1, and for $2 \leqq j \leqq d(i)$ let

$$E_j' = \left(\bigcap_{k=1}^{j-1} \neg E_k \right) \cap E_j.$$

Let

$$A_j = \bigcap_{\substack{v \in \text{adj}(j) \\ d(v) \geqq d(j)}} \neg E_v.$$

Then,

$$\Pr[i \in N(I')] \geqq \sum_{j=1}^{d(i)} \Pr[E_j'] \cdot \Pr[A_j \mid E_j'].$$

But

$$\Pr[A_j \mid E_j'] \geqq \Pr[A_j] \geqq 1 - \sum_{\substack{v \in \text{adj}(j) \\ d(v) \geqq d(j)}} p_v \geqq \frac{1}{2}$$

and

$$\sum_{j=1}^{d(i)} \Pr[E_j'] = \Pr\left[\bigcup_{j=1}^{d(i)} E_j \right].$$

For $k \neq j$, $\Pr[E_j \cap E_k] = p_j \cdot p_k$. Thus, by the principle of inclusion-exclusion, for $1 \leqq l \leqq d(i)$,

$$\Pr\left[\bigcup_{j=1}^{d(i)} E_j \right] \geqq \Pr\left[\bigcup_{j=1}^{l} E_j \right] \geqq \sum_{j=1}^{l} p_j - \sum_{j=1}^{l} \sum_{k=j+1}^{l} p_j \cdot p_k.$$

Let $\alpha = \sum_{j=1}^{d(i)} p_j$. The technical lemma which follows implies that

$$\Pr\left[\bigcup_{j=1}^{d(i)} E_j \right] \geqq \frac{1}{2} \cdot \min\{\alpha, 1\} \geqq \frac{1}{2} \cdot \min\left\{ \frac{\text{sum}(i)}{2}, 1 \right\}.$$

It follows that $\Pr[i \in N(I')] \geqq \frac{1}{4} \cdot \min\{\text{sum}(i)/2, 1\}$. □

TECHNICAL LEMMA. *Let $p_1 \geqq \cdots \geqq p_n \geqq 0$ be real-valued variables. For $1 \leqq l \leqq n$,* let

$$\alpha_l = \sum_{j=1}^{l} p_j, \quad \beta_l = \sum_{j=1}^{l} \sum_{k=j+1}^{l} p_j \cdot p_k, \quad \gamma_l = \alpha_l - c \cdot \beta_l,$$

where $c > 0$ is a constant. Then

$$\max\{\gamma_l \mid 1 \leqq l \leqq n\} \geqq \tfrac{1}{2} \cdot \min\left\{ \alpha_n, \frac{1}{c} \right\}.$$

Proof. It can be shown by induction that β_l is maximized when $p_1 = \cdots = p_n = \alpha_l / l$, and consequently $\beta_l \leqq \alpha_l^2 \cdot (l-1)/2l$. Thus,

$$\gamma_l \geqq \alpha_l \cdot \left(1 - c \cdot \alpha_l \cdot \frac{(l-1)}{2l} \right).$$

If $\alpha_n \leqq 1/c$ then $\gamma_n \geqq \alpha_n/2$. If $\alpha_1 \geqq 1/c$ then $\gamma_1 \geqq 1/c$. Otherwise, $\exists l$, $1 < l < n$ such that $\alpha_{l-1} \leqq 1/c \leqq \alpha_l \leqq 1/c \cdot l/(l-1)$. The last inequality follows because $p_1 \geqq \cdots \geqq p_n$. Then, $\gamma_l \geqq 1/2c$. □

4. Removing randomization from parallel algorithms.

4.1. Overview. This section outlines a general strategy for converting fast parallel Monte Carlo algorithms into fast parallel deterministic algorithms. Algorithm B is

converted into a very simple and fast deterministic algorithm using this strategy. The general strategy contains several ideas that were discovered and used by previous authors. The relationship of the general strategy to other work is discussed in § 5.

To use the general strategy to convert a specific parallel Monte Carlo algorithm into a deterministic algorithm, it must be possible to describe the Monte Carlo algorithm in the following way. All of the randomization is incorporated in one step of the algorithm called the **choice step** (which may be executed more than once during the course of the algorithm). In the **choice step**, values for random variables $X_0, \cdots, X_{n-1}$ are chosen mutually independently, such that on the average a set of values for these random variables is **good**. The algorithm can determine very quickly whether or not a set of values is **good** after the execution of the **choice step**. The analysis of the algorithm shows that if at each execution of the **choice step** a **good** set of values are chosen, then the algorithm outputs a correct output within a specified time bound.

To be able to convert a Monte Carlo algorithm which fits the above description into a deterministic algorithm, the following additional criteria are sufficient.

1. Let r be a positive integer and let $q \geqq n$ be a prime number, such that both r and q are bounded by a polynomial in n. The set of random variables $X_0, \cdots, X_{n-1}$ can be modified so that the range of random variable X_i is $R = \{R_1, \cdots, R_r\}$, such that X_i takes on value R_j with probability n_{ij}/q, where n_{ij} is an integer greater than or equal to zero and $\sum_{j=1}^{r} n_{ij} = q$.

2. The analysis of the algorithm can be modified to show that if $X_0, \cdots, X_{n-1}$ are only *pairwise independent* [Fr] then with positive probability a random set of values for $X_0, \cdots, X_{n-1}$ is **good**.

The deterministic algorithm is the same as the Monte Carlo algorithm except that the **choice step** is simulated by the following: Construct the probability space described in § 4.2 with q^2 sample points, where each sample point corresponds to a set of values of $X_0, \cdots, X_{n-1}$. In parallel, spawn q^2 copies of the algorithm, one for each sample point, and test to see which sample points are **good**. Use the set of values for $X_0, \cdots, X_{n-1}$ corresponding to a **good** sample point as the output from the **choice step**.

Since the analysis shows that with positive probability a random sample point is **good**, at least one of the sample points must be **good**. Since at each **choice step** a **good** set of values for $X_0, \cdots, X_{n-1}$ is used, the algorithm is deterministic and is guaranteed to run within the specified time bound.

This strategy for converting a Monte Carlo algorithm into a deterministic algorithm can be generalized by relaxing criterion 2 above to allow d-wise independence for any integer constant $d \geqq 1$.

4.2. Generating pairwise independent random variables. Let $X_0, \cdots, X_{n-1}$ be a set of random variables satisfying the first criterion stated in § 4.1. In this section a probability space of q^2 sample points is generated, where the random variables are pairwise independent. In contrast, the number of sample points in a probability space where the random variables are mutually independent is $\Omega(2^n)$, assuming each random variable takes on at least two values with a nonzero probability.

Consider an n by q matrix A. The values in row i of A correspond to the possible values for random variable X_i. Row i of A contains exactly n_{ij} entries equal to R_j. Let $0 \leqq x, y \leqq q-1$. The sample space is the collection of q^2 sample points

$$b^{x,y} = (b_0, \cdots, b_{n-1}),$$

where $b_i = A_{i,(x+y \cdot i) \bmod q}$ is the value of X_i at sample point $b^{x,y}$. The probability assigned to each sample point is $1/q^2$.

LEMMA 1. $\Pr[X_i = R_j] = n_{ij}/q$.

Proof. For fixed l, there are exactly q pairs of x, y such that $(x+y\cdot i) \bmod q = l$. There are n_{ij} values of l such that $A_{i,l} = R_j$. □

LEMMA 2. $\Pr[X_i = R_j \text{ and } X_{i'} = R_{j'}] = (n_{ij}\cdot n_{i'j'})/q^2$.

Proof. For fixed l and l', there is exactly one x, y pair such that $(x+y\cdot i) \bmod q = l$ and $(x+y\cdot i') \bmod q = l'$ simultaneously. □

Lemma 2 shows that the random variables $X_0, \cdots, X_{n-1}$ are pairwise independent in the probability space.

4.3. Reanalysis of Algorithm B assuming pairwise independence. In this section Algorithm B is analyzed assuming events are only pairwise independent. More specifically, assume that the collection of random variables $\{\text{coin}(i) \mid i \in V'\}$ are only pairwise independent. $\forall i \in V'$, let E_i be the event that coin $(i) = 1$ and let

$$p_i = \Pr[E_i] = \frac{1}{2d(i)}.$$

The analogue of Lemma B is the following.

LEMMA C. $\Pr[i \in N(I')] \geqq \frac{1}{8}\cdot \min\{\text{sum}(i), 1\}$.

Proof. The notation introduced in the proof of Lemma B is retained. Let $\alpha_0 = 0$ and for $1 \leqq l \leqq d(i)$, let $\alpha_l = \sum_{j=1}^{l} p_j$. As in the proof of Lemma B,

$$\Pr[i \in N(I')] \geqq \sum_{j=1}^{d(i)} \Pr[E_j'] \Pr[A_j \mid E_j'].$$

A lower bound is first derived on $\Pr[A_j \mid E_j']$. $\Pr[A_j \mid E_j'] = 1 - \Pr[\neg A_j \mid E_j']$. But,

$$\Pr[\neg A_j \mid E_j'] \leqq \sum_{\substack{v \in \text{adj}(j) \\ d(v) \geqq d(j)}} \Pr[E_v \mid E_j']$$

and

$$\Pr[E_v \mid E_j'] = \frac{\Pr[E_v \cap \neg E_1 \cap \cdots \cap \neg E_{j-1} \mid E_j]}{\Pr[\neg E_1 \cap \cdots \cap \neg E_{j-1} \mid E_j]}.$$

The numerator is less than or equal to $\Pr[E_v \mid E_j] = p_v$. The denominator is equal to

$$1 - \Pr\left[\bigcup_{l=1}^{j-1} E_l \mid E_j\right] \geqq 1 - \sum_{l=1}^{j-1} \Pr[E_l \mid E_j] = 1 - \alpha_{j-1}.$$

Thus, $\Pr[E_v \mid E_j'] \leqq p_v/(1-\alpha_{j-1})$. Consequently,

$$\Pr[\neg A_j \mid E_j'] \leqq \sum_{\substack{v \in \text{adj}(j) \\ d(v) \geqq d(j)}} \frac{p_v}{1-\alpha_{j-1}} \leqq \frac{1}{2(1-\alpha_{j-1})}.$$

Thus,

$$\Pr[A_j \mid E_j'] \geqq 1 - \frac{1}{1(1-\alpha_{j-1})} = \frac{1-2\alpha_{j-1}}{2(1-\alpha_{j-1})}.$$

Now, a lower bound is derived on $\Pr[E_j']$.

$$\begin{aligned}\Pr[E_j'] &= \Pr[E_j] \Pr[\neg E_1 \cap \cdots \cap \neg E_{j-1} \mid E_j] \\ &= p_j\left(1 - \Pr\left[\bigcup_{l=1}^{j-1} E_l \mid E_j\right]\right) \geqq p_j(1-\alpha_{j-1}).\end{aligned}$$

Thus, for $1 \leqq l \leqq d(v)$ and $\alpha_l < \frac{1}{2}$,

$$\Pr[i \in N(I')] \geqq \sum_{j=1}^{l} \frac{p_j(1-2\alpha_{j-1})}{2} = \frac{1}{2} \cdot \left(\sum_{j=1}^{l} p_j - 2 \cdot \sum_{j=1}^{l} \sum_{k=j+1}^{l} p_j \cdot p_k \right).$$

By the technical lemma,

$$\Pr[i \in N(i')] \geqq \tfrac{1}{4} \cdot \min\{\alpha_{d(i)}, \tfrac{1}{2}\} \geqq \tfrac{1}{8} \cdot \min\{\text{sum}(i), 1\}. \qquad \square$$

The analogue to Theorem 1 for Algorithm B when the random variables are only pairwise independent is:

THEOREM 2. *$E[Y_k^B - Y_{k+1}^B] \geqq \frac{1}{16} Y_k^B$ when the random variables $\{\text{coin}(i) \mid i \in V'\}$ are only pairwise independent.*

Proof. Use Lemma C in place of Lemma B in the proof of Theorem 1. $\square$

This analysis shows that Algorithm B almost fulfills the criteria stated in § 4.1. The range of the set of random variables $\{\text{coin}(i) \mid i \in V'\}$ is $\{0, 1\}$. A **good** set of values for $\{\text{coin}(i) \mid i \in V'\}$ is a set of values such that when they are used in the **choice step** at least $\frac{1}{16}$ of the edges in E' are eliminated. Theorem 2 implies that in any probability space where the random variables $\{\text{coin}(i) \mid i \in V'\}$ are pairwise independent there is at least one sample point which is **good**. To determine whether or not a set of values for $\{\text{coin}(i) \mid i \in V'\}$ is **good**, the steps in the body of the **while** loop are executed using these values and the number of edges eliminated is computed. The only requirement that is not fulfilled is criterion 1.

4.4. Algorithms C and D for the MIS problem. In this section, the deterministic implementation of Algorithm B is presented. The only missing requirement is that the set of random variables $\{\text{coin}(i) \mid i \in V'\}$ as defined in § 3.3 does not fulfill criterion 1 of § 4.1. To fulfill this criterion there are two changes: (1) the probabilities assigned to the random variables $\{\text{coin}(i) \mid i \in V'\}$ are modified slightly, and (2) the algorithm is modified slightly.

The probabilities are modified as follows. Let q be a prime number between n and $2n$. The probability that $\text{coin}(i) = 1$ is $p_i' = \lfloor p_i \cdot q \rfloor / q$, where $p_i = 1/2d(i)$ is the previous probability.

Here is a description of the **choice step** for Algorithm C.

Case 1. There is a vertex $i \in V'$ such that $d(i) \geqq n/16$. The algorithm sets I' to $\{i\}$.

Case 2. $\forall i \in V'$, $d(i) < n/16$. The algorithm constructs the probability space described in § 4.2 and randomly chooses a sample point. The algorithm uses the values for $\{\text{coin}(i) \mid i \in V'\}$ corresponding to the sample point to choose I' as before.

Here is the analysis for Algorithm C. Case 1 can occur at most 16 times, because each time it occurs at least $\frac{1}{16}$ of the vertices in the original graph G are eliminated. In Case 2, $\forall i \in V'$, $d(i) < n/16$, which implies $q/2d(i) \geqq n/2d(i) > 8$ and consequently $p_i' \geqq 8/q$. But this implies that $\lfloor p_i \cdot q \rfloor / 8 \geqq 1$. Thus, since $(\lfloor p_i \cdot q \rfloor + 1)/q \geqq p_i$, $\frac{8}{9} p_i \leqq p_i' \leqq p_i$.

LEMMA D. *Let the set of random variables $\{\text{coin}(i) \mid i \in V'\}$ be pairwise independent such that $\Pr[\text{coin}(i) = 1] = p_i'$ and such that $\forall i \in V'$, $d(i) < n/16$. Then $\Pr[i \in N(I')] \geqq \frac{1}{9} \min\{\text{sum}(i), 1\}$.*

Proof. The same proof as for Lemma C, except $\frac{8}{9} p_i$ is used as a lower bound and p_i is used as an upper bound on $\Pr[\text{coin}(i) = 1]$. $\square$

THEOREM 3. *Consider Algorithm* B, *where the random variables $\{\text{coin}(i) \mid i \in V'\}$ are as described in this section and where $\forall i \in V'$, $d(i) < n/16$. Then $E[Y_k^B - Y_{k+1}^B] \geqq \frac{1}{18} Y_k^B$.*

Proof. Use Lemma D in place of Lemma B in the proof of Theorem 1. □

Thus, if Case 2 is true then Algorithm C eliminates at least $\frac{1}{18}$ of the edges in E' on the average. From this it is easy to see that the bounds on the running time and on the number of random bits used by Algorithm C hold with high probability.

The code for Algorithm C follows. This algorithm is very practical because it is simple, fast, uses a small number of processors and a small number of random bits. Each execution of the body of the **while** loop can be implemented in $O(\log n)$ time on a EREW P-RAM using $O(m)$ processors, where the expected number of random bits used is $O(\log n)$. The expected number of executions of the **while** loop before termination of the algorithm is $O(\log n)$. Thus, the expected running time of the entire algorithm on a EREW P-RAM is $O((\log n)^2)$ using $O(m)$ processors, where the expected number of random bits used is $O((\log n)^2)$.

Algorithm D is the same as Algorithm C, except that the **choice step** is executed by testing in parallel all q^2 sets of values for $\{\text{coin}(i) \mid i \in V'\}$ and using the set of values which maximizes the number of edges eliminated from G'. Theorem 3 shows that the best set will eliminate at least $\frac{1}{18}$ of the edges in the graph G'. This algorithm can be implemented on a EREW P-RAM with $O(mn^2)$ processors with running time $O((\log n)^2)$. The number of executions of the **while** loop is guaranteed to be at most

$$\frac{\log(n^2)}{\log(18/17)} + 16 \leqq 25 \cdot \log n + 16.$$

begin
  $I \leftarrow \varnothing$
  compute $n = |V|$
  compute a prime q such that $n \leqq q \leqq 2n$
  $G' = (V', E') \leftarrow G = (V, E)$
  while $G' \neq \varnothing$ **do**
  begin
    In parallel, $\forall i \in V'$, compute $d(i)$
    In parallel, $\forall i \in V'$
      if $d(i) = 0$ **then** add i to I and delete i from V'.
    find $i \in V'$ such that $d(i)$ is maximum
    if $d(i) \geqq n/16$ **then** add i to I and let G' be the
      graph induced on the vertices $V' - (\{i\} \cup N(\{i\}))$
    else $(\forall i \in V', d(i) < n/16)$
    begin
(choice) **randomly choose** x and y such that $0 \leqq x, y \leqq q-1$
      $X \leftarrow \varnothing$
      In parallel, $\forall i \in V'$,
      begin
        compute $n(i) = \lfloor q/2d(i) \rfloor$
        compute $l(i) = (x + y \cdot i) \bmod q$
        if $l(i) \leqq n(i)$ **then** add i to X
      end
      $I' \leftarrow X$
      In parallel, $\forall i \in X, j \in X$,
        if $(i, j) \in E'$ **then**
          if $d(i) \leqq d(j)$ **then** $I' \leftarrow I' - \{i\}$
          else $I' \leftarrow I' - \{j\}$

$I \leftarrow I \cup I'$
$Y \leftarrow I' \cup N(I')$
$G' = (V', E')$ is the induced subgraph on $V' - Y$.
end
end
end

5. A history of the ideas used in the general strategy. Some of the ideas outlined in § 4 have been developed in other contexts. The purpose of this section is threefold:

1. Give proper credit to previous researchers.
2. Indicate the innovations in § 4 by contrasting this work with previous work.
3. Highlight techniques common to a large body of seemingly unrelated work.

The last point may be the most important: fragments of these techniques have been used successfully many times in different contexts by many authors. By making the connections between these techniques explicit, a more unified approach to problem solving using these techniques may evolve.

There are many randomizing algorithms in the computer science literature. These algorithms generally use random input bits to generate in a straightforward way a number of mutually independent random variables. One of the general strategies used in this paper consists of two parts:

(1) Prove that the necessary probability theorems still hold when the random variables are pairwise independent instead of mutually independent (the proof itself is not always straightforward, e.g. the proof in this paper).
(2) Construct a sample space with special structural properties where the random variables are pairwise independent (constructing the same space with the necessary structural properties is usually quite easy, given the plethora of previous research in this area which is discussed in §§ 5.1 and 5.2).

The reason for doing this is that the special structural properties of the sample space can be used to great advantage. Previous papers which exploit this strategy are [ACGS], [CW], [Si], [Sh]. In this paper, the special structural property of the sample space is that all of the sample points can be constructed efficiently in parallel. This property is used to deterministically find a good sample point quickly in parallel.

5.1. Generating *d*-wise independent random variables. The construction given in § 4.2 was subsequently generalized to a probability distribution where the random variables $X_0, \cdots, X_{n-1}$ are d-wise independent, where there are q^d sample points for any integer constant $d \geqq 1$. Credit for this generalization goes to Noga Alon [Al], Richard Anderson [An] and Paul Beame [Be], each of whom found it independently.

Let $0 \leqq x_0, \cdots, x_{d-1} \leqq q-1$. The sample space is the collection of q^d sample points

$$b^{x_0, \cdots, x_{d-1}} = (b_0, \cdots, b_{n-1}),$$

where b_i is the $(i, (\sum_{j=0}^{d-1} x_j \cdot i^j) \bmod q)$ entry in matrix A. The probability assigned to each sample point is $1/q^d$. It is possible to show that the random variables $X_0, \cdots, X_{n-1}$ are d-wise independent and that $\Pr[X_i = R_j] = n_{ij}/q$. [CFGHRS] proves that any probability space with n d-wise independent random variables must contain at least $n^{d/2}$ sample points.

5.2. History of *d*-wise independence constructions. The first person to give an example of random variables which are pairwise independent but not mutually independent is Bernstein in 1945 (see [Fr, p. 126]). His example consists of three $\{0, 1\}$ valued random variables. Lancaster [La] generalizes this example to $n-1$ pairwise independent

variables on n sample points, giving constructions involving the Hadamard matrix and Latin squares. Joffe [Jo1], [Jo2] gives a construction for generating d-wise independent random variables which is exactly the same as the constructions given in §§ 4.2 and 5.1, except as noted below. O'Brien [OB] discusses generating pairwise independent random variables with additional properties. Incomplete Block Designs (see [Ha]) are another way to generate pairwise independent random variables.

All of the work mentioned in the preceding paragraph concentrates solely on constructions of pairwise (d-wise) independent random variables such that each value of each random variable is equally likely. The constructions given in §§ 4.2 and 5.1 generalizes this slightly: the random variables are allowed to take on different values with different probabilities.The freedom to specify nonequal probabilities for different values of the random variables is essential to the general strategy described in § 4.

5.3. Techniques for analyzing d-wise independent random variables. One of the important features of the general strategy given in § 4 is the separation of the analysis of d-wise independent random variables from the construction of the probability space. The modularity of this approach allows the researcher to study the properties of d-wise independent random variables without worrying about the details of the probability space construction. A short list of techniques for analyzing d-wise independent random variables is given here.

1. Markov's inequality holds for random variables assuming no independence.
2. Chebyshev's inequality holds for pairwise independent random variables.
3. Bonferroni bounds and stronger similar bounds [CE], [Ga] are useful. These types of bounds are used extensively in the proof of the technical lemma, which is similar to a lemma contained in [CE].
4. Use of the d-wise independence in a natural way to divide a complicated expression into simpler expressions. For example, $E[A|B]=E[A]$ if A and B are pairwise independent random variables.
5. The pigeon-hole principle. For example, if it can be proved that $E[X]>c$ assuming d-wise independent random variables, where X is a real-valued random variable which is possibly a function of several of the original random variables and c is a constant, then there must be a sample point where $X>c$.

The proofs of Lemmas A, B, C and D use a rich mixture of these techniques.

5.4. Other applications of d-wise independent random variables. In this section, some of the uses of d-wise independent random variables are highlighted. A complete history is beyond the scope of this paper.

Carter and Wegman [CW] use constructions for pairwise independent random variables to generate hashing functions with certain desirable properties. One of the constructions they use is very similar to that discussed in [Jo2], but not quite as general as the construction given in § 5.1. They are perhaps the first to separate the analysis of the algorithm from the construction used to generate the probability space. The techniques they use to analyze the algorithm include techniques 1, 3 and 4 from the previous section. The work of [CW] is used by Sipser to simplify the proof in [Si] that BPP is at the second level of the polynomial time hierarchy. Similar ideas appear in [St].

Alexi, Chor, Goldreich and Schnorr [ACGS] prove that RSA/Rabin bits are secure. They use the same construction as given in [Jo2] to generate pairwise independent random variables and they use Chebyshev's inequality in their analysis. Chor and Goldreich [CG] have a simple solution to a problem originally introduced and solved in a more complicated way by Karp and Pippenger [KP]. The solution of [CG] uses the same construction as given in [Jo2] to generate pairwise independent random

variables and Chebyshev's inequality in the analysis. Shamir [Sh] uses the construction given in [Jo2] to distribute a secret among n people in such a way that no subset of d people can determine anything about the secret, but any subset of $d+1$ people can completely determine the secret.

Karp and Wigderson [KW] are the first to introduce a special case of the strategy given in § 4. They use their strategy to convert their Monte Carlo MIS algorithm into a deterministic algorithm. They use an incomplete block design to construct a probability space with $\{0, 1\}$ valued pairwise independent random variables, where each random variable takes on value 1 with probability $\frac{1}{2}$. Their analysis of the algorithm, which is heavily dependent upon their particular construction of the probability space, is quite complicated. There is a simpler analysis of their algorithm, which uses the techniques in § 5.3, which shows that the algorithm still works well on the average if the random variables are only pairwise independent. This analysis, together with the construction given in § 4.2, where random variables can take on different values with different probabilities, gives a much simpler and faster deterministic algorithm than the one they give. However, the algorithm is still not as simple nor as fast as those presented in this paper.

6. Generalizations of the MIS problem.

6.1. The Maximal Coloring problem. The Maximal Coloring problem generalizes the MIS problem. The input to the Maximal Coloring problem is an undirected graph $G=(V, E)$ and a set of colors C_v for each vertex $v \in V$. The output is a maximal coloring. In a maximal coloring, each vertex v is either assigned a color from C_v or is not assigned a color, subject to the restrictions that no two adjacent vertices are assigned the same color and that if v is not assigned a color then each color in C_v must be assigned to at least one neighbor of v. The MIS problem is a special case of the Maximal Coloring problem where $C_v = \{\text{red}\}$ for each vertex $v \in V$. The set of vertices colored red in any maximal coloring are a MIS.

Another problem which the Maximal Coloring problem generalizes is the $\Delta+1$VC problem. The input to the $\Delta+1$VC problem is an undirected graph $G=(V, E)$. Let Δ be the maximum degree of any vertex in V, let $\Delta' = \Delta+1$ and let $C = \{c_1, \cdots, c_{\Delta'}\}$ be a set of distinct colors. The output is an assignment of a color from C to each vertex such that no two adjacent vertices are assigned the same color. The $\Delta+1$VC problem is the special case of the Maximal Coloring problem where for each vertex $v \in V$, $C_v = C$. In any Maximal Coloring each vertex will be assigned some color from C because Δ' is larger than $d(v)$. The obvious sequential algorithm for the $\Delta+1$VC problem follows: For $v = 1, \cdots, n$, vertex v is assigned the smallest indexed color from C which is not assigned to a smaller indexed adjacent vertex. One might hope to devise a fast parallel algorithm for the $\Delta+1$VC problem by emulating the sequential algorithm. However, this is unlikely since

LEMMA 3. *The problem of deciding what color vertex n is assigned by the above sequential algorithm is* NC^1 *complete for P.*

Proof. There is an easy reduction from the LFMIS problem (see § 1) to this problem. □

Thus, as was the case for the MIS problem, the coloring algorithm cannot simply emulate the sequential algorithm.

There is a logspace reduction from the Maximal Coloring problem to the MIS problem. Given a Maximal Coloring problem with input $G=(V, E)$ and color sets $\{C_v\}$, a new graph G' is formed. The vertices in G' are $V' = \{(v, c): v \in V \text{ and } c \in C_v\}$.

The edges in G' are

$$E' = \{((v, c_1), (v, c_2)) : v \in V \text{ and } c_1, c_2 \in C_v\} \cup \{((v, c), (w, c)) : (v, w) \in E \text{ and } c \in C_v \cap C_w\}.$$

There is a one-to-one correspondence between maximal colorings in G and maximal independent sets in G'. This reduction together with the MIS algorithm shows that the Maximal Coloring problem is in NC^2.

The ΔVC problem is to color all the vertices using only Δ distinct colors. Brooks' theorem [Br] proves that all but very special graphs can be colored with Δ colors, and implicitly gives a polynomial time sequential algorithm. Karloff, Shmoys and Soroker [KSS] have found a NC parallel algorithm for the ΔVC problem when Δ is polylog in the number of vertices. Their algorithm uses the algorithm for the $\Delta+1$VC problem as a subroutine. The classification of the ΔVC problem with respect to parallel computation is still open for unrestricted Δ.

6.2. Binary coherent systems. Recently, researchers in Artificial Intelligence have been actively investigating various connectionist models of the brain [Fe], [Go], [Hi], [Ho]. Some of the basic features of the connectionist model are shared by knowledge representation schemas [Ts].

One particular model of the brain is a binary coherent system [Ho], [Hi]. The binary coherent system (BCS) problem, which is studied by Hopfield [Ho], can be formally stated as follows. The input is an undirected graph $G = (V, E)$ together with a real-valued weight w_e for each edge and a real-valued threshold t_v for each vertex. Each vertex v has a state s_v which can be either -1 or 1. The state of the system is a tuple $(s_1, \cdots, s_{|V|})$. The energy of vertex v in a system state is

$$s_v \cdot \left(t_v + \sum_{e=(v,z)\in E} w_e \cdot s_z \right).$$

The output is a system state where all vertices have energy greater than or equal to zero. The BCS problem has a polynomial time sequential algorithm if all of the weights and thresholds are input in unary. The algorithm repeats the following step until all vertices have energy greater than or equal to zero: find a vertex with negative energy and flip its state. The running time of this algorithm is slow if the system is large. Hopfield suggests a simple asynchronous parallel algorithm for this problem, but provides no formal analysis of its running time, although he does give some empirical evidence that it is fast. An open question is whether or not the BCS problem has an NC algorithm.

The MIS problem is a special case of the BCS problem, where all edge weights are -1 and for each $v \in V$, $t_v = -d(v) + 1$. Thus, Algorithm D shows that at least a special case of the BCS problem is in NC, and Baruch Awerbuch has observed that Algorithm A can be easily modified to run asynchronously in parallel.

Another natural problem which is a special case of the BCS problem is the Different Than Majority Labelling (DTML) problem. The input to this problem is an undirected graph $G = (V, E)$. The output is a label of -1 or 1 for each vertex v such that at least half of the neighbors of v have the opposite label. The DTML problem is a BCS problem where all thresholds are zero and all edge weights are -1. The DTML problem may also be viewed as a graph partition problem: partition the vertices of an undirected graph into two sets such that for each vertex v at least half of the edges out of v cross the partition. (Using the techniques developed in this paper, there is a fast parallel algorithm for an easier graph partition problem which can be stated as follows: partition

the vertices of an undirected graph into two sets such that at least half of the edges in the graph cross the partition.) Karloff [Kf2] has found a NC algorithm for this problem when the input is a cubic graph, but the general problem is still open. However, there is evidence that a different type algorithm than the MIS algorithm will have to be found.

THEOREM 4. *The problem of deciding whether there is a* DTML *for a graph such that two specified vertices receive the same label is* NP-complete. □

This theorem gives evidence that no fast algorithm for the DTML problem can permanently decide the labels of vertices in a local manner in the same way as is the case for the MIS algorithm.

7. Open problems and further work.

1. Find other Monte Carlo algorithms for which the techniques developed in this paper are applicable for converting the algorithm into a deterministic algorithm.

2. Develop probabilistic bounds on random variables that are d-wise independent.

3. Algorithm A has a local property which seems particularly well suited to distributed computing networks where the processors can only communicate with neighboring processors. Find applications for this algorithm in distributed computing protocols.

4. There is no known lower bound on the parallel complexity of the MIS problem. Either find a problem which is complete in some complexity class (like NL) and reduce it to the MIS problem, or else find a faster MIS algorithm.

Acknowledgments. I thank Paul Beame for extensive discussions which significantly simplified the analysis of the algorithms. Paul also made numerous suggestions which substantially streamlined the presentation. I thank Stephen Cook for many discussions about the MIS problem and related problems, and for his unswerving belief that there must be a faster algorithm for the MIS problem. Thanks go to both Charlie Rackoff and Gary Miller for helpful discussions about the MIS problem, and more specifically for suggesting that an analysis be done on an algorithm very similar to the algorithm described in § 3.2. I thank Allan Borodin for introducing me to the work on connectionist models. I thank both Paul Beame and Richard Anderson for pointing out some of the references discussed in § 5.

REFERENCES

[Al] N. ALON, private communication.

[An] R. ANDERSON, private communication.

[ACGS] W. ALEXI, B. CHOR, O. GOLDREICH AND C. SCHNORR, *RSA/Rabin bits are* $\frac{1}{2}+1/\text{poly}(\log N)$ *secure*, Preliminary version in Proc. 25th IEEE Symposium on Foundations of Computer Science, October 1984; this Journal, to appear.

[ABI] N. ALON, L. BABAI AND A. ITAI, *A fast and simple randomized parallel algorithm for the maximal independent set problem*, private communication.

[Be] P. BEAME, private communication.

[Br] R. L. BROOKS, *On colouring the nodes of a network*, Proc. Cambridge Philos. Soc., 37 (1941), pp. 194–197.

[CE] K. CHUNG AND P. ERDOS, *On the application of the Borel–Cantelli lemma*, Amer. Math. Soc., 72 (1952), pp. 179–186.

[CFGHRS] B. CHOR, G. FRIEDMAN, O. GOLDREICH, J. HASTAAD, S. RUDICH, AND R. SMOLANSKI, *The bit extraction problem*, preprint, accepted to Proc. 26th IEEE Symposium on Foundations of Computer Science.

[CG] B. CHOR AND O. GOLDREICH, *On the power of two-points based sampling*, technical report, MIT Laboratory for Computer Science, Cambridge, MA.

[Co] S. COOK, *A taxonomy of problems with fast parallel algorithms*, Information and Control, Vol. 64, Nos. 1–3, January/February/March 1985, Academic Press, New York.

[CL] S. COOK AND M. LUBY, *A fast parallel algorithm for finding a truth assignment to a* 2-CNF *formula*, in preparation.

[CW] J. CARTER AND M. WEGMAN, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.

[Fe] J. A. FELDMAN, *A connectionist model of visual memory*, Parallel Models of Associative Memory, G. E. Hinton and J. A. Anderson, eds., Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.

[Fr] W. FELLER, *An Introduction to Probability Theory and Its Applications*, Vol. 1, 3rd edition, John Wiley, New York, 1968.

[FW] S. FORTUNE AND J. WYLLIE, *Parallelism in random access machines*, Proc. Tenth ACM Symposium on Theory of Computing, 1978, pp. 114–118.

[Ga] J. GALAMBOS, *Bonferroni inequalities*, Ann. Probab., 5 (1977), pp. 577–581.

[Go] L. M. GOLDSCHLAGER, *A computational theory of higher brain function*, Technical Report, Stanford Univ. Stanford, CA, April 1984.

[Ha] M. HALL, Jr., *Combinatorial Theory*, Blaisdell, Waltham, MA, 1967.

[Ho] J. J. HOPFIELD, *Neural networks and physical system with emergent collective computational abilities*, Proc. National Acad. Sci., 79 (1982), pp. 2554–2558.

[Hi] G. E. HINTON, T. SEJNOWSKI AND D. ACKLEY, *Boltzmann machines: constraint satisfaction networks that learn*, Technical Report CMU-CS-84-119, Carnegie-Mellon Univ., Pittsburgh, PA.

[II] A. ISRAELI AND A. ITAI, *A fast and simple randomized parallel algorithm for maximal matching*, Computer Science Dept., Technion, Haifa, Israel, 1984.

[IS] A. ISRAELI AND Y. SHILOACH, *An improved maximal matching parallel algorithm*, Technical Report 333, Computer Science Dept., Technion, Haifa, Israel, 1984.

[Jo1] A. JOFFE, *On a sequence of almost deterministic pairwise independent random variables*, Proc. Amer. Math. Soc., 29 (1971), pp. 381–382.

[Jo2] ———, *On a set of almost deterministic k-independent random variables*, Ann. Probab., 2 (1974), pp. 161–162.

[Kf1] H. KARLOFF, *Randomized parallel algorithm for the odd-set cover problem*, preprint.

[Kf2] ———, private communication.

[KP] R. KARP AND N. PIPPENGER, *A time-randomness tradeoff*, presented at the AMS Conference on Probabilistic Computational Complexity, Durham, NH, 1982.

[KSS] H. KARLOFF, D. SHMOYS AND D. SOROKER, *Efficient parallel algorithms for graph coloring and partitioning problems*, preprint.

[KW] R. M. KARP AND A. WIGDERSON, *A fast parallel algorithm for the maximal independent set problem*, Proc. 16th ACM Symposium on Theory of Computing, 1984, pp. 266–272.

[KUW] R. M. KARP, E. UPFAL AND A. WIGDERSON, *Constructing a perfect matching is in random* NC, Proc. 17th ACM Symposium on Theory of Computing, 1985, pp. 22–32.

[La] H. LANCASTER, *Pairwise statistical independence*, Ann. Math. Statist., 36 (1965), pp. 1313–1317.

[Le] G. LEV, *Size bounds and parallel algorithms for networks*, Report CST-8-80, Dept. Computer Science, Univ. Edinburgh, 1980.

[Lu] M. LUBY, *A simple parallel algorithm for the maximal independent set problem*, Proc. 17th ACM Symposium on Theory of Computing, 1985, pp. 1–10.

[OB] G. O'BRIEN, *Pairwise independent random variables*, Ann. Probab., 8 (1980), pp. 170–175.

[R] A. RÉNYI, *Probability Theory*, North-Holland, Amsterdam, 1970.

[Si] M. SIPSER, *A complexity theoretic approach to randomness*, Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 330–335.

[Sh] A. SHAMIR, *How to share a secret*, Comm. ACM, 22 (1979), pp. 612–613.

[St] L. STOCKMEYER, *The complexity of approximate counting*, Proc. 15th ACM Symposium on Theory of Computing, 1983, pp. 118–126.

[Ts] J. TSOTSOS, *Representational axes and temporal cooperative processes*, Technical Report RBCV-84-2, Univ. Toronto, Toronto, Ontario, April 1984.

[Va] L. G. VALIANT, *Parallel computation*, Proc. 7th IBM Symposium on Mathematical Foundations of Computer Science, 1982.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
013

FINDING A MAXIMUM CLIQUE IN AN ARBITRARY GRAPH*

EGON BALAS† AND CHANG SUNG YU‡

Abstract. We describe a new type of branch and bound procedure for finding a maximum clique in an arbitrary graph $G=(V, E)$. The two main ingredients, both of $O(|V|+|E|)$ time complexity, are (i) an algorithm for finding a maximal triangulated induced subgraph of G; and (ii) an algorithm for finding a maximal k-chromatic induced subgraph of G. We discuss computational experience on randomly generated graphs with up to 400 vertices and 30,000 edges.

Key words. maximum clique, vertex packing, triangulated subgraphs, k-chromatic subgraphs, implicit enumeration

AMS(MOS) subject classifications. 05, 90, 68

1. Introduction.

In this paper we address the problem of finding a maximum cardinality clique in an arbitrary undirected graph. Given a graph $G=(V, E)$ with vertex set V and edge set E, a *complete subgraph* of G is one whose vertices are pairwise adjacent. A *clique* is a maximal complete subgraph. For the sake of brevity, the vertex set of a clique is also called a clique. A *vertex packing* (or *stable set* or *independent set* of vertices) is a vertex set whose elements are pairwise nonadjacent. A *vertex cover* is a set of vertices that cover (i.e., meet) all the edges. The *complement* of a graph $G=(V, E)$ is the graph $\bar{G}:=(V, \bar{E})$, where $\bar{E}:=\{(i, j)\notin E:\ i, j\in V,\ i\neq j\}$. For $S\subset V$, the subgraph of G *induced* by S is $G(S):=(S, (S\times S)\cap E)$.

The following three statements concerning any $S\subset V$ are easily seen (and well known) to be equivalent:

(1) S is the vertex set of a maximum clique in G,
(2) S is a maximum vertex packing in $\bar{G}$,
(3) $V\setminus S$ is a minimum vertex cover in $\bar{G}$.

Accordingly, the three problems of

(1) finding a maximum clique in G,
(2) finding a maximum vertex packing in $\bar{G}$,
(3) finding a minimum vertex cover in $\bar{G}$,

are equivalent. They are also known to be NP-complete. These problems have many applications, a few of which are listed below.

Information retrieval. If the vertices of a graph represent stored pieces of information and the edges describe the interrelations between the pieces, then the problem of retrieving from storage a maximum subset of totally interrelated pieces of information is a maximum clique problem.

Experimental design. If the vertices of a graph represent subsets (blocks) of size n of a set S of treatments in a statistical experiment and the edges designate those pairs of subsets having at most k elements in common, then the problem of finding a

* Received by the editors February 12, 1985, and in revised form September 15, 1985. The research underlying this report was supported by the National Science Foundation under grant ECS-8218181 and the U.S. Office of Naval Research under contract N00014-82-K-0329 NR047-607.

† Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213.

‡ AT&T Bell Laboratories, Murray Hill, New Jersey 07974.

maximum collection of blocks with at most k common elements between any two blocks is a maximum clique problem.

Signal transmission. If the vertices of a graph represent signals and the edges mark those pairs of signals that can be clearly distinguished from each other, then the problem of selecting a maximum set of clearly distinguishable signals is again a maximum clique problem.

Computer vision. A relational structure is a set of elements partitioned into color classes, along with a binary relation between the elements. Given two relational structures, an input A and a reference B, one wants to optimally "match" A to B, i.e., find a substructure of B, to which A comes as "close" as possible. To this end one constructs an association graph G that has a vertex (a, b) for every pair of elements $a \in A$, $b \in B$, belonging to the same color class, and an edge for every pair of vertices (a, b), (c, d) such that the relation between a and c in A is the same as that between b and d in B. Then optimally "matching" A to B amounts to finding a maximum clique in the association graph G.

In each of the above cases, if we adopt definitions for the edges complementary to the above ones, we obtain of course a maximum vertex packing problem.

Earlier work on vertex packing or vertex covering algorithms includes the papers by Nemhauser and Trotter [1975], Balas and Samuelsson [1977], Tarjan and Trojanowski [1977]. The first two of these present branch and bound methods that use information from a partial polyhedral description of the convex hull of feasible points, and were tested on randomly generated graphs with up to 50 vertices and 50% density. The third one describes a recursive algorithm that depends on a somewhat complicated case analysis, but has a worst case bound of $O(2^{n/3})$. For the maximum clique problem itself, we mention the procedures of Bron and Kerbosch [1973], Gerhards and Lindenberg [1979], and Loukakis and Tsouros [1982]. These are implicit enumeration algorithms that use some logical tests to eliminate subproblems. We implemented and tested two of them and used them as benchmarks for the testing of our own procedure.

The approach that we describe in this paper uses the properties of a special class of perfect graphs, known as triangulated or chordal, in which it is easy to find a maximum clique. It also uses the relationship between cliques and vertex colorings of a graph in order to derive bounds on the clique size. It tends to create fewer branches than any of the above methods and is consequently more efficient.

The paper is organized as follows. Section 2 gives the necessary background information on triangulated graphs and outlines our general approach. Sections 3 and 4 describe the two main ingredients of our procedure: algorithms for finding (i) a maximal triangulated induced subgraph, and (ii) a maximal k-chromatic induced subgraph, of a given graph. Section 5 states the procedure in its entirety, while § 6 discusses computational experience.

2. Cliques, colorings and triangulated graphs. A (vertex) *coloring* of a graph G assigns colors to the vertices of G in such a way that no two adjacent vertices get the same color. Thus a coloring is a partitioning of the vertex set of G into independent sets (color classes). The cardinality of a minimum (vertex) coloring is called the *chromatic number* of G. G is *k-chromatic* if its chromatic number is k.

If $G = (V, E)$ is a graph and A is the incidence matrix of maximal vertex packings (independent sets) versus vertices of G, i.e., $A = (a_{ij})$ with $a_{ij} = 1$ if independent set i contains vertex j, $a_{ij} = 0$ otherwise, then the maximum clique and the minimum coloring

problems can be stated as

$$\text{(P1)} \qquad \begin{aligned} \omega(G) = \max\ & 1x \\ & Ax \leqq 1 \\ & x_j \in \{0, 1\} \quad \forall j \end{aligned}$$

and

$$\text{(P2)} \qquad \begin{aligned} \gamma(G) = \min\ & 1y \\ & A^T y \geqq 1 \\ & y \in \{0, 1\} \quad \forall i, \end{aligned}$$

respectively. Since the relaxation of the integrality conditions on x and y turns (P1) and (P2) into a pair of dual linear programs, clearly $\omega(G) \leqq \gamma(G)$ always holds. G is called *perfect* if $\omega(G') = \gamma(G')$ for all induced subgraphs G' of G. The perfect graph theorem (conjectured by Berge [1961] and proved by Lovász [1972]) asserts that G is perfect if and only if $\bar{G}$ is.

When G is perfect, the linear programming relaxations of (P1) and (P2) have integer solutions for arbitrary objective functions (Chvátal [1975]); hence they can be solved in time polynomial in the input. Although this does not imply polynomiality in $n = |V|$, since the matrix A may have a number of rows exponential in n, it has recently been shown by Grötschel, Lovász and Schrijver [1984] that if G is perfect, problem (P1) is solvable in time polynomial in n.

A special class of perfect graphs whose properties make them particularly well suited for our purposes, is the class of triangulated graphs. A graph G is *triangulated* (or *chordal*) if every cycle of G of length at least 4 has a chord (or, equivalently, if G has no holes, i.e., no chordless cycles of length $\geqq 4$). Triangulated graphs have been studied by Dirac [1961], Fulkerson and Gross [1965], Gavril [1972], Rose, Tarjan and Lueker [1976] and others. For an up-to-date survey of their properties see Golumbic [1980].

A vertex v of an arbitrary graph G is called *simplicial* if all vertices adjacent to v are pairwise adjacent among each other, i.e., if v belongs to exactly one clique. An ordering $v_1, \cdots, v_n$ of the vertices of G is called *perfect* if for $k = 1, \cdots, n$, v_k is simplicial in $G(\{v_k, \cdots, v_n\})$. The basic properties of triangulated graphs that we will be using can now be stated as follows:

(i) every triangulated graph has at least two simplicial vertices;
(ii) $G = (V, E)$ is triangulated if and only if V admits a perfect ordering; and
(iii) determining whether G is triangulated requires $O(|V| + |E|)$ steps.

If G is triangulated, then

(iv) every induced subgraph of G is triangulated;
(v) the cardinality of a maximum clique is equal to that of a minimum coloring;
(vi) G has at most $|V|$ cliques; and
(vii) finding a maximum clique and a minimum coloring in G requires $O(|V| + |E|)$ steps.

We use the above properties to give an $O(|V| + |E|)$ algorithm for generating a maximal triangulated induced subgraph (MTIS) $G(T)$ of an arbitrary graph G, and finding a maximum clique of $G(T)$. This algorithm, called TRIANG, is one of the two main ingredients of our procedure. The second ingredient, called COLOR, finds a minimum coloring of $G(T)$, which is of the same cardinality k as the maximum clique; then it extends $G(T)$ by appending vertices to it while maintaining its chromatic number, until it becomes a maximal k-chromatic induced subgraph of G.

Let this subgraph be $G(W)$. If $G(W)=G$, we are done: the maximum clique found in $G(T)$ is maximum in G. Otherwise we branch, based on the principle that any clique larger than the current largest one must contain one of the vertices in $V\setminus W$. The branching thus replaces the current problem by a set of new subproblems, each one defined on a vertex set contained in the neighbor set of some $v\in V\setminus W$. The procedure, or part of it, is then applied to each new subproblem, with variations to be discussed after we have described the main subroutines.

3. Finding a MTIS and its maximum clique. Determining whether a graph $G=(V, E)$ is triangulated can be done most efficiently by checking whether G admits a perfect ordering. This can be done by applying n times, for $k=1, \cdots, n$, the following step:

Find a simplicial vertex in $G(V\setminus\{w_1, \cdots, w_{k-1}\})$. If none exists, stop: G is not triangulated. Otherwise call the new vertex w_k and set $k\leftarrow k+1$. If $k+1=n$, stop: $(w_1, \cdots, w_n)$ is a perfect ordering. Otherwise repeat.

The complexity of this procedure is $O(|V|^3)$; but one can do better, as shown by Rose, Tarjan and Lueker [1976], whose procedure has a complexity of $O(|V|+|E|)$. Their approach essentially uses the following property.

Given an arbitrary ordering $\sigma=(v_1, \cdots, v_n)$ of V, call the elements of the set

$$N^+(v_i) := \{v_j\in V\setminus\{v_i\} \mid (v_i, v_j)\in E \text{ and } j>i\}$$

the *successors* of v_i, and call the successor of v_i with smallest index, the *first successor* of v_i. We say that v_i is *quasi-simplicial* in $\sigma=(v_1, \cdots, v_n)$ if the first successor of v_i is adjacent to every other successor of v_i.

THEOREM 1. *Given a graph $G=(V, E)$ and an ordering $\sigma=(v_1, \cdots, v_n)$, the following two statements are equivalent*;

(i) *for $i=1, \cdots, n$, v_i is simplicial in $G(\{v_i, \cdots, v_n\})$*;

(ii) *for $i=1, \cdots, n$, v_i is quasi-simplicial in σ.*

Proof. (i) obviously implies (ii). Now suppose (ii) holds; then v_n is simplicial in $G(\{v_n\})$. Assume v_i is simplicial in $G(\{v_i, \cdots, v_n\})$ for $i=n, n-1, \cdots, k+1$, and let $i=k$. If v_k has less than two successors, we are done; otherwise let $w_1, \cdots, w_p$ be the successors of v_k, with $w_1=v_l$ the first successor. Since v_k is quasi-simplicial, $w_2, \cdots, w_p$ are adjacent to $w_1=v_l$. Since v_l by assumption is simplicial, $w_2, \cdots, w_p$ are pairwise adjacent. Hence v_k is simplicial, and the induction is complete. □

COROLLARY 1.1. *An ordering $\sigma=(v_1, \cdots, v_n)$ is perfect if and only if for $i=1, \cdots, n$, v_i is quasi-simplicial in σ.*

While the complexity of checking whether a vertex is simplicial is $O(|V|^2)$, checking for quasi-simpliciality requires only $O(|V|)$ adjacency tests.

Rose, Tarjan and Lueker [1976] use a lexicographic labeling algorithm (LEX P) for generating an ordering of V that is perfect if and only if G is triangulated; they then use an adjacency testing algorithm (FILL) to check whether the ordering produced by LEX P is perfect. The adjacency testing algorithm is a straightforward application of Corollary 1.1.

As to LEX P, it starts by assigning the label $\varnothing$ to every vertex. It proceeds by numbering the vertices $n, n-1, \cdots, 1$ in an order based on the lexicographic labels which are updated at every iteration. For $i=n, n-1, \cdots, 1$, it thus performs the following steps:

1. Choose a vertex v with lexicographically largest label, and assign the number i to v.

2. Append i to the label of every unnumbered vertex adjacent to v, and set $i\leftarrow i-1$.

THEOREM 2 (Rose, Tarjan and Lueker [1976]). *G is triangulated if and only if the ordering generated by algorithm* LEX P *is perfect.*

Our procedure combines this lexicographic labeling algorithm with a modified version of the adjacency testing algorithm, which screens the vertices to be included into the MTIS, rejecting all those whose addition would make the ordering imperfect. Thus our algorithm orders V into a sequence σ, but also generates a subsequence $\sigma(T)$ whose elements T induce a maximal triangulated subgraph in G.

ALGORITHM TRIANG. Let i denote the current number (rank in σ) to be assigned, T the set of vertices currently included into the triangulated subgraph, $\sigma(T)$ the ordering defined on T.

0. *Initialization.* Assign the label $\varnothing$ to every vertex. Set $i \leftarrow n$, $T \leftarrow \varnothing$, $\sigma(T) \leftarrow \varnothing$.

1. *Choosing v.* If $i = 0$, stop: $G(T)$ is a maximal triangulated subgraph. Otherwise, choose an unnumbered vertex v with lexicographically largest label, assign number i to v, i.e., set $v_i \leftarrow v$, and go to 2.

2. *Checking whether v can be added to T.* Let σ^0 be the sequence obtained from $\sigma(T)$ by adding v as the first element. If v is quasi-simplicial in σ^0, set $T \leftarrow T \cup \{v\}$, $\sigma(T) \leftarrow \sigma^0$, and go to 3. Otherwise set $i \leftarrow i-1$ and go to 1.

3. *Labeling.* Append i to the label of each unnumbered vertex w adjacent to v, set $i \leftarrow i-1$, and go to 1.

THEOREM 3. *Algorithm* TRIANG *finds a* MTIS *of G in $O(|V|+|E|)$ time.*

Proof. (i) Let $\sigma(T) = (w_1, \cdots, w_{|T|})$ be the ordering generated by TRIANG. Since step 2 guarantees that for $i = 1, \cdots, |T|$, w_i is quasi-simplicial in $G(\{w_i, \cdots, w_{|T|}\})$, from Theorem 1 the ordering $\sigma(T)$ is perfect. Hence the subgraph $G(T)$ is triangulated.

(ii) We claim that $G(T)$ is maximal. For if not, then there exists a vertex $v \in V \setminus T$ such that $G(T \cup \{v\})$ is triangulated. Since $v \notin T$, v was rejected at some iteration as not being quasi-simplicial in the sequence σ^0 formed in step 2. Let this particular sequence be σ^{00} and let T^0 be the vertex set ordered by σ^{00} (containing v as its first vertex). Then $G(T^0)$ is not triangulated. But $T^0 \subset T \cup \{v\}$, and $G(T \cup \{v\})$ is triangulated, a contradiction. Thus $G(T)$ is a MTIS of G.

(iii) Next we establish the complexity of TRIANG. One way of carrying out the steps of TRIANG efficiently is to keep a list of subsets of vertices with the same label, where the subsets are ordered according to lexicographically decreasing labels. Whenever we append an entry to the label of a vertex v in step 3, we take v out of its subset S. It is easy to see that the new label of v cannot be greater than that of the vertices in the subset S' preceding S. If it is the same as the latter, we put v into S'; otherwise we create a new subset $S'' = \{v\}$, placed between S' and S. This operation requires constant time for each vertex whose label is updated, and it keeps the vertex with lexicographically largest label at the head of the list. Thus each application of step 3 requires $O(\deg(v))$ time (where v is the vertex whose neighbors are having their labels updated and $\deg(v)$ is the degree of v), and each application of step 1 requires a constant amount of time (choose the vertex at the head of the list). Finally, checking in step 2 whether v is quasi-simplicial in σ^0 involves checking for each successor of v in σ^0, other than the first one, whether it is adjacent to the latter; and this again requires $O(\deg(v))$ time. To obtain the complexity of the entire sequence of steps we have to sum over all $v \in V$, which yields $O(|V|+|E|)$.

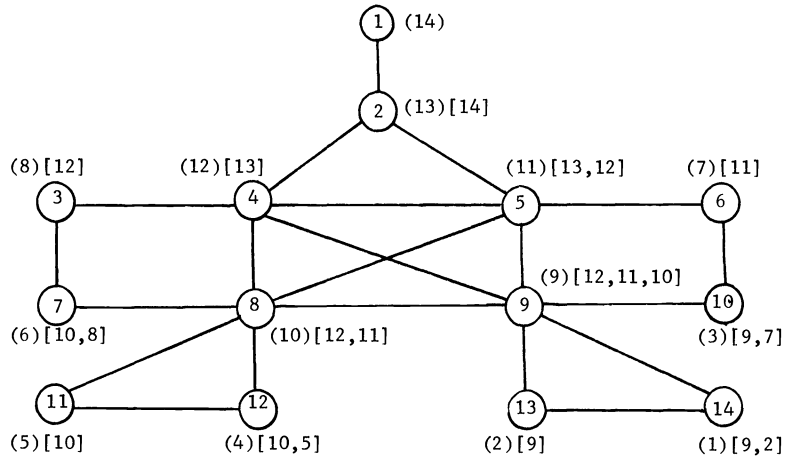
Notice that, as a byproduct of applying TRIANG, we also identify the (or a) largest clique of $G(T)$: it consists of the vertex v with the largest number of successors in $\sigma(T)$ (the perfect ordering produced by TRIANG), together with the successors of v. A minor modification of step 2 provides for counting the successors of v and storing the new largest clique whenever the earlier largest clique size is exceeded.

Next we illustrate TRIANG on an example.

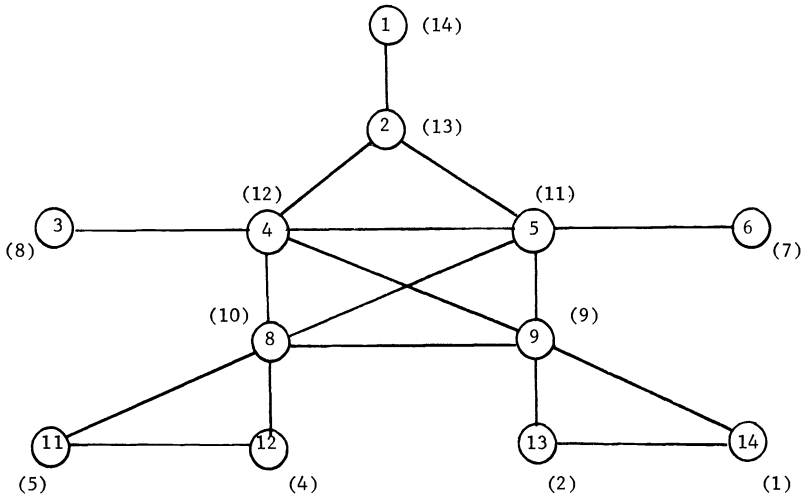
Example 1. Let G be the graph of Fig. 1(a), where the number (rank in σ) of each vertex is shown in parentheses, while its label is shown in square brackets.

After initializing the label of each vertex with $\varnothing$ (not shown in the figure), we choose vertex 1 (breaking ties arbitrarily) and assign to it number 14. Clearly, vertex 1 is quasi-simplicial in $G(\{1\})$, and so we set $T=\{1\}$, $\sigma(T)=(1)$. We then append 14 to the label of the unique vertex adjacent to 1, namely 2. At the next iteration we choose vertex 2, assign to it number 13, set $T=\{1,2\}$, $\sigma(T)=(2,1)$, and append 13 to the labels of 4 and 5. Next we choose vertex 4 (tied with 5 for the lexicographically largest label), number it 12, set $T=\{1,2,4\}$, $\sigma(T)=(4,2,1)$, and add 12 to the labels of 3, 5, 8, 9.

At the next iteration, vertex 5 becomes number 11, and 11 is added to the labels of 6, 8, 9. Then vertex 8 becomes number 10, and 10 is appended to the labels of 7,



(a)



(b)

FIG. 1

9, 11, 12. Vertex 9 becomes number 9, and 9 is appended to the labels of 10, 13 and 14. Vertex 3 becomes number 8, and 8 is appended to the label of 7. Vertex 6 becomes 7, with 7 added to the label of 10.

At this point, the next vertex with lexicographically largest label is 7; it gets the number 6, but setting $\sigma^0 = (7, 6, 3, 9, 8, 5, 4, 2, 1)$ in step 2 shows that 7 is not quasi-simplicial in σ^0: the first successor of 7, which is 3, is not adjacent to 8, also a successor of 7. Thus 7 is rejected (not included in T).

As the process continues, vertex 10 is also rejected (upon becoming number 3), and all the remaining vertices are added to T. The final sequence $\sigma(T)$ is (14, 13, 12, 11, 6, 3, 9, 8, 5, 4, 2, 1), and the MTIS $G(T)$ is shown in Fig. 1(b). The vertex with the largest number of successors in $\sigma(T)$ is 9, and the corresponding maximum clique has vertex set {9, 8, 5, 4}.

The MTIS found by TRIANG is maximal with respect to set inclusion, but not necessarily of maximum cardinality. The problem of finding a maximum cardinality triangulated induced subgraph of an arbitrary graph is NP-complete (Yannakakis [1978]). A graph may have many MTIS's, and they may be of different sizes. More importantly, they may have maximum cliques of different sizes.

Note that in Step 1 of TRIANG one is free to choose any unnumbered vertex with a lexicographically largest label, and different rules for breaking ties produce different MTIS's. We implemented and tested four tie breaking rules, based on the following criteria: (1) random choice; (2) largest number of adjacent vertices in the current set T; (3) largest number of adjacent vertices in $T \cup C$, where C is the set of candidates; (4) largest degree in G. The comparative results are discussed in § 6.

4. Finding a maximal k-chromatic induced subgraph. Having generated a MTIS $G(T)$, next we extend $G(T)$ to a maximal induced subgraph $G(W)$ of G with the same maximum clique size as $G(T)$. This is done as follows. Since $G(T)$ is triangulated, its maximum clique size is the same as its chromatic number, i.e., the cardinality of a minimum coloring of (the vertices of) $G(T)$. Furthermore, given a perfect ordering $\sigma = (v_1, \cdots, v_{|T|})$ of the vertices of $G(T)$, finding a minimum coloring in $G(T)$ is quite straightforward: initialize k color classes (independent sets), where k is the size of a maximum clique in $G(T)$, and examine the vertices of $G(T)$ in the reverse order of σ (i.e., starting with $v_{|T|}$), putting each vertex in the first color class where it fits (i.e., where it has no neighbors). Since each v_i has at most $k-1$ successors in $G(v_i, \cdots, v_{|T|})$, each vertex fits into some color class. Expanding $G(T)$ while preserving k color classes can then be done simply by examining each vertex $v \in V \setminus T$ in turn, and putting it into the first color class where it fits, or leaving it out if it does not fit into any class. Here is a formal statement of the procedure.

ALGORITHM COLOR.

0. Initialize k color classes $C_1, \cdots, C_k$, $C_j = \emptyset$, $j = 1, \cdots, k$, set $V \leftarrow V \setminus T$, and go to 1.

1. If $T = \emptyset$, go to 2. Otherwise, choose $v \in T$ with the highest rank in σ, and a color class C_j that contains no vertex adjacent to v, set $C_j \leftarrow C_j \cup \{v\}$, $T \leftarrow T \setminus \{v\}$, and go to 1.

2. Choose any $v \in V$. If there exists a color class that contains no vertex adjacent to v, choose such a color class C_j and set $C_j \leftarrow C_j \cup \{v\}$. In any case, set $V \leftarrow V \setminus \{v\}$. If $V \neq \emptyset$, go to 2. If $V = \emptyset$, stop: $G(W)$, where $W = \bigcup_{i=1}^{k} C_i$, is a maximal k-chromatic induced subgraph of G.

The above algorithm extends a MTIS of G with maximum clique size k to a maximal k-chromatic induced subgraph of G. In the branch and bound procedure to

be described below, we sometimes need to find a maximal k-chromatic induced subgraph of some graph for which we do not have a MTIS. In these cases we use a modified version of the above algorithm, called COLOR 2, in which the instruction "set $V \leftarrow V \setminus T$ and go to 1" of the Initialization Step is replaced by "go to 2," and Step 1 is skipped.

Since checking whether a vertex v fits into any of the k color classes requires $\deg(v)$ adjacency tests, the complexity of COLOR is again $O(|V|+|E|)$.

A more sophisticated version of the coloring procedure, called COLINT, amends the above algorithm with an interchange heuristic; i.e., having found a maximal k-chromatic induced subgraph $G(W)$ by steps 0-1-2 (or 0-2) above, it then tries to increase the cardinality of W by performing the following step for each $v \in V \setminus W$ in turn (here V is again the vertex set of G):

3. Put v into the color class C_j that contains the smallest number of vertices adjacent to v, and try to transfer each $w \in C_j$ adjacent to v into some other color class that contains no vertex adjacent to w.

If Step 3 is successful for α vertices $v \in V \setminus W$, the cardinality of W increases by α.

Step 3 is computationally more expensive than the procedure 0-1-2. Its calculations can at best be organized so as to keep its complexity within $O(\deg(v))$ for each vertex v whose coloring is attempted without success, and $O(|E|)$ for each vertex colored in Step 3. To attain this performance, it is recommendable to amend the adjacency list of each $v \in V$, say $L_1(v)$, with a second list of the form

$$L_2(v) = (j_1, \cdots, j_{|L_1|}; j_*; n(j_*)),$$

where for $i = 1, \cdots, |L_1|$, j_i is the index of the color class containing the ith element of $L_1(v)$; j_* is the index of the color class containing the fewest vertices adjacent to v; and $n(j_*)$ is the number of vertices adjacent to v in color class C_{j*}. Then identifying the color class C_{j*} containing the smallest number of vertices adjacent to v requires looking up the next to last entry of $L_2(v)$; and checking for every $w \in C_{j*}$ adjacent to v whether it can be transferred into some other color class where it has no neighbors amounts to checking whether the last entry of $L_2(w)$ is 0. However, upon the successful transfer of some $w \in C_{j*}$ to a new color class, the lists $L_2(u)$ have to be updated for every u adjacent to w, a task which for all $w \in C_{j*}$ adjacent to v requires a total effort of $O(|E|)$. Hence the total effort involved in the interchange heuristic is $O(|V| \cdot |E|)$.

Thus the worst case bound on the computational effort required by COLINT is higher than for COLOR, and it is not clear a priori whether the increase in the size of W justifies the increased effort. Therefore, pending some computational testing that we intend to carry out in the near future, we have not incorporated COLINT into the current version of our algorithm.

5. The algorithm as a whole. Our algorithm starts by finding a MTIS $G(T)$ of G, and then extends $G(T)$ to a maximal k-chromatic induced subgraph $G(W)$ of G, where k is the size of a maximum clique in $G(T)$ (hence in $G(W)$). If $W = V$, we are done; otherwise we branch, based on the following considerations. For $v \in V$, we denote $N(v) = \{w \in V \setminus \{v\} \mid (v, w) \in E\}$.

THEOREM 4. *Let k be the cardinality of a maximum clique of $G(W)$, and let $(v_1, \cdots, v_m)$ be an arbitrary ordering of the set $V \setminus W$. If G has a clique K^* such that $|K^*| > k$, then K^* is contained in one of the m sets*

$$V_i := \{v_i\} \cup N(v_i) \setminus \{v_1, \cdots, v_{i-1}\}, \qquad i = 1, \cdots, m,$$

where for $i = 1$ we define $\{v_1, \cdots, v_{i-1}\} = \varnothing$.

Proof. Since $|K^*| > k$, $K^* \not\subseteq W$ and thus $v_i \in K^*$ for some $i \in \{1, \cdots, m\}$. Since K^* is a clique, this implies $K^* \subseteq \{v_i\} \cup N(v_i)$ for some $i \in \{1, \cdots, m\}$. Thus either $K^* \subseteq \{v_1\} \cup N(v_1)$, or else K^* is contained in one of the sets $\{v_i\} \cup N(v_i) \backslash \{v_1\}$, $i = 2, \cdots, m$. In the latter case, either $K^* \subseteq \{v_2\} \cup N(v_2) \backslash \{v_1\}$, or else K^* is contained in one of the sets $\{v_i\} \cup N(v_i) \backslash \{v_1, v_2\}$. The result follows by induction. □

Theorem 4 leads to the following *branching rule.* Node t of the search tree (where t is a string) is characterized by the pair $[I_t, E_t]$, where I_t and E_t are the sets of vertices forcibly included into, and excluded from, the graph on which the current subproblem is defined. In the language of implicit enumeration, the variables corresponding to I_t and E_t are fixed at 1 and 0, respectively, while the variables corresponding to $S_t = V \backslash (I_t \cup E_t)$ are free. Let $G(W_t)$ be a maximal k-chromatic induced subgraph of $G(S_t)$, and let $S_t \backslash W_t = \{v_1, \cdots, v_m\}$. We then generate m new nodes (subproblems) $t1, \cdots, tm$, by setting

$$
\begin{aligned}
&I_{ti} = I_t \cup \{v_i\}, \\
&E_{ti} = E_t \cup (S_t \backslash (\{v_i\} \cup N(v_i))) \cup \{v_1, \cdots, v_{i-1}\}, \qquad i = 1, \cdots, m.
\end{aligned} \tag{5.1}
$$

By construction, for any t the set $S_t = V \backslash (I_t \cup E_t)$ is contained in $\cap_{v \in I_t} N(v)$. Therefore I_t consists of pairwise adjacent vertices, and the cardinality of I_t (which is the same as the length of the string t) is equal to the level of node t in the search tree (where the level of the root of the tree is defined to be 0). The subproblem associated with node t of the search tree consists of finding a maximum clique in $G(S_t)$: if K_t is a clique in $G(S_t)$, then $K_t^* = K_t \cup I_t$ is a clique in G.

The problems $G(S_t)$ could be solved by applying to them recursively the two subroutines TRIANG and COLOR. However, computational testing has convinced us that this is not the best way to proceed: finding a MTIS in each subgraph $G(S_t)$ is relatively expensive, and the occasions when $G(S_t)$ contains a vertex set K_t such that $K_t \cup I_t$ is a clique larger than the current largest one, are not too frequent. It is considerably cheaper to simply use the algorithm COLOR to find a maximal $(k - |I_t|)$-chromatic induced subgraph of $G(S_t)$ (where k is the size of the largest clique in G found so far), except in cases where there is serious indication that $G(S_t)$ may contain a clique larger than $k - |I_t|$. Such is the case, for instance, whenever $|I_t| = k$ and $S_t \neq \emptyset$. The motivation for choosing $k - |I_t|$ as the chromatic member of the maximal subgraph we construct, is that if $G(S_t)$ itself turns out to be such a subgraph, then P_t can be eliminated.

We are now in a position to state our algorithm formally. We denote by L the list of live (active) nodes of the search tree, and by P_t the subproblem at node t.

Step 0 (*Initialization*). Put into L the problem P_0 of finding a maximum clique in $G = (V, E)$. Set $t = 0$, $I_t = E_t = \emptyset$, $S_t = V$, $k = 0$, and go to 1.

Step 1 (*Subproblem selection*). If $L = \emptyset$, stop: the current clique is maximum in G. Otherwise choose an element P_t of L and remove it from L.

If $|V \backslash E_t| \leqq k$, eliminate P_t and go to 1.

If $|V \backslash E_t| > k$ but $|I_t| = k$, go to 2.

Otherwise go to 4.

Step 2 (*TRIANG*). Find a MTIS $G(T_t)$ of $G(S_t)$, and a maximum clique K_t in $G(T_t)$. Store $I_t \cup K_t$ as the current largest clique, and set $k \leftarrow |I_t \cup K_t|$. If $T_t = S_t$, eliminate P_t and go to 1. Otherwise go to 3.

Step 3 (*COLOR*). Find a minimum coloring of $G(T_t)$, and expand its color classes to find a maximal $|K_t|$-chromatic induced subgraph $G(W_t)$ of $G(S_t)$. If $W_t = S_t$, eliminate P_t and go to 1. Otherwise go to 5.

Step 4 (*COLOR* 2). Find a maximal $(k-|I_t|)$-chromatic subgraph $G(W_t)$ of $G(S_t)$. If $W_t = S_t$, eliminate P_t and go to 1. Otherwise go to 5.

Step 5 (*Branching*). Order the set $S_t \setminus W_t$ into a sequence $(v_1, \cdots, v_m)$. Generate m new subproblems defined by (5.1), place them into L, and go to 1.

The strategy of this algorithm differs from that of most branch and bound procedures in that no attempt is made to derive an upper bound on the objective function value of the subproblems P_t generated during the procedure. Instead, the focus is on finding maximal subgraphs (i.e., subproblems) for which the current lower bound is also an upper bound, and which can thus be eliminated. Since we have no known upper bound on the value of each P_t, the search strategy used is depth first.

Our approach is based on the idea of finding a maximal induced subgraph G' of G, of a type for which it is easy to find a maximum clique in G'; and we chose G' to be triangulated. But triangulated subgraphs are not the only ones in which it is easy to locate a maximum clique. Another such class is that of *cotriangulated* graphs, i.e., complements of triangulated graphs. Finding a maximum clique in a cotriangulated graph G'' amounts to finding a maximum vertex packing in the corresponding triangulated graph $\bar{G}''$ (the complement of G''), and this can also be done in $O(|V|+|E|)$ time (see, for instance, Golumbic [1980]). In particular, when the graph G is dense, there are good chances that a maximal cotriangulated induced subgraph (MCIS) is larger, and contains a larger clique, than a MTIS of the same graph. Thus Step 2 of our algorithm can be replaced by the following more expensive, but also more efficient version.

Step 2′ (*TRIANG*). Find a MTIS $G(T_t')$ of $G(S_t)$ and a maximum clique K_t' in $G(T_t')$. Find a MCIS $G(T_t'')$ of $G(S_t)$ and a maximum clique with vertex set K_t'' in $G(T_t'')$. Choose $|K_t| = \max\{|K_t'|, |K_t''|\}$, and set $T = T'$ if the maximum is attained for $|K_t'|$, $T = T''$ otherwise. Break ties by choosing $|T| = \max\{|T'|, |T''|\}$. Store $I_t \cup K_t$ as the current clique and set $k \leftarrow |I_t \cup K_t|$. If $T_t = S_t$, eliminate P_t and go to 1. Otherwise go to 3.

Another version, which avoids the effort involved in finding both a MTIS and a MCIS of G, makes a decision as to which one to use, based on the density of the given subgraph: if the number of edges of $G(S_t)$ is $\leqq 1/4|S_t|(|S_t|-1)$, it finds a MTIS; otherwise, it finds a MCIS in $G(S_t)$. Then it finds a maximum clique K_t in the resulting graph $G(T_t)$.

These amended versions of the algorithm have not yet been implemented: our computational experience is limited to the basic version stated above.

Next we illustrate this basic version of the algorithm on a numerical example.

Example 2. Consider the graph G shown in Fig. 2(a). We initialize and go to Step 1. We choose and remove from L the problem P_0 defined on G. Since $|V \setminus E_t| = |V| > k$ and $|I_t| = k = 0$, we go to Step 2, i.e., apply TRIANG. Suppose we generate the ordering $\sigma = (2, 12, 8, 5, 9, 1, 4, 11, 10, 7, 3, 6)$ and the maximal triangulated subgraph $G(T_0)$, with $T_0 = \{1, 3, 5, 6, 7, 8, 10, 11\}$ shown in Fig. 2(b). A maximum clique in $G(T_0)$ has vertex set $K_0 = \{5, 7, 8\}$. We store K_0, set $k = 3$ and go to Step 3.

COLOR finds the minimum coloring $(\{6, 1, 5\}, \{3, 7, 10, 11\}, \{8\})$ in $G(T_0)$, and extends it by including vertex 12 into the first, and vertex 2 into the second color class. Thus our maximal 3-chromatic induced subgraph is $G(W_0)$, with all but 2 vertices (4 and 9) contained in W_0.

The Branching Step leads to the creation of two subproblems, P_1 and P_2, defined by

$$I_1 = \{4\}, \qquad E_1 = \{2, 6, 10, 11, 12\}$$

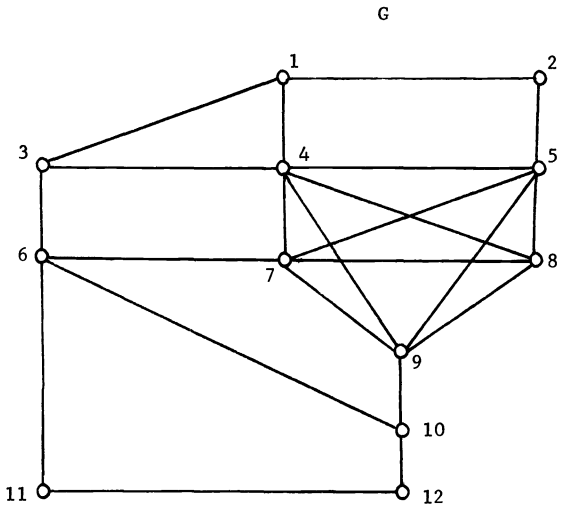


FIG. 2(a)

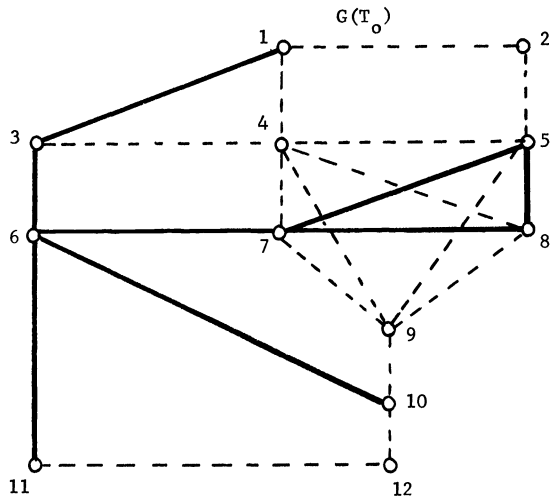


FIG. 2(b)

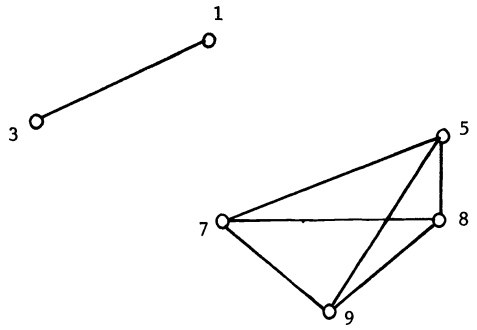


FIG. 2(c)

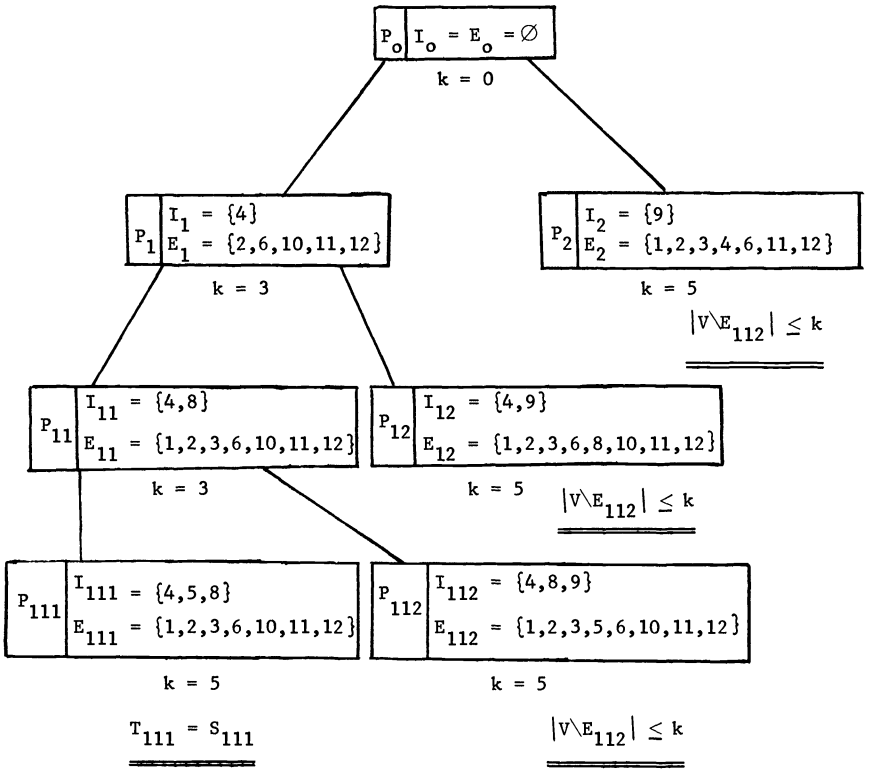


FIG. 2(d)

and

$$I_2 = \{9\}, \qquad E_2 = \{1, 2, 3, 4, 6, 11, 12\},$$

respectively.

We choose P_1 first. Since $|V \setminus E_1| = 7 > k = 3$, and $|I_1| = 1 < k$, we go to Step 4. Since $k - |I_1| = 2$, we use COLOR 2 to find a maximal 2-chromatic induced subgraph of $G(S_1) = G(\{1, 3, 5, 7, 8, 9\})$. This is $G(W_1)$, shown in Fig. 2(c), with the coloring $(\{1, 7\}, \{3, 5\})$ and $W_1 = \{1, 3, 5, 7\}$. Since $S_1 \setminus W_1 = \{8, 9\} \neq \emptyset$, we go to Step 5.

Branching creates two successors of P_1, the subproblems P_{11} and P_{12}, defined by

$$I_{11} = I_1 \cup \{8\} = \{4, 8\}, \qquad E_{11} = E_1 \cup \{1, 3\} = \{1, 2, 3, 6, 10, 11, 12\}$$

and

$$I_{12} = I_1 \cup \{9\} = \{4, 9\}, \qquad E_{12} = E_1 \cup \{1, 3, 8\} = \{1, 2, 3, 6, 8, 10, 11, 12\}$$

respectively.

We choose problem P_{11} and since $|V \setminus E_{11}| = 5 > k = 3$ and $|I_{11}| = 2 < k$, we go to Step 4. Since $k - |I_{11}| = 1$, we construct a maximal 1-chromatic induced subgraph $G(W_{11})$ of $G(S_{11}) = G(\{5, 7, 9\})$. We have $W_{11} = \{7\}$ and since $S_{11} \setminus W_{11} = \{5, 9\} \neq \emptyset$, we go to the Branching Step. P_{11} has two successors, P_{111}, and P_{112}, defined by

$$I_{111} = I_{11} \cup \{5\} = \{4, 5, 8\}, \qquad E_{111} = E_{11},$$

and

$$I_{112} = I_{11} \cup \{9\} = \{4, 8, 9\}, \qquad E_{112} = E_{11} \cup \{5\} = \{1, 2, 3, 5, 6, 10, 11, 12\},$$

respectively.

We choose P_{111} and since $|I_{111}| = k = 3$, we go to Step 2. Applying TRIANG to $G(S_{111}) = G(\{7, 9\})$ yields the clique $\{7, 9\}$, which together with I_{111} forms the 5-clique with node set $\{4, 5, 7, 8, 9\}$ in G. Thus we store this as the current clique (in place of $\{5, 7, 8\}$), set $k = 5$ and since $T_{111} = S_{111}$, eliminate P_{111}.

Next we choose P_{112}. Since $|V \setminus E_{112}| = 4 \leqq k = 5$, we eliminate P_{112}. Then choosing P_{12}, we find that $|V \setminus E_{12}| = 4 \leqq k$ and eliminate P_{12}. The only subproblem remaining in L is now P_2. Since $|V \setminus E_2| = 5 = k$, we eliminate P_2 and terminate with the maximum clique induced by $\{4, 5, 7, 8, 9\}$ found at P_{111}.

The search tree is shown in Fig. 2(d).

6. Computational experience. Several variants of the basic version of our algorithm were implemented and tested. Four of the variants (TC1–TC4) correspond to the tie breaking rules used in TRIANG, as defined at the end of § 3. The variant TC* does not have Step 4; instead, Step 1 is always followed by 2 and 3. In other words, this variant finds a MTIS in every subgraph generated by the procedure. Finally, the variant C does not use TRIANG at all, and from Step 1 either goes to Step 4 or again to Step 1 (thus both Steps 2 and 3 are eliminated).

For purposes of comparison, we have also implemented Algorithm CACM457 of Bron and Kerbosch [1973], described by Gerhards and Lindenberg [1979] as the currently known most efficient clique finding procedure. Since CACM457 was originally designed to list all the cliques of a graph, we modified it to just find a largest clique,

TABLE 5.1
Computational results with PASCAL *codes on a* DEC 20-60, *on random graphs with* 50 *vertices.*

		CPU time (ms)						
D	Q	CACM457*	TC1	TC2	TC3	TC4	TC*	C
5	3	100	49	45	92	53	91	32
10	4	118	54	51	83	52	130	43
15	4	147	67	64	113	60	184	51
20	4	200	85	69	129	76	282	66
25	6	218	89	87	108	87	365	75
30	6	263	84	112	121	78	354	96
35	6	372	69	116	117	128	761	100
40	6	615	130	168	202	149	2731	119
45	7	630	146	147	173	162	+	153
50	7	969	249	262	234	210	+	207
55	8	1346	291	223	274	269	+	287
60	9	1838	293	406	346	309	+	336
65	9	3333	675	514	479	510	+	478
70	11	4809	543	634	750	463	+	568
80	13	19548	3212	2170	1732	1476	+	2023
85	18	18458	638	1350	753	1030	+	1430
90	23	25763	832	966	462	571	+	1674
95	27	45542	712	358	490	277	+	4086

D = density of the graph (%). Q = size of maximum clique. + runs not attempted in view of the obvious inferiority of this version.

by adding tests to eliminate those nodes of the search tree that cannot contain a clique larger than the current largest one. We call this modified algorithm CACM457*. We also implemented a more recent algorithm, due to Loukakis and Tsouros [1982], originally designed for finding a maximum vertex packing. Our implementation, which we call LT, finds a maximum clique in G by essentially applying the Loukakis-Tsouros algorithm to $\bar{G}$. Both CACM457* and LT are rather straightforward implicit enumeration procedures, very similar to each other. CACM457* has one more logical test (exclusion rule), which makes it on the average more efficient than the LT algorithm. (Although the Loukakis-Tsouros paper is more recent, its authors do not seem to have been aware of the existence of the Bron-Kerbosch algorithm of 1973.)

TABLE 5.2
Computational results with C codes on a VAX 11-780, *on random graphs with* 50-400 *vertices.*

Graph characteristics								
			No. of search tree nodes			CPU time (ms)		
Vertices	Density (%)	Size of maximum clique	CACM	L-T	B-Y	CACM	L-T	B-Y
50	10	3	28	21	7	116	133	50
50	20	4	62	59	25	183	200	67
50	30	5	87	84	40	233	283	67
50	40	6	175	208	82	467	617	116
50	50	8	244	265	139	700	983	200
50	60	9	547	968	284	1,283	3,450	383
50	70	12	1,310	4,076	551	3,100	20,199	883
50	80	15	4,334	16,143	825	10,600	104,096	1,917
50	90	22	9,907	85,869	421	27,499	778,335	1,983
100	10	4	73	62	27	500	567	200
100	20	5	225	234	92	999	1,067	267
100	30	6	623	670	327	2,233	2,499	599
100	40	7	1,459	1,600	643	4,966	5,816	1,083
100	50	9	4,186	5,290	1,938	13,416	22,716	3,217
100	60	12	14,304	20,600	7,798	50,648	126,645	15,883
100	70	15	93,392	335,011	53,074	308,254	1,073,020	112,695
200	10	4	291	277	160	2,799	2,750	983
200	20	5	1,368	1,341	700	7,233	6,283	1,899
200	30	7	5,247	5,400	2,464	26,732	26,832	5,599
200	40	9	19,118	22,445	9,490	94,730	128,045	19,782
200	50	11	—	—	61,374	—	—	123,628
200	60	14	—	—	526,852	—	—	1,164,170
300	10	5	538	—	354	8,482	—	2,416
300	20	6	4,420	—	1,775	30,382	—	5,783
300	30	8	17,409	—	11,587	127,611	—	25,965
300	40	10	114,416	—	55,417	654,790	—	118,495
300	50	13	—	—	526,078	—	—	1,156,020
400	10	5	1,266	—	619	17,056	—	4,066
400	20	7	9,391	—	3,575	78,463	—	13,383
400	30	8	63,753	—	32,092	435,916	—	80,997
400	40	10	—	—	238,790	—	—	562,627

CACM = CACM457*; L-T = Loukakis-Tsouros; B-Y = Balas-Yu.

The computational results listed below are of two kinds. The six variants of our algorithm, as well as the algorithm CACM457*, were first coded in PASCAL and run on a DEC 20-60 computer at CMU, on 18 problems defined on randomly generated graphs with 50 vertices, of density (i.e., probability of presence of a given edge) ranging from 5% to 95%. The results are shown in Table 5.1.

A comparison of the 6 variants of our algorithm shows TC*, i.e., the variant that finds a MTIS at every iteration, clearly inferior to the other five. Of the remaining five variants, TC4, i.e., the one that uses the degree in G as a tie breaking rule for choosing the next vertex to be numbered in TRIANG, seems slightly superior to the others on the harder problems, i.e., on graphs with largest clique size of at least 7, whereas on the easier problems TC1, the variant that breaks ties by random choice, performs best. Surprisingly, the variant C that uses only the coloring routine without looking for a MTIS, does equally well or even better than the others on problems with largest clique size $\leqq 7$, although its relative performance rapidly deteriorates for largest clique sizes greater than 11.

In the above discussion we used the largest clique size rather than the density of the graph as a criterion for distinguishing between easy and hard problems. Both criteria are of course relevant, but the size of the largest clique seems to be a better measure of problem difficulty.

Variant TC4 of our algorithm, along with CACM457* and the Loukakis-Tsouros algorithm, were subsequently coded in C and run on a VAX 11-780 at Bell Labs, on a set of larger random graphs, with up to 400 vertices and 30,000 edges. The results are shown in Table 5.2.

REFERENCES

[1] E. BALAS AND H. SAMUELSSON, *A node covering algorithm*, Naval Res. Log. Quart., 24(2) (1977), pp. 213-233.

[2] C. BERGE, *Farbung von Graphen, deren sämtliche bzw. deren ungerade Kreise Starr sind*, Wiss. Z. Martin-Luther-Universität, Halle-Wittenberg, Math-Natur. Reihe (1961), pp. 114-115.

[3] C. BRON AND J. KERBOSCH, *Finding all cliques of an undirected graph*, Comm. ACM, 16(9) (1973), pp. 575-577.

[4] V. CHVÁTAL, *On certain polytopes associated with graphs*, J. Combin. Theory, B, 18 (1975), pp. 138-154.

[5] G. A. DIRAC, *On rigid circuit graphs*, Abh. Math. Sem. Univ., Hamburg, 25 (1961), pp. 71-76.

[6] D. R. FULKERSON AND O. A. GROSS, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835-855.

[7] F. GAVRIL, *Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of chordal graph*, this Journal, 1 (1972), pp. 180-187.

[8] L. GERHARDS AND W. LINDENBERG, *Clique detection for nondirected graphs: Two new algorithms*, Computing, 21 (1979), pp. 295-322.

[9] M. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[10] M. GRÖTSCHEL, L. LOVÁSZ AND A. SCHRIJVER, *Polynomial Algorithms for Perfect Graphs*, C. Berge and V. Chvátal, eds., Topics on Perfect Graphs, North-Holland, Amsterdam, 1984, pp. 325-356.

[11] E. LOUKAKIS AND C. TSOUROS, *Determining the number of internal stability of a graph*, Intern. J. Computer Math., 11 (1982), pp. 207-220.

[12] L. LOVÁSZ, *Normal hypergraphs and the perfect graph conjecture*, Discrete Math., 2 (1972), pp. 253-267.

[13] G. L. NEMHAUSER AND L. E. TROTTER, JR., *Vertex packings: structural properties and algorithms*, Math. Programming, 8 (1975), pp. 232-248.

[14] D. J. ROSE, R. E. TARJAN AND G. S. LUEKER, *Algorithmic aspects of vertex elimination on graphs*, this Journal, 5 (1976), pp. 266-283.

[15] R. E. TARJAN AND A. E. TROJANOWSKI, *Finding a maximum independent set*, this Journal, 6 (1977), pp. 537-546.

[16] M. YANNAKAKIS, *Node and edge deletion of NP-complete problems*, Proc. 10th Annual ACM Symposium on Theory of Computing, ACM, New York, 1978, pp. 253-264.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
014

AVERAGE CASE ANALYSIS OF MARKING ALGORITHMS*

D. S. HIRSCHBERG† AND L. L. LARMORE‡

Abstract. The Lindstrom marking algorithm uses bounded workspace. Its time complexity is $O(n^2)$ in all cases, but it has been assumed that the average case time complexity is $O(n \log n)$. It is proven that the average case time complexity is $\Theta(n^2)$ for a wide variety of probability distributions. Similarly, the average size of the Wegbreit bit stack is shown to be $\Theta(n)$.

Key words. Lindstrom, garbage collection, marking algorithm

AMS(MOS) subject classification. 68Q25

1. Introduction. Consider a data store organized into nodes, each of which has two link fields, L and R. Each link field contains either the address of a node in the store, or an uninterpreted atomic value. One specific node will be designated as the root.

In garbage collection, we may wish to mark all nodes accessible from the root; assume then that each node has a mark bit. Several algorithms to do this are known. Baer and Fries [1] analyze some of these, including the Schorr-Waite tag bit algorithm [4], the Wegbreit bit-stack algorithm [5] and the Lindstrom algorithm [3].

Both the Schorr-Waite and Wegbreit algorithms have time complexity $O(n)$ where n is the number of nodes accessible from the root, and both use $O(n)$ additional workspace. The Schorr-Waite algorithm requires a tag bit in each node, whereas the Wegbreit algorithm uses a bit stack, whose size is $O(n)$ in the worst case, for the same purpose. At first glance, it may appear that the maximum size of the Wegbreit bit stack is $O(\log n)$ in the average case, but we show that it is $\Theta(n)$ under a wide variety of probability distributions.

The Lindstrom algorithm uses bounded workspace; in particular, it uses no stack or tag bits. Its time complexity is $O(n^2)$ in all cases, but Lindstrom asserted [3] that the average case time complexity is $O(n \log n)$. We show that this assertion is false. In particular, we prove that the average case time complexity is $\Omega(n^2)$.

2. The Lindstrom algorithm. We refer the reader to [3] for a description of the algorithm. It is important to note that λ, the nil address, is *not* an atom. Essentially, descent is always to the left, providing the left link leads to a previously unvisited node. Otherwise, descent is to the right. If no descent is possible because both links are either atomic or lead to previously visited nodes, ascent is initiated and continues until a node with an unvisited right child is found.

At all times, all fully-processed nodes (all of whose descendents have been visited) have a marked mark field (value 1), and all unvisited nodes have an unmarked mark field (value 0). All other nodes must be on the trace path, the path of nodes from the root to the current node. For a node on the trace path, the mark field is 0 if both links are nonatomic and descent from that node was to the right. Otherwise, it is 1.

The algorithm uses link reversal so it is necessary, during ascent, to determine whether that ascent is from the left or the right. If one link field is atomic, there is no problem. Otherwise, the mark bit contains the information.

* Received by the editors February 10, 1984, and in revised form October 15, 1985. This research was supported in part by National Science Foundation grant MCS-82-00362, and by a California State University PAID grant.

† Department of Information and Computer Science, University of California, Irvine, California 92717.

‡ Department of Computer Science, California State University, Dominguez Hills, California 94707.

During descent, if a node whose mark bit is 0 (unmarked) and both of whose link fields are nonatomic is encountered, there is no immediate way to determine whether the node is unvisited or is on the trace path. The algorithm must search for this node upward along the trace path. If the node is unvisited, the time required for this step is clearly $\Omega(k)$, where k is the length of that trace path. Lindstrom's error was his supposition that the length of the trace path is $O(\log n)$ in the average case. Under that hypothesis, the upward searching portion of the algorithm could be done in $O(n \log n)$ time, and other parts of the algorithm are clearly $O(n)$. It may actually be true that the depth of an average node is $O(\log n)$, measured along the shortest possible rooted path. But the Lindstrom algorithm does not visit each node first by the shortest path, but rather by the leftmost path which may be far longer. In fact, by Theorem 2 of this paper, the average case length of that path is $\Theta(n)$.

3. The trace tree. We use the term *list structure* to refer to the collection of all nodes accessible from the root, together with their pointers. The Schorr-Waite, Wegbreit, and Lindstrom marking algorithms visit the nodes of a list structure, X, in the same order (counting only first visits). We call this the *visitation order.*

We denote the *trace tree* of list structure X by $\tau(X)$, or by just τ when X is understood. τ is a binary tree which has exactly the same nodes as does X, but only a subset of the links. For all nodes in X, the father of node x in τ is the node in X from which x was first visited in the visitation order. All other links of τ are λ. As a result of this definition, the preorder of τ will be the visitation order of X, and a path in τ from the root to a given node P will be exactly the trace path when that node is visited for the first time by one of the marking algorithms.

The following algorithm visits all nodes of X in visitation order (i.e., preorder for τ). Let S be a stack.

Visitation Traversal
$S \leftarrow \varnothing$
$P \leftarrow root$
While S is nonempty or P is a new node, do
  If P is an atom or an old node,
    $P \leftarrow$ top of stack S; pop S
  Otherwise (i.e., P is a new node),
    visit P; push $R(P)$ onto stack S; $P \leftarrow L(P)$

4. Time analysis. For any node P, let $depth_\tau(P)$ be the depth of P in τ. The time required for the upward searching portion of the Lindstrom algorithm is clearly $\Theta(\sum_{P \in X} depth_\tau(P))$.

Let *height* be the height of τ. We can put a lower bound on the time complexity of the Lindstrom algorithm as follows.

THEOREM 1. *The time complexity of the Lindstrom algorithm is* $\Omega(n \log n + (height)^2)$.

Proof. Since the average depth of a node must be $\Omega(\log n)$, the number of upward searches must be $\Omega(n \log n)$. On the other hand, let $Q_0, Q_1, \cdots, Q_{height}$ be a longest rooted path in τ, where Q_0 is *root.* Then $\sum_{P \in X} depth_\tau(P) \geqq \sum_{0 \leqq i \leqq height} depth_\tau(Q_i) = \sum_{0 \leqq i \leqq height} i = height\ (height+1)/2$. □

5. Average case analysis. In order to do an average case analysis, it is necessary to have some understanding of what an "average case" is. We shall assume that we are given some distribution on the class of *all* list structures of size n, and that when

we say "average" we shall mean average weighted by this distribution. We will freely use words such as *probability*, *conditional probability* and *expected value* in their usual meanings, where we assume that a list structure has been selected at random from all list structures, using the given distribution. We use Prob (x) to denote the probability that x occurs.

Uniform Distribution. Fix n, and fix a. Let $N[n, a]$ be the number of shapes of list structures which have n nodes and a atoms. Note that a structure which has n nodes has $2n$ address fields, and at least $n-1$ of them must be links for the structure to be connected. Thus, $a \leqq n+1$. The *Uniform Distribution* is that distribution such that the probability that the structure has a given shape with n nodes and a atoms is $1/N[n, a]$.

Weaker conditions more likely to occur in practice. It is unlikely that the uniform distribution would occur in practice, since in any given application there would be some sort of bias. We would like to do an average case analysis which gives valid results for a wide class of reasonable distributions.

Henceforth in this section, fix n, the number of nodes in the structure. We suppose there exists a constant α such that, when any link is examined, the probability that that link contains an atom does not exceed α. Since the structure must be connected, $\alpha \leqq \frac{1}{2}$. However, we do not, in this paper, analyze distributions for which $\alpha = \frac{1}{2}$. We only consider distributions for which $\alpha < \frac{1}{2}$.

We impose one more requirement on our distribution. Suppose that the algorithm has already traversed k nodes, and is currently examining a new link. If the link does not contain an atom, it must contain the address of one of the n nodes of the structure. If all addresses were equally likely, the probability that the address would point to an old node would be k/n. We require only the much weaker assumption that the probability that the address points to an old node is $O(k/n)$.

We hypothesize that the distribution of list structures which will arise in most practical applications will satisfy the above two conditions, which we combine and formalize (below), as the *Weak Randomness Assumption.*

Weak Randomness Assumption. There exist constants $0 \leqq \alpha < \frac{1}{2}$, $\beta > 0$, and $0 < \gamma \leqq 1$ (not dependent on n) such that, if P is a new link and $k \leqq \gamma n$ is the number of nodes already visited:

(i) the probability that P is an atom does not exceed α,

(ii) the probability that P is an old node does not exceed $\beta k/n$.

Intuitively we mean that, although we may allow knowledge of previously examined links to influence the expectation of P, we place a definite bound on that effect. In particular, the probability that a given link is an atom is bounded below $\frac{1}{2}$, and the probability that it points to an old node is of the order of the proportion of old nodes in the structure.

Note that β is sufficiently large, γ is sufficiently small, and α is sufficiently close to $\frac{1}{2}$.

We expect that most distributions which arise in practice satisfy the Weak Randomness Assumption. In particular, we show (in the Appendix) that the Uniform Distribution satisfies this assumption.

THEOREM 2. *In the average case, height* $= \Theta(n)$.

Proof. Refer to the Visitation Traversal Algorithm of § 3. Consider what the situation is after t iterations of the loop in that algorithm. Stack S will have some size, say *size* (t), P will be some node, say *node* (t), and P will be at depth $depth_\tau(P)$ in the trace tree. But note that each element on stack S is the right child of some node in the trace path of P. It follows that $depth_\tau(node\ (t)) \geqq size\ (t)$.

We think of *size* as a stochastic function. It is clear that the number of old nodes after t iterations of the loop of the above algorithm is $\leqq t$, and that

$$size\,(t+1) = \begin{cases} size\,(t)+1 & \text{if } node\,(t) \text{ is a new node,} \\ size\,(t)-1 & \text{otherwise.} \end{cases}$$

By the Weak Randomness Assumption, the probability that *node* (t) is a new node is at least $1-\alpha-\beta t/n$, and the probability that *node* (t) is not a new node is at most $\alpha+\beta t/n$. Therefore (see, for example, [2]),

$$\begin{aligned} E(t) &= \text{the expected value of } size\,(t) \\ &= E(t-1) + \text{Prob}\,(node\,(t) \text{ is a new node}) \\ &\quad - \text{Prob}\,(node\,(t) \text{ is not a new node}) \\ &\geqq E(t-1) + (1-\alpha-\beta t/n) - (\alpha+\beta t/n). \end{aligned}$$

Since $size(0) = 0$, we therefore have

$$E(t) \geqq \sum_{1 \leqq i \leqq t} (1-2\alpha-2\beta i/n) = t(1-2\alpha) - \beta t(t+1)/n.$$

Now choose $t = \lfloor n(1-2\alpha)/(2\beta) \rfloor$ or γn, whichever is smaller. It is seen that the expected value of *size* (t), and hence the expected value of $depth_\tau(node\,(t))$, must be at least $n(1-2\alpha)^2/4\beta - 1$ or, if $t = \gamma n$ was chosen, at least $\beta\gamma(\gamma n - 1)$. □

COROLLARY. *The size of the Wegbreit bit-stack is* $\Theta(n)$ *in the average case.*

THEOREM 3. *The average case time complexity of the Lindstrom algorithm is* $\Theta(n^2)$.

Proof. It is already known [3] that the time is $O(n^2)$, and it is $\Omega(n^2)$ by Theorems 1 and 2. □

6. Open questions. The Schorr-Waite algorithm has both time and space complexity $\Theta(n)$. The Wegbreit algorithm has time complexity $\Theta(n)$ and average case space complexity $\Theta(n)$. The Lindstrom algorithm has bounded space complexity and average case time complexity $\Theta(n^2)$.

A natural question to ask is whether there exists any marking algorithm, the product of whose average time and space complexities is less than quadratic.

The question could arise whether the results of this paper would apply if the Weak Randomness Assumption were modified to allow $\alpha \geqq \frac{1}{2}$. The given proofs depend on α being strictly less than $\frac{1}{2}$, and we conjecture that distributions exist, which satisfy the modified Weak Randomness Assumption with $\alpha = \frac{1}{2}$, for which the average height of the trace tree of the list structure is less than linear.

Appendix. We show here that the Uniform Distribution satisfies the Weak Randomness Assumption.

Fix $0 \leqq probatom < \frac{1}{2}$. Let n be a positive integer, and $a = \lfloor probatom \cdot 2n \rfloor$ (since there are $2n$ links). Suppose that a random list structure of size n with a atoms has been partially traversed, and that exactly $k < n$ nodes have been visited, and that the next step is to examine a link. Let *current* be this link.

There are three possibilities:

1. *IsNew* = "*current* contains the address of a new (i.e., unvisited) node," or
2. *IsOld* = "*current* contains the address of an old (i.e., visited) node," or
3. *IsAtom* = "*current* contains an atomic value."

All probabilities are necessarily conditional probabilities; we have already visited k nodes, and some number of links have already been examined and found to contain addresses of nodes or atoms. We can refer to the portion of the list structure thus

already examined as the *Stem*, and the uniform distribution implies that all list structures consistent with the *Stem* are equally likely.

Let **S** be the set of all list structures with n nodes and a atoms consistent with the *Stem*. The Weak Randomness Assumption deals with the conditional probabilities of the events *IsNew*, *IsOld* and *IsAtom*, and is restated below as Theorem 4.

THEOREM 4. *There exist constants* $\alpha < \frac{1}{2}$, $\beta > 0$, *and* $\gamma > 0$, *dependent on probatom but not dependent on n, such that*

(I) $k < \gamma n \Rightarrow \text{Prob}(IsAtom) \leqq \alpha$,

(II) $\text{Prob}(IsOld) \leqq \beta k/n$.

Proof of Theorem 4. Pick $\alpha = (\frac{1}{2} + probatom)/2$, pick $\beta = 1$, and pick $\gamma = \frac{1}{2} - probatom$. Assume that $k \leqq \gamma n$ (recall, k = number of nodes in the *Stem*). We first state and prove two lemmas:

LEMMA 4.1. $\text{Prob}(IsOld) \leqq (k/n) \cdot [\text{Prob}(IsOld) + \text{Prob}(IsNew)]$.

LEMMA 4.2. $\text{Prob}(IsAtom) \leqq a/(2n - 2k + 1)$.

Proof of Lemma 4.1. Recall that **S** is the set of all link structures with n nodes and a atoms which are consistent with the *Stem*, the portion of the list structure already examined. Let **T** be the subset of **S** consisting of those list structures which would remain connected if *current* is changed to an atom. Under the uniform distribution assumption, the list structure must be some member of **S**, and is equally likely to be any one of them. If it is a member of **S** but not of **T**, then *current* must contain the address of a new node, by definition of **T**. On the other hand, among all members of **T**, the value of *current*, if it is the address of a node, is equally likely to point to any node in the entire structure. Thus,

$$\begin{aligned}\text{Prob}(IsNew)/[\text{Prob}(IsNew) + \text{Prob}(IsOld)] &= (|\mathbf{S}| - |\mathbf{T}| + |\mathbf{T}|(n-k)/n)/|\mathbf{S}| \\ &= 1 - (k/n)|\mathbf{T}|/|\mathbf{S}| \\ &\geqq 1 - k/n.\end{aligned}$$

Lemma 4.1 follows directly. □

Proof of Lemma 4.2. Let **X** be the subset of **S** consisting of those structures where *current* is an atom. Let **Y** be the subset of **S** consisting of those structures where *current* contains the address of a node. Note that **X** and **Y** are disjoint, and their union is **S**. Lemma 4.2 can be rephrased to state that $|\mathbf{X}|/|\mathbf{S}| \leqq a/(2n - 2k + 1)$, i.e., that $(2n - 2k + 1)|\mathbf{X}| \leqq a|\mathbf{S}|$.

In order to compare the cardinalities of $|\mathbf{X}|$ and $|\mathbf{S}|$, we shall employ a representation graph of the set of list structures. Define a bipartite graph G as follows: the nodes correspond to list structures in $\mathbf{S} = \mathbf{X} \cup \mathbf{Y}$. Each edge of G is between an element of **X** and an element of **Y**. There is an edge between $X \in \mathbf{X}$ and $Y \in \mathbf{Y}$ if and only if Y can be obtained from X by exchanging the values of exactly two link fields, one of which is *current*, an atom, and the other of which is some link field of X which is not an atom.

We will finish the proof by analyzing the degrees of the nodes of the graph G. If $X \in \mathbf{X}$, the number of $Y \in \mathbf{Y}$ connected to X is equal to the number of choices of the other link, *other*, whose value must be exchanged with that of *current* to obtain Y. *Other* must not be an atom, and must be examined after *current* in the visitation order, since Y and X are both consistent with the *Stem*. The number of links in list structure X is $2n$, of which a are atoms, and at most $2k - 1$ precede *current* in the visitation order. Thus, the degree of X in the graph G is at least $2n - a - 2k + 1$.

If $Y \in \mathbf{Y}$, for each X connected to Y in G there must be a link field *other* in Y which, when exchanged with *current*, yields X. Since *other* must be an atom in Y, there are no more than a choices of *other*. Hence Y has degree no greater than a in G.

It follows that $(2n-a-2k+1)\cdot|\mathbf{X}| \leqq a\cdot|\mathbf{Y}|$. Add $a\cdot|\mathbf{X}|$ to both sides to obtain $(2n-2k+1)\cdot|\mathbf{X}| \leqq a\cdot|\mathbf{X}|+a\cdot|\mathbf{Y}| = a\cdot|\mathbf{S}|$. Finally, Prob $(IsAtom) = |\mathbf{X}|/|\mathbf{S}| \leqq a/(2n-2k+1)$. □

We now complete the proof of Theorem 4. Part (II) follows immediately from Lemma 4.1, since Prob $(IsOld)$ + Prob $(IsNew) \leqq 1 = \beta$.

To prove part (I) of the theorem we use Lemma 4.2. First, note that if $probatom = 0$, it is trivial, since $a = 0$. Otherwise, assume that $probatom > 0$. We will need to make use of a well-known algebraic inequality: if x, y are positive real numbers, then $1/(x+y) \leqq 1/(4x)+1/(4y)$. Also, by hypothesis of (I),

$$(**) \qquad k < \gamma n = n/2 - probatom\cdot n.$$

Then,

$$\begin{aligned}
\text{Prob}\,(IsAtom) &\leqq a/(2n-2k+1) && \text{from Lemma 4.2,}\\
&\leqq a/[n+2n\cdot probatom] && \text{from (**),}\\
&\leqq a/(4n)+a/(8n\cdot probatom) && \text{by well-known algebraic inequality,}\\
&\leqq probatom/2+1/4 && \text{since } a \leqq probatom\cdot 2n,\\
&= \alpha. && \square
\end{aligned}$$

REFERENCES

[1] J.-L. BAER AND M. FRIES, *On the Efficiency of Some List Marking Algorithms*, Inform. Process. 77, IFIP, North-Holland, Amsterdam, 1977, pp. 751-756.

[2] P. G. HOEL, S. C. PORT AND C. J. STONE, *Introduction to Stochastic Processes*, Houghton Mifflin, Boston, MA, 1972.

[3] G. LINDSTROM, *Copying list structures in bounded workspace*, Comm. ACM, 17 (1974), pp. 198-202.

[4] H. SCHORR AND W. M. WAITE, *An efficient machine independent procedure for garbage collection in various list structures*, Comm. ACM, 10 (1967), pp. 501-506.

[5] B. A. WEGBREIT, *A space efficient list structure tracing algorithm*, IEEE Trans. Comput., C-21 (1972), pp. 1009-1010.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
015

SEARCHING IN TREES, SERIES-PARALLEL AND INTERVAL ORDERS*

U. FAIGLE†, L. LOVÁSZ‡, R. SCHRADER† AND GY. TURÁN§

Abstract. Linial and Saks [2] have shown that $O(\log N)$ evaluations of an order preserving map $f: P \to \mathbb{R}$ are necessary and sufficient to determine whether $\alpha \in f(P)$, where N is the number of ideals of P and $\alpha \in \mathbb{R}$ is a given real number. In this paper, we investigate the problem of how to perform the evaluations so that Linial and Saks' bound is guaranteed, and solve the problem for the classes of interval and series-parallel orders and hence, in particular, for rooted trees. We observe that the greedy-type binary search algorithm, which is optimal for chains, already need not be optimal for general rooted trees. We furthermore discuss the computational complexity of the general search problem and obtain results indicating that the general problem might be hard.

Key words. searching trees, series-parallel orders, interval orders, complexity

AMS(MOS) subject classifications. 68E05, 06A10

1. Introduction.

The complexity of retrieving data depends on the order structure of the stored data. For instance, if $f: P \to \mathbb{R}$ is an arbitrary function assigning real numbers to the elements of the finite set P, $|P|$ evaluations of f will generally be necessary (and sufficient) to decide whether a given real number $\alpha \in \mathbb{R}$ is in the image of f. If P is *linearly* ordered and $f: P \to \mathbb{R}$ is order-preserving, i.e., $i \leqq j$ implies $f(i) \leqq f(j)$, one may carry out a binary search and decide whether $\alpha \in f(P)$ in at most $\log_2 |P|$ evaluations of f.

Linial and Saks [2] have considered the corresponding general search problem: Let P be a finite (partially) ordered set and $f: P \to \mathbb{R}$ an (unknown) order-preserving function. Given $\alpha \in \mathbb{R}$, decide whether $\alpha \in f(P)$ by evaluating f at elements of P.

Linial and Saks have shown that the problem can always be solved in at most $c \cdot \log_2 N(P)$ evaluations of f, where $N(P)$ is the number of ideals of P, and c is a constant with value $c \approx 3.73$. They also show that $\log_2 N(P)$ evaluations are necessary. The upper bound is implied by the following remarkable theorem:

In every finite ordered set P, there exists an element x for which

$$\delta_0 \leqq \frac{N(x)}{N(P)} \leqq 1 - \delta_0,$$

where $N(x)$ is the number of ideals containing x and $\delta_0 = \frac{1}{4}(3 - \log_2 5) \approx 0.17$. (A result of Sands [5] shows that the best possible value for δ_0 is at most 0.228.)

Linial and Saks use an intricate averaging argument together with the FKG-inequality in order to obtain an existence proof, which, however, does not provide an efficient algorithm to find an element x with the desired property.

* Received by the editors July 17, 1984, and in revised form October 25, 1985. This work was done at the Institut für Operations Research, Universität Bonn, Bonn, West Germany, and was supported by the joint research project "Algorithmic Aspects of Combinatorial Optimization" of the Hungarian Academy of Sciences (Magyar Tudományos Akadémia) and the German Research Association (Deutsche Forschungsgemeinschaft, SFB 21).

† Institut für Okonometrie und Operations Research, Rheinische Friedrich-Wilhelms-Universät Bonn, D-5300 Bonn 1, West Germany.

‡ On leave from ELTE Matematikai Intezet, Budapest, Hungary.

§ On leave from the Automata Theory Res. Gr. of the Hungarian Academy of Sciences, József Attila University, Szeged, Hungary.

Thus Linial and Saks' result raises the questions:

(i) For what classes of ordered sets can the best constant δ_0 be determined?

(ii) For what classes of ordered sets can a "good" point be determined efficiently?

(iii) How difficult is it to determine a "good" point in general?

In this paper we investigate some classes of orders with respect to (i) and (ii). For series-parallel and interval orders (both generalize the classes of linear orders and antichains discussed above), it is shown that the sharper bound $\delta_0 = \frac{1}{4}$ holds (and is the best possible constant for these classes). For the class of trees we derive the best possible constant $\delta_0 = \frac{1}{3}$. These bounds imply sharper bounds for the number of evaluations necessary in orders belonging to the classes considered. Moreover, there exists a polynomial algorithm for series-parallel and interval orders to find a point x with ratio closest to $\frac{1}{2}$. (For comparison we note that for linear orders the above search algorithm reduces to binary search.) We point out that the "greedy" algorithm evaluating f at this element is not optimal (although it is clearly optimal for linear orders). These results are presented in §§ 3 and 4. (We note that in the earlier paper [1], Linial and Saks give sharp bounds for the complexity in terms of $\log_2 N$ in the case of rooted forests.)

In § 5, we give some arguments indicating that the determination of a "good" point may be a hard computational problem for general ordered sets. A polynomial algorithm for this problem would imply that the number of ideals can be approximated within a factor of $1+\varepsilon$ in polynomial time.

2. Preliminaries. Let $(P, \leqq)$ be a (finite) ordered set with $|P| = n$. An element x is a *lower* (resp. *upper*) *cover* of y in P if $x < y$ (resp. $x > y$) and there is no z with $x < z < y$ (resp. $x > z > y$). The set of minimal (resp. maximal) elements is MIN P(resp. MAX P). The *dual* of $(P, \leqq)$ is $(P, \leqq')$ with $x \leqq' y$ iff $y \leqq x$. For every $x \in P$ we let $S(x) = \{y \in P: x < y\}$. A subset $I \subseteq P$ is an *ideal*, if $x \in I$ and $y < x$ implies $y \in I$. The number of ideals of P is denoted by N, and the number of ideals containing a given element x is denoted by $N(x)$. A subset $F \subseteq P$ is a *filter*, if $x \in F$ and $y > x$ implies $y \in F$. The number of filters is M and the number of filters containing x is $M(x)$.

The class of *series-parallel orders* is defined inductively as follows:

(i) a single node is series-parallel;

(ii) if $(P_1, \leqq_1)$ and $(P_2, \leqq_2)$ are series-parallel with $P_1 \cap P_2 = \emptyset$, then the *series composition* $P = (P_1 \cup P_2, \leqq)$ is series-parallel, where $x \leqq y$ in P iff $x \leqq_1 y$ or $x \leqq_2 y$ or $x \in P_1$ and $y \in P_2$;

(iii) If $(P_1, \leqq_1)$ and $(P_2, \leqq_2)$ are series-parallel with $P_1 \cap P_2 = \emptyset$, then the *parallel composition* $(P_1 \cup P_2, \leqq)$ is series-parallel, where $x \leqq y$ in P iff $x \leqq_1 y$ or $x \leqq_2 y$.

An *interval order* is an ordered set whose elements are nonempty closed intervals in $\mathbb{R}$ such that for all $x, y \in P$ $x < y$ in P iff $a < b$ in $\mathbb{R}$ for all $a \in x$ and $b \in y$.

We shall need the following well-known property of interval orders.

(2.1) For any two elements $x, y \in P$, either $S(x) \subseteq S(y)$ or $S(y) \subseteq S(x)$.

This property easily follows from the definition. In fact, it characterizes interval orders (see, e.g., Papadimitriou and Yannakakis [3]).

An element x of an interval order $(P, \leqq)$ is called *S-maximal* if $S(y) \subseteq S(x)$ for all $y \in P$. It is obvious that every S-maximal element of P is a member of MIN P.

FACT:

(2.2) *If x is S-maximal and $y \in P \backslash$MIN P, then $x < y$.*

Proof. Let $z \in$ MIN P with $y \in S(z)$. Since x is S-maximal we have $S(z) \subseteq S(x)$. □

An ordered set is a *tree* if its Hasse diagram is a tree as a graph. It is a *rooted tree* if when directing edges (x, y) of the Hasse diagram from y to x for $x<y$, we obtain an outtree. (Thus a rooted tree is series-parallel, while a general tree is not necessarily series-parallel.) An ordered set is a *forest* if it is the parallel composition of trees.

FACT. *If P is an ordered set, then*

$$N = M \text{ and } N(x)+M(x) = N \text{ for every } x \in P. \tag{2.3}$$

3. Recursion formulas for series-parallel and interval orders. In this section we give simple recursion formulas for the computation of the values of N and $N(x)$ for series-parallel orders and interval orders.

LEMMA 1. *Let P_1 and P_2 be two ordered sets, N_1 and N_2 the respective number of ideals and $N_1(x)$, $N_2(x)$ the corresponding ideal counting functions. Then the following formulas hold:*

(i) *If P is the series composition of P_1 and P_2 then $N = N_1+N_2-1$ and*

$$N(x)=\begin{cases} N_1(x)+N_2-1 & \text{for } x \in P_1, \\ N_2(x) & \text{for } x \in P_2. \end{cases}$$

(ii) *If P is the parallel composition of P_1 and P_2 then $N = N_1 \cdot N_2$ and*

$$N(x)=\begin{cases} N_1(x) \cdot N_2 & \text{for } x \in P_1, \\ N_2(x) \cdot N_1 & \text{for } x \in P_2. \end{cases}$$

Proof. It is easily verified. □

We remark that the labeling rule of Steiner [6] for the class of ordered sets with order dimension at most two may also be used to determine the numbers N and $N(x)$ efficiently. Even for the subclass of series-parallel orders, Steiner's rule yields a different algorithm than Lemma 1, which is based on series-parallel decomposition. It would be interesting to know whether Theorem 4 holds, more generally, for ordered sets of dimension at most two.

LEMMA 2. *Let P be an ordered set and $a \in \text{MIN } P$ such that for every $y \in P \backslash \text{MIN } P$, $a<y$ holds. Consider $P' = P\backslash\{a\}$ and let N' be the number of ideals of P' and $N'(x)$ be the ideal counting function of P'. Then*

$$N = N'+2^{|\text{MIN } P|-1},$$

$$N(x)=\begin{cases} N' & \text{for } x = a, \\ N'(x) & \text{for } x \in P\backslash\text{MIN } P, \\ N'(x)+2^{|\text{MIN } P|-2} & \text{for } x \in \text{MIN } P\backslash\{a\}. \end{cases}$$

Proof. If I is an ideal of P' then $I \cup \{a\}$ is an ideal of P. This establishes a bijection between the ideals of P' and of the ideals of P containing a. The ideals of P not containing a can only consist of minimal elements of P since a is S-maximal.

Conversely, every subset of MIN $P\backslash\{a\}$ is an ideal of P not containing a. Hence $N = N'+2^{|\text{MIN } P|-1}$.

Similarly, one proves the recursion for $N(x)$. □

The two lemmas above imply

THEOREM 3. *If P is a series-parallel order or an interval order and $x \in P$ then N and $N(x)$ can be computed in polynomial time.*

Proof. First assume P is a series-parallel order. We can find a decomposition tree for P in polynomial time using, e.g., the algorithm of Valdes, Tarjan and Lawler [7]. Based on this decomposition and Lemma 1 we can compute N and $N(x)$ recursively.

If P is an interval order, then we can find in polynomial time a sequence $(x_1, x_2, \ldots, x_n)$ of elements of P such that x_i is S-maximal in the interval order $P\backslash\{x_{i+1}, \ldots, x_n\}$. By (2.2) and Lemma 2 we again can compute N and $N(x)$ recursively. □

Remark. Note that Theorem 3 in fact yields an efficient algorithm for the determination of a "good" point x: one simply computes the ratios $N(x)/N$ for all $x \in P$ according to the updating formulas given in Lemma 1 and selects a $x \in P$ with good ratio. For the two extreme cases discussed in the Introduction, of course, explicit updating is not necessary. For linear orders a "best" point x is a point closest to the "middle," whereas for antichains any point may be chosen.

4. Bounds for δ. We now turn to the question of determining the possible values of δ for series-parallel orders, interval orders and trees. First we consider an example.

Example. Let $P^{(n)} = \{x_1, \cdots, x_n, y_1, \cdots, y_n\}$ be an ordered set with $x_i < y_j$ for $1 \leqq i, j \leqq n$. Then $N = 2^{n+1} - 1$, $N(x_i) = 2^n + 2^{n-1} - 1$ for $1 \leqq i \leqq n$ and $N(y_j) = 2^{n-1}$ for $1 \leqq j \leqq n$. Hence $N(x)/N$ is either $\frac{1}{4} + \varepsilon_n$ or $\frac{3}{4} - \varepsilon_n$, where $\varepsilon_n \to 0$ when $n \to \infty$.

Since $P^{(n)}$ is a series-parallel as well as an interval order, this example shows:

(4.1) For series-parallel and interval orders, the best δ is at most $\frac{1}{4}$.

We now show that in the first two cases $\delta = \frac{1}{4}$ can be guaranteed, whereas in the third case the bound $\delta = \frac{1}{3}$ is sharp.

THEOREM 4. *Let P be a series-parallel order. Then there exists an $x \in P$ with $\frac{1}{4} \leqq N(x)/N \leqq \frac{3}{4}$.*

Proof. Proceeding by induction on $|P| = n$, we order the elements $x_1, x_2, \cdots, x_n$ of P such that $N(x_1) \leqq N(x_2) \leqq \cdots \leqq N(x_i) \leqq \cdots \leqq N(x_n)$. Then we claim for $i = 1, \cdots, n-1$,

$$N(x_{i+1}) - N(x_i) \leqq \tfrac{1}{2}N. \tag{4.2}$$

Observing that a maximal (minimal) element $x \in P$ must satisfy

$$\frac{N(x)}{N} \leqq \frac{1}{2}\left(\frac{N(x)}{N} \geqq \frac{1}{2}\right), \tag{4.3}$$

Theorem 4 will immediately follow from the claim since x_1 is maximal and x_n is minimal in P.

We prove (4.2) by considering two cases.

Case 1. P is the series composition of P_1 and P_2. As $x < y$ for $x \in P_1$, $y \in P_2$, we have $x_1, \cdots, x_i \in P_2$, $x_{i+1}, \cdots, x_n \in P_1$, for some $1 \leqq i \leqq n-1$. For $j \leqq i-1$, we have

$$N(x_{j+1}) - N(x_j) = N_2(x_{j+1}) - N_2(x_j) \leqq \tfrac{1}{2}N_2 < \tfrac{1}{2}N$$

by Lemma 1 and induction. For $j \geqq i+1$ we have

$$N(x_{j+1}) - N(x_j) = N_1(x_{j+1}) - N_1(x_j) \leqq \tfrac{1}{2}N_1 < \tfrac{1}{2}N$$

also by Lemma 1 and induction. In the remaining case $j = i$ we have to show

$$N(x_{i+1}) - N(x_i) \leqq \tfrac{1}{2}N.$$

Using Lemma 1, this is equivalent to

$$N_1(x_{i+1}) + N_2 - 1 - N_2(x_i) \leqq \tfrac{1}{2}(N_1 + N_2 - 1).$$

But x_i must be a minimal element in P_2 and x_{i+1} must be a maximal element in P_1. Thus

$$N_1(x_{i+1}) \leqq \tfrac{1}{2}N_1,$$

$$N_2 - 1 - N_2(x_i) \leqq (N_2 - N_2(x_i)) - \tfrac{1}{2} \leqq \tfrac{1}{2}(N_2 - 1),$$

and (4.2) follows.

Case 2. P is the parallel composition of P_1 and P_2. Consider $N(x_{i+1})$ and $N(x_i)$. If $x_i, x_{i+1} \in P_l$ $(l=1,2)$ then

$$N(x_{i+1}) - N(x_i) = (N_l(x_{i+1}) - N_l(x_i)) \cdot N_{3-l} \leqq \tfrac{1}{2} N_l \cdot N_{3-l} = \tfrac{1}{2} N$$

by Lemma 1 and induction. If there exists a $j > i+1$ such that $x_j, x_i \in P_l$ $(l=1,2)$ or a $k < i$ such that $x_{i+1}, x_k \in P_l$ $(l=1,2)$ then

$$N(x_{i+1}) - N(x_i) \leqq N(x_j) - N(x_i) \leqq \tfrac{1}{2} N,$$

and

$$N(x_{i+1}) - N(x_i) \leqq N(x_{i+1}) - N(x_k) \leqq \tfrac{1}{2} N$$

follows similarly.

If we can find neither a j nor a k with the above property, then we have $x_1, \cdots, x_i \in P_l, x_{i+1}, \cdots, x_n \in P_{3-l}$. Taking (4.3) into consideration this implies $N(x_i) = \frac{1}{2} N$ for $i = 1, \ldots, n$ and the claim holds again (in fact, in this case P is an antichain).

We now turn to the case of interval orders.

THEOREM 5. *Let P be an interval order. Then there exists an $x \in P$ with*

$$\tfrac{1}{4} \leqq N(x)/N \leqq \tfrac{3}{4}.$$

Proof. Let $(x_1, \cdots, x_n)$ be the sequence selected in the proof of Theorem 3, i.e., assume x_i is S-maximal in $P_i = P \backslash \{x_{i+1}, \cdots, x_n\}$. Denote by N_i the number of ideals of P_i and by $N_i(x)$ the ideal counting function of P_i. Lemma 2 implies

$$N = 1 + \sum_{j=1}^{n} 2^{|\mathrm{MIN}\, P_j| - 1}.$$

Then

$$N(x_{i+1}) - N(x_i) \leqq N_{i+1}(x_{i+1}) + \sum_{j=i+2}^{n} 2^{|\mathrm{MIN}\, P_j| - 2} - N_i(x_i)$$

using Lemma 2 for x_{i+1} and $N_i(x_i) \leqq N(x_i)$,

$$N_{i+1}(x_{i+1}) = N_i, \qquad N_i(x_i) = N_{i-1}$$

from Lemma 2. Thus

$$\begin{aligned} N(x_{i+1}) - N(x_i) &\leqq N_i - N_{i-1} + \sum_{j=i+2}^{n} 2^{|\mathrm{MIN}\, P_j| - 2} \\ &= \frac{1}{2} 2^{|\mathrm{MIN}\, P_i|} + \frac{1}{2} \sum_{j=i+2}^{n} 2^{|\mathrm{MIN}\, P_j| - 1} \\ &\leqq \frac{1}{2} \left(N_i + \sum_{j=i+2}^{n} 2^{|\mathrm{MIN}\, P_j| - 1} \right) \leqq \frac{1}{2} N, \end{aligned}$$

where we used $2^{|\mathrm{MIN}\, P_i|} \leqq N_i$. Similarly

$$\begin{aligned} N(x_i) - N(x_{i+1}) &\leqq N_i(x_i) + \sum_{j=i+1}^{n} 2^{|\mathrm{MIN}\, P_j| - 2} - N_{i+1}(x_{i+1}) \\ &= N_{i-1} - N_i + \sum_{j=i+1}^{n} 2^{|\mathrm{MIN}\, P_j| - 2} \leqq \frac{1}{2} N. \end{aligned}$$

This implies $|N(x_{i+1}) - N(x_i)| \leqq \frac{1}{2} N$.

Furthermore, x_1 must be a maximal element and x_n must be minimal. Thus, by (4.3), $N(x_1)/N \leqq \frac{1}{2}$, $N(x_n)/N \geqq \frac{1}{2}$. If $\frac{1}{4} \leqq N(x_1)/N$, we are done; otherwise let x_{i-1} be the last element in the sequence $x_1, \cdots, x_n$ with $N(x_{i-1})/N < \frac{1}{4}$. Then by the above,

$$\tfrac{1}{4} \leqq N(x_i)/N \leqq \tfrac{3}{4}.$$

□

Next we discuss the cases of forests. Observing Lemma 2 we can restrict ourselves to trees.

The series composition of a k-element antichain and a 2^{k-1} element chain yields a family of trees showing that the best δ_0 we can have is $\frac{1}{3}$ (if $k=1$, we have the 2-element chain).

First we prove a lemma.

LEMMA 6. *Let P be a tree, I an ideal of P with* MIN $P \subseteq I$, MAX $P \cap I = \emptyset$. *Then either there exists an element $y \notin I$ s.t. y has a unique lower cover x and $x \in I$, or there exists an element $y \in I$ s.t. y has a unique upper cover x and $x \notin I$.*

Proof. Let $\alpha = (x_1, \cdots, x_m)$ be a path in the Hasse diagram of P with the maximal number of edges, connecting I and $P - I$.

Assume $x_1, \cdots, x_k \in I$, $x_{k+1} \notin I$. This implies $x_k < x_{k+1}$. We claim x_{k+1} is the unique upper cover of x_k. If this is not the case, MAX $P \cap I = \emptyset$ and the fact that P is a tree imply the existence of a path $\beta = (y_1, \cdots, y_l, x_k)$ s.t. $y_1 \notin I$ and $\beta \cap \alpha = \{x_k\}$. Then $\alpha' := (y_1, \cdots, y_t, x_k, x_{k+1}, \cdots, x_m)$ is a path containing more edges, connecting I and $P - I$, than α. This shows that x_{k+1} is the unique upper cover of x_k.

When $x_1, \cdots, x_k \notin I$, $x_{k+1} \in I$, an analogous argument shows that x_{k+1} is the unique lower cover of x_k. □

THEOREM 7. *Let P be a tree. Then there exists an $x \in P$ with*

$$\frac{1}{3} \leqq \frac{N(x)}{N} \leqq \frac{2}{3}.$$

Proof. Assume the statement is false for P. Let $I := \{x \in P: N(x)/N > \frac{2}{3}\}$. Then I is an ideal of P and MIN $P \subseteq I$, MAX $P \cap I = \emptyset$ hold. For every $y \notin I$ we have $N(y)/N < \frac{1}{3}$.

Using Lemma 6, assume first that there exists a $y \notin I$ s.t. y has a unique lower cover x and $x \in I$.

$$\text{Then } N(x) = N(y) + |\{\mathcal{F}: \mathcal{F} \text{ is an ideal, } x \in \mathcal{F}, y \notin \mathcal{F}\}|.$$

But $|\{\mathcal{F}: \mathcal{F} \text{ is an ideal, } x \in \mathcal{F}, y \notin \mathcal{F}\}| \leqq N(y)$ as adding y to any such $\mathcal{F}$ we get an ideal containing y (thereby using that x is the unique lower cover of y).

$$\text{Thus } N(x) \leqq 2N(y), \text{ contradicting } N(x) > \tfrac{2}{3}N,\ N(y) < \tfrac{1}{3}N.$$

If, using Lemma 6, we get an element $y \in I$ with a unique upper cover x and $x \notin I$, we apply the same argument to the dual of P. (Fact (2.3) implies that if P is a counterexample, its dual is a counterexample as well.) □

5. Searching algorithms. The bounds above have direct implications for the complexity of the search problem in the corresponding classes of ordered sets.

COROLLARY 8. *The search problem for series-parallel and interval orders can be solved with at most $2.41 \cdot \log_2 N$ evaluations of f. In the case of forests, at most $1.71 \cdot \log_2 N$ evaluations suffice.*

Proof. The algorithm is identical to the one indicated in [2] and is based on evaluating at x with $N(x)/N$ closest to $\frac{1}{2}$.

If $f(x) = \alpha$ we are done. If $f(x) > \alpha$ ($f(x) < \alpha$) we can reduce the search to $P' := \{y \in P: y \not\geqq x\}$, ($P' := \{y \in P: y \not\leqq x\}$) and thus proceed recursively. Using

$$N(P') = \text{the number of ideals in } P \text{ not containing } x,$$

($N(P') =$ the number of ideals containing x), the number of steps in the case of series-parallel and interval orders is bounded by $\log_{4/3} N = (2 - \log_2 3)^{-1} \cdot \log_2 N$. The

same argument gives the bound for forests. (Note that all these classes are closed under forming suborders.) □

The algorithm described above is "greedy" in the sense that it always performs the next evaluation at the element with ideal ratio closest to $\frac{1}{2}$. Clearly in the case of chains this strategy is optimal. However, the examples below show that already for rooted trees the greedy algorithm can have a relatively poor performance.

Let $Q(P)$ denote the number of evaluations used by an optimal algorithm for P (such an algorithm clearly exists), and $G(P)$ denote the number of evaluations used by the greedy algorithm. (As usual, both are worst-case measures.)

THEOREM 9.

(a) $\overline{\lim}\{G(P)/Q(P)\colon P \textit{ rooted tree}\} \geqq 2-\frac{1}{3}\log_2 5 \geqq 1.226$;

(b) $\overline{\lim}\{G(P)/Q(P)\colon P \textit{ series-parallel}\} \geqq \frac{3}{2}$.

Proof. a) Let P have elements x_i^j $(1 \leqq i \leqq n,\ 1 \leqq j \leqq l)$, $y_i (1 \leqq i \leqq k)$ with covering pairs (x_i^j, x_i^{j+1}) for $1 \leqq i \leqq n, 1 \leqq j \leqq l-1$, (x_j^l, y_1) for $1 \leqq i \leqq n$, and (y_i, y_{i+1}) for $1 \leqq i \leqq k-1$. (Thus P is the series composition of n parallel l-chains and a k-chain.) Then

$$N = (l+1)^n + k, \quad N(y_i) = k-i+1, \quad N(x_i^j) = (l-j+1)\cdot(l+1)^{n-1} + k.$$

If $k > (l-1)(l+1)^n$, then

$$\frac{N(x_i^j)}{N} \geqq \frac{N(x_i^l)}{N} = \frac{(l+1)^{n-1}+k}{(l+1)^n+k} > \frac{l}{(l+1)}.$$

In this case the greedy algorithm chooses y_i with

$$k-i+1 = \lfloor \tfrac{1}{2}((l+1)^n + k) \rfloor,$$

implying

$$i \geqq \tfrac{1}{2}(k-(l+1)^n).$$

Thus putting

$$i_0 := k, \qquad i_j := \tfrac{1}{2}(i_{j-1} - 1 - (l+1)^n),$$

if for some r, $i_r > (l+1)^{n+1}$, then the greedy algorithm chooses elements in the y-chain for at least r steps—assuming that all evaluations give larger values than the original number α we are searching for. In each l-chain the algorithm can be forced to ask at least $\lceil \log_2 (l+1) \rceil$ questions. Thus

$$G(P) \geqq r + \lceil \log_2 (l+1) \rceil \cdot n. \tag{5.1}$$

Let $k := 2^{n\cdot\lceil \log_2 (l+1)\rceil}$. Then $Q(P) \leqq n \cdot \lceil \log_2 (l+1) \rceil + 1$ follows from the strategy that evaluates y_1 first. (In fact, $Q(P) = n \cdot \lceil \log_2 (l+1) \rceil + 1$.)

The recursion above gives

$$i_j = \frac{1}{2^j}\cdot k - \left(1 - \frac{1}{2^j}\right)((l+1)^n + 1).$$

Thus $i_r > (l-1)(l+1)^n$ for $r = \lfloor n(\lceil \log_2 (l+1) \rceil - \log_2 (l+1)) - \log_2 (l+1) \rfloor$ and substituting into (5.1) we get

$$\frac{G(P)}{Q(P)} \geqq \frac{\lfloor (2\lceil \log_2 (l+1) \rceil - \log_2 (l+1)) \cdot n - \log_2 (l+1) \rfloor}{\lceil \log_2 (l+1) \rceil \cdot n + 1}.$$

Putting $l := 4$, we get a).

b) This case is proved similarly with P being the series composition of 2 n-antichains and a 2^{2^n}-chain. The computation is omitted. □

6. Relations to ideal counting problems. The question of course arises whether a polynomial algorithm exists in general for finding an element in an arbitrary ordered set guaranteeing an ideal ratio between δ and $1-\delta$. In this section we will give some partial results into this direction. It appears unlikely that such an algorithm exists: in particular, we show that its existence would also imply the existence of a polynomial algorithm for approximating the number of ideals of an arbitrary ordered set.

THEOREM 10. *The following statements are equivalent*:

(1) *there is a* $\delta > 0$ *and a polynomial algorithm A which, given an order P, determines an element x of P with* $\delta \leqq N(x)/N \leqq 1-\delta$;

(2) *for every* $\delta > 0$ *there is a polynomial algorithm* A_δ *which, given an order P, determines an element x of P with* $\delta_0 - \delta \leqq N(x)/N \leqq 1-(\delta_0-\delta)$ *(where* $\delta_0 = 0.17\cdots$ *is given in the Introduction)*;

(3) *there is an* $r > 0$ *and a polynomial algorithm B which, given an order P, determines an integer* $B(P)$ *with* $1/r \leqq B(P)/N \leqq r$;

(4) *for every* $\varepsilon > 0$ *there is a polynomial algorithm* B_ε *which, given an order P, determines an integer* $B_\varepsilon(P)$ *with* $1-\varepsilon \leqq B_\varepsilon(P)/N \leqq 1+\varepsilon$.

Proof. We show (1)⇒(3)⇒(4)⇒(2). (The implication (2)⇒(1) is obvious.)

(1)⇒(3). Let C_l be an antichain on l elements and P_l be the series composition of C_l and P (i.e. elements of C_l are smaller than elements of P). N_l is the number of ideals in P_l and $N_l(x)$ is the corresponding ideal count function. From Lemma 1, we have

$$N_l = N + 2^l - 1;$$

$$N_l(x) = \begin{cases} N(x) & \text{if } x \in P, \\ N + 2^{l-1} - 1 & \text{if } x \in C_l. \end{cases} \tag{6.1}$$

As every P with $|P| = n$ has $n+1 \leqq N \leqq 2^n$ (the lower bound follows, e.g., by considering ideals of a linear extension of P), we can assume that n satisfies $(n+1)/(n+2) > 1-\delta$ as the small cases can be checked directly.

Then for $l = 1$ and $x \in C_1$ it follows from (6.1) that

$$\frac{N_1(x)}{N_1} = \frac{N}{N+1} \geqq \frac{n+1}{n+2} > 1-\delta. \tag{6.2}$$

Thus applying algorithm A to P_1, we get an element $x \in P$. On the other hand for $l = n + \lceil \log_2(1/\delta) \rceil$ and $y \in P$, (6.1) yields

$$\frac{N_l(y)}{N_l} = \frac{N(y)}{N + 2^l - 1} < \frac{N}{2^l} \leqq 2^{n-l} \leqq \delta. \tag{6.3}$$

Thus applying algorithm A to P_l, we get an element $y \in C_l$.

Hence applying the algorithm to $P_1, P_2, \cdots, P_l$, we get an i with $1 \leqq i \leqq l-1$ such that for P_i the algorithm A gives $x_i \in P$ and for P_{i+1} it gives $x_{i+1} \in C_{i+1}$. Then

$$\delta \leqq \frac{N_i(x_i)}{N_i} = \frac{N(x_i)}{N + 2^i - 1} \leqq \frac{N}{2^i} \tag{6.4}$$

and

$$\frac{N_{i+1}(x_{i+1})}{N_{i+1}} = \frac{N + 2^i - 1}{N + 2^{i+1} - 1} \leqq 1 - \delta \tag{6.5}$$

imply

$$\frac{\delta}{1-\delta} \leqq \frac{2i}{N} \leqq \frac{1}{\delta}. \tag{6.6}$$

Thus we get an algorithm B for $r = 1/\delta$ by applying A to $P_1, \cdots, P_l$ and putting $B(P) = 2^i$ as above.

(3)$\Rightarrow$(4). Consider the parallel composition P_k of k copies $P^{(1)}, \cdots, P^{(k)}$ of P. Then for the number N_k of ideals in P_k we have $N_k = N^k$. Algorithm B determines an integer $B(P_k)$ with

$$\frac{1}{r} \leqq \frac{B(P_k)}{N^k} \leqq r. \tag{6.7}$$

Thus

$$\frac{1}{r^{1/k}} \leqq \frac{B(P_k)^{1/k}}{N} \leqq r^{1/k}. \tag{6.8}$$

Hence for every $\varepsilon > 0$ we can determine a k s.t. $r^{1/k} \leqq 1+\varepsilon$, $r^{-1/k} \geqq 1-\varepsilon$ and the algorithm B_ε can be the computation of $\lceil B(P_k)^{1/k} \rceil$.

(4)$\Rightarrow$(2). Given an order P and $x \in P$, put $L(x) := \{y \in P: y \leqq x\}$. Then

$$N(x) = \text{the number of ideals in } P - L(x). \tag{6.9}$$

Given $\delta > 0$, choose $\varepsilon > 0$ so that

$$\delta_0 - \delta < \left(\frac{1-\varepsilon}{1+\varepsilon}\right)^2 \delta_0 \quad \text{and} \quad \left(\frac{1+\varepsilon}{1-\varepsilon}\right)^2 (1-\delta_0) < 1 - \delta_0 + \delta.$$

Compute N and $N(x)$ for every $x \in P$ approximately using algorithm B_ε to obtain the values $\tilde{N}$, $\tilde{N}(x)$. Let the output of B_ε be the first element x with

$$\frac{1-\varepsilon}{1+\varepsilon}\,\delta_0 \leqq \frac{\tilde{N}(x)}{\tilde{N}} \leqq \frac{1+\varepsilon}{1-\varepsilon}\,(1-\delta_0). \tag{6.10}$$

Such an element exists as if x has $\delta_0 \leqq N(x)/N \leqq (1-\delta_0)$ (the existence of such an element is guaranteed by the theorem of Linial and Saks); then

$$\frac{1-\varepsilon}{1+\varepsilon} < \frac{\tilde{N}(x)/N}{N(x)/N} = \frac{\tilde{N}(x)}{N(x)} \cdot \frac{N}{\tilde{N}} < \frac{1+\varepsilon}{1-\varepsilon} \tag{6.11}$$

holds. For any such element

$$\left(\frac{1-\varepsilon}{1+\varepsilon}\right)^2 \delta_0 < \frac{N(x)}{N} < \left(\frac{1+\varepsilon}{1-\varepsilon}\right)^2 (1-\delta_0). \tag{6.12}$$

□

7. Some remarks and open problems. The problem of computing the number of ideals in ordered sets is shown to be $\#P$-complete in the sense of Valiant [8] in a recent paper of Provan and Ball [4], even if the ordered sets are restricted to be of height 1. They also show that for some counting problems even approximation is $\#P$-complete. It is an open problem whether approximating the number of ideals within a constant factor is $\#P$-complete as well.

In § 4 we have described how an element $x \in P$ satisfying $1/4 \leqq N(x)/N \leqq 3/4$ can efficiently be found if P is a series-parallel ordered set or an interval order. Series-parallel ordered sets form a proper subclass of the class of N-free ordered sets, i.e. ordered sets containing no N in their Hasse diagram. Furthermore, single element

decomposition reveals structural similarities between interval orders and N-free ordered sets. It would therefore be interesting to know whether the results of § 4 also extend to the class of N-free ordered sets.

Also note that the updating formulas of Lemma 1, in particular, allow the determination of N and $N(x)$ for rooted trees. We do not know whether efficient updating formulas exist for general trees. It is also not known, even in the case of rooted forests, whether there exists an optimal (or asymptotically optimal) algorithm which determines the next step efficiently.

REFERENCES

[1] N. LINIAL AND M. E. SAKS, *Searching ordered structures*, J. Algorithms, 6 (1985), pp. 86–103.

[2] ———, *Information bounds are good for search problems on ordered data structures*, Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 473–475.

[3] C. H. PAPADIMITRIOU AND M. YANNAKAKIS, *Scheduling interval-ordered tasks*, this Journal, 8 (1979), pp. 405–409.

[4] J. S. PROVAN AND M. O. BALL, *The complexity of counting cuts and of computing the probability that a graph is connected*, this Journal, 12 (1983), pp. 777–788.

[5] B. SANDS, *Counting antichains in finite partially ordered sets*, Disc. Math., 35 (1981), pp. 213–228.

[6] G. STEINER, *Single machine scheduling with precedence constraints of dimension* 2, Math. Oper. Res., 9 (1984), pp. 248–259.

[7] J. VALDES, R. E. TARJAN AND E. L. LAWLER, *The recognition of series-parallel digraphs*, this Journal, 11 (1981), pp. 298–313.

[8] L. G. VALIANT, *The complexity of enumeration and reliability problems*, this Journal, 8 (1979), pp. 410–421.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
016

PROCESSOR-SHARED TIME-SHARING MODELS IN HEAVY TRAFFIC*

DONALD P. GAVER† AND PATRICIA A. JACOBS†

Abstract. Probability models are presented for computer systems with processor-shared (time-sliced) service discipline. The response (sojourn) time of an arriving job that requires T units of processing time is shown to be approximately Gaussian/normal under moderately heavy traffic conditions, e.g. when the number of terminals is large.

Key words. integral of Markov process, time transformation of Markov processes, heavy traffic approximation, diffusion process, central limit theorem, response time, Markovian computer system model

1. Introduction. Processor sharing (PS) is a mathematically tractable approximation to time sharing, a procedure followed in many actual computer systems. In effect, PS assigns to each job of the i, $(i = 1, 2, \cdots)$ present for processing $1/i$th of the total processing effort; equivalently, a single job with Markovian service rate μ completes processing in $(t, t+dt)$ with probability $(\mu/i)dt + o(dt)$. One advantage of PS is that short jobs are not trapped behind long jobs, as is possible in a FC-FS discipline.

Various mathematical results have been obtained about certain processor sharing models. An early example was the paper of Coffman et al. (1970). Recently extensive results have been obtained for Markovian systems by D. Mitra (1981), and for non-Markovian single-server Poisson arrival systems by T. Ott (1984) and V. Ramaswami (1984).

This paper is a continuation of work reported in Gaver, Jacobs, and Latouche (1984), henceforth GJL, where emphasis was placed on proposing and evaluating simple approximations to the distribution of delay experienced by a particular "tagged" job approaching a time-shared processor. In that paper it was shown that under heavy traffic conditions, *or* if the tagged job duration became large, then the distribution of tagged job *response time* (also called *sojourn time* by others) approaches the normal or Gaussian distribution.

We first show that the analysis of our system and the problem solution can naturally and conveniently be conducted in *work time* rather than in ordinary clock time. Subsequently we focus upon heavy-traffic approximations to the distribution of response time, $R(T)$, given the actual work (computer time) requirement, T. Specific models proposed and examined are (i) for a system of many identical terminals that independently submit jobs or programs according to the same Markovian process, and (ii) for a system having two terminal types, each of which submits jobs in a manner governed by its own Markov process. The methodology extends to general k terminal types as well, and to other models.

The approximation solutions are evaluated for accuracy by means of Monte Carlo simulations.

2. The work-time concept. Imagine that a tagged job requiring T units of processing approaches the computer. Assume that it arrives when the system is in steady state. After that initial moment it undergoes processing, at various rates governed by the

* Received by the editors January 30, 1985, and in revised form December 3, 1985. This research was partially supported by the National Science Foundation under grant ECS 82 16852 and by the Probability and Statistics Program of the U.S. Office of Naval Research.

† Operations Research Department, Naval Postgraduate School, Monterey, California 93943-5100.

amount of its accompaniment, until T units of service or work are accumulated, at which point it departs after a random delay of $R(T)$. In following the tagged job's delay, it turns out to be convenient to *measure time in terms of the amount of actual work or processing that has been accomplished on the tagged job.* Thus let $\{X(w), w \geqq 0\}$ denote the number of programs or jobs undergoing service at a moment when exactly w units of processing have been accomplished on the tagged job. The instantaneous rate of accrual of clock or response time at work time w is clearly $X(w)$: if $X(w) = 1$ then the tagged job is alone and response (clock) time and work time advance at the same rate, while if $X(w) = 17$ the tagged job is accompanied by 16 others and 17 units of response time accrue for every single work time unit. It follows that

$$R(T) = \int_0^T X(w)\, dw. \tag{2.1}$$

For the models to be considered, the process $X(w)$ is a birth-and-death, or simple Markov, process related to $N(t)$, the number of jobs in the system at clock time t; its transition rates, after adjusting for tagged job entry, are seen to be

$$\begin{aligned} \lambda_i^t\, dt &= \lambda(N-i)\, dt = \lambda(N-i) i\, dw = \lambda_i^w\, dw, \\ \mu_i^t\, dt &= \mu\, dt = \mu i\, dw = \mu_i^w\, dw, \end{aligned} \tag{2.2}$$

for $1 \leqq i \leqq N$ where N is the total number of terminals in the system. The term $i\, dw$ is required to allow the work-time process to advance appropriately in clock time. Henceforth we drop the superscripts, allowing the context to imply the appropriate rate.

The approach taken here invokes the work time concept described above to facilitate calculations, first for a single terminal or job type situation, but later on for a system with more diversified traffic patterns.

3. Heavy traffic analysis of a single terminal type Markovian system. Suppose that N terminals have access to a single computer. Each terminal has Markovian demand rate λ, and expected service time μ^{-1}. Service times are assumed to be independently and exponentially distributed. The discipline at the computer is PS. It is clear that if $N(t)$ is the number of terminals that have submitted jobs that are undergoing service at the computer at (clock) time t, then $\{N(t), t \geqq 0\}$ is a Markov process in continuous time that is identical to the classical single repairman problem; see Feller (1957, p. 416). This is so, since if $N(t) = i > 0$, then each individual job or program receives (dt/i) units of processing time in $(t, t+dt)$, and hence departs with probability $\mu(dt/i) + o(dt)$, but the probability that *some* job departs is $i\mu(dt/i) + o(dt) = \mu\, dt + o(dt)$. It has been shown by Iglehart (1965) and by Burman (1979) that under heavy traffic conditions ($N \to \infty$) one may approximate $N(t)$ by a suitable Gaussian process, namely the Ornstein-Uhlenbeck process. This fact alone enables one to study the distribution of $R(T)$, and to deduce approximate normality; see GJL for a first analysis.

3.1. Diffusion approximation in work time. Here is a diffusion process approximation for $X(w)$. On the basis of intuition write down the stochastic differential equation for $\tilde{X}(w)$, the approximationn to $X(w)$:

$$\begin{aligned} d\tilde{X}(w) = {} & \lambda[N - \tilde{X}(w)]\tilde{X}(w)\, dw - \mu\tilde{X}(w)\, dw \\ & + \sqrt{\lambda[N - \tilde{X}(w)]\tilde{X}(w) + \mu\tilde{X}(w)}\, dB(w), \end{aligned} \tag{3.1}$$

where $\{B(w), w \geqq 0\}$ is standard Brownian motion. The first right-hand side term represents infinitesimal drift of $\tilde{X}(w)$, while the second is the diffusion or infinitesimal variance term, the form of which is obtained from the observation that arrival and

departure processes compete like independent Poisson processes in short time periods. Now suppose that, as $N \to \infty$,

$$\tilde{X}(w) \simeq Nm(w) + \sqrt{N}\, Z(w) \tag{3.2}$$

where $m(w)$ is a deterministic function of time, and $\{Z(w), w \geqq 0\}$ is a stochastic process, the properties of which must be discovered. Substitute (3.2) into (3.1) to obtain

(3.3)

$$\begin{aligned} &N\, dm(w) + \sqrt{N}\, dZ(w) \\ &= \lambda[N - Nm(w) - \sqrt{N}\, Z(w)][Nm(w) + \sqrt{N}\, Z(w)]\, dw - \mu[Nm(w) + \sqrt{N}\, Z(w)]\, dw \\ &\quad + \sqrt{\lambda[N - Nm(w) - \sqrt{N}\, Z(w)][Nm(w) + \sqrt{N}\, Z(w)] + \mu[Nm(w) + \sqrt{N}\, Z(w)]} \\ &\qquad \cdot dB(w). \end{aligned}$$

Next isolate terms of order N and $\sqrt{N}$; the result is, after stipulating that $\lambda' = \lambda N$, a constant as $N \to \infty$,

$$O(N): \frac{dm(w)}{dw} = \lambda'[1 - m(w)]m(w) - \mu m(w), \tag{3.4}$$

$$\begin{aligned} O(\sqrt{N}): dZ(w) &= \{\lambda'[1 - 2m(w)] - \mu\} Z(w)\, dw \\ &\quad + \sqrt{\lambda'[1 - m(w)]m(w) + \mu m(w)}\, dB(w); \end{aligned} \tag{3.5}$$

the stochastic differential equation (3.5) is of Ornstein-Uhlenbeck (O-U) form; see Arnold (1974).

Next obtain the approximate long-run mean as the solution of (3.4) with $dm/dw = 0$, examining only the heavy-traffic situation in which $\lambda' > \mu$:

$$\begin{aligned} m(\infty) &= 1 - \frac{\mu}{\lambda'}, \qquad \lambda' > \mu \\ &\equiv 1 - \frac{\mu}{\lambda N}. \end{aligned} \tag{3.6}$$

If the above solution is used to define the stochastic differential equation parameters, there results

$$dZ(w) = -(\lambda' + \mu) Z(w)\, dw + \sqrt{2\mu(1 - (\mu/\lambda'))}\, dB(w), \tag{3.7}$$

which suggests that $\{Z(w)\}$ can be considered an O-U process with constant coefficients, namely

$$dZ(w) = -\rho Z(w)\, dw + \sigma\, dB(w), \tag{3.8}$$

the solution to which is

$$Z(w) = Z(0)\, e^{-\rho w} + \sigma \int_0^w e^{-\rho u}\, dB(u). \tag{3.9}$$

The parameters $\rho = (\lambda' + \mu)$ and $\sigma^2 = 2\mu(1 - \mu/\lambda')$.

3.2. Response time evaluation. Let

$$\tilde{R}(T) = \int_0^T \tilde{X}(w)\, dw \simeq \int_0^T [Nm(w) + \sqrt{N}\, Z(w)]\, dw \tag{3.10}$$

approximate the response time; in this approximation $\tilde{R}(T)$ is normally distributed

(Gaussian). First,

$$E[R(T)] \simeq E[\tilde{R}(T)] \simeq N \int_0^T m(w)\, dw = N\left(1-\frac{\mu}{\lambda'}\right) T. \tag{3.11}$$

Second,

$$\begin{aligned} \operatorname{Var}[R(T)] &\simeq \operatorname{Var}[\tilde{R}(T)] = \operatorname{Var}\left[\sqrt{N}\int_0^T Z(w)\, dw\right] \\ &= N\left\{E\left[\int_0^T Z(w)\, dw \int_0^T Z(u)\, du\right] - \left(E\left[\int_0^T Z(w)\right]\right)^2\right\}; \end{aligned} \tag{3.12}$$

for ease of writing we have left the initial condition $Z(0)$ implicit. In order to evaluate the above, recall that the tagged job approaches the server when the latter is in equilibrium, i.e., at $t=\infty$. It may be shown that the diffusion approximation for $N(t)$, the number undergoing service at clock time t, is

$$\tilde{N}(t) = Na(t) + \sqrt{N}\, Y(t) \tag{3.13}$$

where $a(t)$ is a deterministic function of time and $\{Y(t)\}$ is a particular Ornstein-Uhlenbeck process. A similar analysis to that leading to (3.6) and (3.7) yields

$$\begin{aligned} a(\infty) &= 1-\frac{\mu}{N\lambda}, \qquad N\lambda > \mu, \\ &= 1-\frac{\mu}{\lambda'} \end{aligned} \tag{3.14}$$

and

$$E[Y(\infty)] = 0, \qquad \operatorname{Var}[Y(\infty)] = \frac{\mu}{N\lambda} \equiv \frac{\mu}{\lambda'}; \tag{3.15}$$

see GJL for a derivation; (3.13) provides the initial condition for evaluating moments of $R(T)$, using (3.10). Identify $Z(0)$, the initial value of the work time noise process $Z(w)$, with $Y(\infty)$. According to (3.9), this implies that $E[\int_0^T Z(w)\, dw] = 0$. In order to compute the $\operatorname{Var}[R(T)]$ it is next necessary to evaluate the following integral:

$$\begin{aligned} I(T) &= E\left[\int_0^T Z(w)\, dw \int_0^T Z(u)\, du\right] \\ &= E\left[2\int_0^T Z(w)\, dw \int_w^T E[Z(u)|Z(w)]\, du\right] \\ &= 2E\left[\int_0^T Z(w)\, dw \int_w^T Z(w)\, e^{-\rho(u-w)}\, du\right] \\ &= 2E\left[\int_0^T Z(w)\, dw\, Z(w)\frac{1}{\rho}(1-e^{-\rho(T-w)})\right] \\ &= 2E\left[\int_0^T E[Z(w)^2|Z(0)]\frac{1}{\rho}(1-e^{\rho(T-w)})\, dw\right]. \end{aligned} \tag{3.16}$$

Square (3.9) and take the expectation to see that, conditionally on $Z(0)$,

$$\begin{aligned} I(T) &= 2E\left[\int_0^T (Z(0)^2\, e^{-2\rho w} + \frac{\sigma^2}{2\rho}(1-e^{-2\rho w}))\frac{1}{\rho}(1-e^{\rho(T-w)})\, dw\right] \\ &= \frac{2}{\rho}E[Z(0)^2]\left[\int_0^T e^{-2\rho w}(1-e^{\rho(T-w)})\, dw\right] + \frac{\sigma^2}{\rho^2}\int_0^T (1-e^{-2\rho w})(1-e^{-\rho(T-w)})\, dw. \end{aligned}$$

Now put $E[Z(0)^2] = E[Y(\infty)^2] = \mu/\lambda' = \sigma^2/2\rho$ to see that

$$I(T) = \frac{\sigma^2}{\rho^2}\int_0^T [1 - e^{-\rho(T-w)}]\, dw = \frac{\sigma^2}{\rho^2}\left[T - \frac{1}{\rho}(1 - e^{-\rho T})\right] = \frac{\sigma^2 T}{\rho^2}\left[1 - \frac{1}{\rho T}(1 - e^{-\rho T})\right].$$

Thus it follows that

$$\text{Var}[R(T)] \simeq \text{Var}[\tilde{R}(t)] = NT\frac{\sigma^2}{\rho^2}\left[1 - \frac{1}{\rho T}(1 - e^{-\rho T})\right]. \tag{3.17}$$

To terms of order T this agrees with (4.10) of GJL; not surprisingly, the additional factor in (3.14) can actually provide numerical results superior to those of GJL.

The form of the heavy traffic approximation, namely the limiting normal form with parameters (3.11) and (3.14), can be more rigorously validated by use of the theory of convergence of suitably normalized sequences of semigroups of transformations; see Burman (1979). Details appear in the Appendix to this paper.

4. Heavy traffic analysis of a K-terminal-type processor sharing system. Consider the following natural extension of the previous model. The processor is jointly utilized by K sets of terminals, each generating distinctive job types. There are N_i terminals in the ith set, and arrival rate and service rate are λ_i and μ_i respectively. Again the discipline at the computer is PS. Of course this is *not* the same as a situation in which all terminals are the same, but Type j jobs occur with probability p_j from each terminal. The latter model can, however, be studied in an analogous heavy-traffic manner, as can other interesting models.

4.1. A diffusion model for the work-time process. Let $\{X_i(w), i = 1, \cdots, K\}$ represent the number of jobs of all types present at the computer at work time w. The present model implies that $\{X_i(w)\}$ is a multivariate or vector-state birth and death Markov process. We choose to study a diffusion approximation $\{\tilde{X}_i(w)\}$ to $\{X_i(w)\}$ that is described by the following system of s.d.e.:

$$\begin{aligned} d\tilde{X}_i(w) &= \lambda_i(N_i - \tilde{X}_i(w))\left(\sum_{k=1}^{K} \tilde{X}_k(w)\right) dw - \mu_i \tilde{X}_i(w)\, dw \\ &\quad + \sqrt{\lambda_i(N_i - \tilde{X}_i(w)) \sum_{k=1}^{K} \tilde{X}_k(w) + \mu_i \tilde{X}_i(w)}\, dB_i(w), \qquad i = 1, 2, \cdots, K \end{aligned} \tag{4.1}$$

where $\{B_i(w)\}$ are mutually independent standard Brownian motion or Wiener processes. The work-time process is a transformation of the clock-time process; in particular, the drift of the ith component of the clock-time process $\{N_i(t)\}$ is seen to be

$$\lambda_i(N_i - N_i(t))\, dt - \mu_i \frac{N_i(t)\, dt}{\sum_{k=1}^{K} N_k(t)}, \tag{4.2}$$

which exhibits the processor-sharing effect in the term multiplying μ_i. Multiplication by the total in service, $\sum N_i(t)$, converts to the work-time transition rates, in analogy with (2.2).

Now once again approximate by writing

$$\tilde{X}_i(w) = N_i m_i(w) + \sqrt{N_i}\, Z_i(w), \qquad i = 1, 2, \cdots, K; \tag{4.3}$$

$m_i(w)$ and $\{Z_i(w)\}$ are to be determined, subject to the normalization $N = \sum_{k=1}^{K} N_k \to \infty$ but with

$$N_i/N \to l_i, \qquad 0 \leqq l_i \leqq 1, \tag{4.4a}$$

and

$$N\lambda_i \to \lambda_i', \quad 0 \leqq \lambda_i' \leqq \infty, \quad \lambda_i' > \mu_i. \tag{4.4b}$$

Conditions (4.3) and (4.4) are referred to as the *heavy traffic normalization* (HTN). The result of isolating terms according to order in (4.1) is:

$$O(N): \frac{dm_i(w)}{dw} = \lambda_i'(1-m_i(w)) \sum_{k=1}^{K} l_k m_k(w) - \mu_i m_i(w), \tag{4.5}$$

$$\begin{aligned} O(\sqrt{N}): dZ_i(w) = \lambda_i' \Bigg\{ \Bigg(\sum_{k=1}^{K} l_k m_k(w) \Bigg) Z_i(w) + \sqrt{l_i}(1-m_i(w)) \sum_{k=1}^{K} \sqrt{l_k}\, Z_k(w) \Bigg\} dw \\ - \mu_i Z_i(w)\, dw + \sqrt{\lambda_i'(1-m_i(w)) \sum_{k=1}^{K} l_k m_k(w) + \mu_i m_i(w)}\, dB_i(w) \end{aligned} \tag{4.6}$$

for $i = 1, 2, \cdots, K$. Thus $(m_i(w);\ w \geqq 0;\ i = 1, 2, \cdots, K)$ must be found by solving a system of ordinary first-order, but nonlinear differential equations, while (4.6) shows that $\{Z_i(w);\ w \geqq 0,\ i = 1, 2, \cdots, K\}$ is a multivariate Ornstein–Uhlenbeck process.

4.2. A diffusion model for the clock-time process. In order to provide the initial conditions encountered by the tagged job, it is necessary to study the clock-time process $N_i(t)$; see (3.13). The corresponding approximation has s.d.e.

$$\begin{aligned} d\tilde{N}_i(t) = \lambda_i(N_i - \tilde{N}_i(t))\, dt - \mu_i \frac{\tilde{N}_i(t)}{\sum_{k=1}^{K} \tilde{N}_k(t)}\, dt \\ + \sqrt{\lambda_i(N_i - \tilde{N}_i(t)) + \mu_i \Bigg(\tilde{N}_i(t) \Big/ \sum_{k=1}^{K} \tilde{N}_k(t) \Bigg)}\, dB_i(t), \qquad i = 1, 2, \cdots, K. \end{aligned} \tag{4.7}$$

Now invoke the HTN:

$$\tilde{N}_i(t) = N_i a_i(t) + \sqrt{N_i}\, Y_i(t), \tag{4.8}$$

and again $N = \sum_{k=1}^{K} N_k \to \infty$, with

$$N_i / N \to l_i, \qquad 0 \leqq l_i \leqq 1, \tag{4.9}$$

but

$$\mu_i / N \to \mu_i', \quad 0 \leqq \mu_i' < \infty, \quad \lambda_i > \mu_i'. \tag{4.10}$$

The result of isolating terms is

$$O(N): \frac{da_i(t)}{dt} = \lambda_i(1-a_i(t)) - \mu_i' \frac{a_i(t)}{\sum_{k=1}^{K} l_k a_k(t)}, \tag{4.11}$$

$$\begin{aligned} O(\sqrt{N}): dY_i(t) = -\lambda_i Y_i(t)\, dt - \mu_i' \Bigg\{ \frac{Y_i(t)}{\sum_{k=1}^{K} l_k a_k(t)} + \sqrt{l_i}\, a_i(t) \frac{\sum_{k=1}^{K} \sqrt{l_k}\, Y_k(t)}{\left(\sum_{k=1}^{K} l_k a_k(t)\right)^2} \Bigg\} dt \\ + \sqrt{\lambda_i(1-a_i(t)) + \mu_i'(a_i(t) / \textstyle\sum_{k=1}^{K} l_k a_k(t))}\, dB_i(t). \end{aligned} \tag{4.12}$$

These equations closely resemble those describing the work-time approximation; again the semigroup approach is applicable.

If a long-run solution to the $O(N)$ term exists in work time, and consequently $dm_i/dw \to 0$ as $w \to \infty$, the result is the system of equations for $m_i(\infty) \equiv m_i$:

$$\rho_i(1-m_i) \sum_{k=1}^{K} l_k m_k - m_i = 0, \tag{4.13}$$

where $\rho_i = \lambda_i'/\mu_i = N\lambda_i/\mu_i$. Now these same equations are satisfied by a presumed long-run solution in clock time, i.e., if $da_i/dt \to 0$ in (4.11); for $a_i(\infty) = a_i$:

$$\rho_i(1-a_i) - \frac{a_i}{\sum_{k=1}^{K} l_k a_k} = 0. \tag{4.14}$$

Consequently the long-run solutions in work and clock time agree at the $O(N)$ term level; this means that the long-run mean number present in both clock and work time agree:

$$E[N_i(t)] \sim Na_i(\infty) = Nm_i(\infty) \sim E[X_i(t)]. \tag{4.15}$$

Next substitute these long-run results in the s.d.e. to see that as $t, w \to \infty$, Y_i and Z_i are essentially the same process. Put $S = \sum_{k=1}^{K} l_k a_k$ to simplify writing. Then

$$\begin{aligned} dY_i(t) = {} & -\lambda_i Y_i(t)\, dt - \mu_i' \frac{Y_i(t)}{S}\, dt + \lambda_i(1-a_i)\sqrt{l_i}\, \frac{\sum_{k=1}^{K} \sqrt{l_k}\, Y_k(t)}{S}\, dt \\ & + \sqrt{2\lambda_i(1-a_i)}\, dB_i, \end{aligned} \tag{4.16}$$

or

$$\begin{aligned} dY_i(t) = {} & \lambda_i \left\{ -Y_i S + \sqrt{l_i}\,(1-a_i) \sum_{k=1}^{K} \sqrt{l_k}\, Y_k(t) \right\} \frac{dt}{S} - \mu_i' Y_i(t) \frac{dt}{S} \\ & + \sqrt{2\lambda_i(1-a_i)S}\, \frac{1}{\sqrt{S}}\, dB_i(t), \end{aligned} \tag{4.17}$$

and a direct comparison with the corresponding equation in work time, (4.6), shows that *the long-run behaviors of the two processes* $\{Z_i(w)\}$ and $\{Y_i(t)\}$ *are identical except for a constant time-scale change*: for large w and t,

$$\{Z_i(w)\} \quad \text{and} \quad \left\{ Y_i\left(\frac{t}{NS}\right) \right\} \tag{4.18}$$

have the same probability law; i.e., finite-dimnsional distributions and limiting distribution.

4.3. Response time. We discuss the response time under these conditions: a tagged job approaches the processor when the latter has been operating for some time, so the long-run clock-time distribution prevails; after arrival, the job remains present until the total work time accumulated on the job is T, the requested service time, giving

$$\begin{aligned} R(T) &= \int_0^T \sum_{i=1}^{K} [X_i(w)\, dw] \simeq \sum_{i=1}^{K} \int_0^T \tilde{X}_i(w)\, dw \\ &= N \sum_{i=1}^{K} l_i \int_0^T m_i(w)\, dw + \sqrt{N} \sum_{i=1}^{K} \sqrt{l_i} \int_0^T Z_i(w)\, dw \end{aligned} \tag{4.19}$$

where it is understood that the initial condition for the $Z_i(w)$ integrand in (4.19) is given by the approximate stationary distribution from the clock-time process. In view of (4.18), this is equivalent to removing the initial condition by the long-run distribution of the work-time process itself.

Since the long-run situation is being discussed it is first necessary to solve the steady-state version of (4.5)

$$0 = \lambda_i'(1-m_i)\sum_{k=1}^{K} l_k m_k - \mu_i m_i, \qquad i = 1, 2, \cdots, K. \tag{4.20}$$

Then the solution provides parameters for the long-run version of (4.6), here written in matrix form

$$d\underline{Z}(w) = \underline{A}\,\underline{Z}(w)\,dw + \underline{\sigma}\,d\underline{B}. \tag{4.21}$$

Now to find the variance of $R(T)$, append the row

$$dZ_{k+1}(w) = \sum_{i=1}^{K} \sqrt{l_i}\, Z_i(w)\,dw \tag{4.22}$$

to the former drift matrix $\underline{A}$ of (4.21), and consider the system

$$d\underline{Z}^*(w) = \underline{A}^*\underline{Z}^*(w)\,dw + \underline{\sigma}^*\,dB^* \tag{4.23}$$

the solution to which can be formally written out in terms of the appropriate fundamental matrix, and computed in terms of eigenvalues and eigenvectors of the matrix $\underline{A}^*$. See e.g. Arnold (1974, Chap. 8) and Coddington and Levinson (1955) for details. A convenient way of formalizing the calculations is actually by using Laplace transforms. Unfortunately, no truly simple formulas result. Finally, the covariance matrix $\underline{C}(w)$ of the components of $\underline{Z}^*(w)$ satisfies the matrix differential equation

$$\frac{d\underline{C}(w)}{dw} = (\underline{A}^*)\underline{C}(w) + \underline{C}(w)(\underline{A}^*)' + (\underline{\sigma})(\underline{\sigma})'$$

where $'$ denotes transpose; the initial conditions are provided by the long-run distribution in clock time, or in view of (4.18), of the work-time process $\{\underline{Z}\}$ itself. It is the $(K+1)$st diagonal element of $\underline{C}(w)$, evaluated at $w = T$ and multiplied by N that provides the required approximate Var$[R(T)]$.

5. Simulation studies of the accuracy of the normal approximations to the distribution of response time. In this section we use simulation to study the numerical accuracy of normal approximations to the distribution of the response time. Two continuous time Markov chain models were simulated. In one there is a single terminal type; in the second, a two-terminal type system is examined. Two normal approximations were evaluated: one results from a central limit theorem, and the other results from applying the previously derived diffusion approximation to the Markov processes.

5.1. A single terminal type Markovian system. Let $\bar{X}(w)$ denote the number of other jobs undergoing service at a moment when exactly w units of processing has been accomplished on the tagged job for the single terminal type Markovian model of § 3. Since $\{\bar{X}(w);\ w \geqq 0\}$ is a Markov process and

$$R(T) = \int_0^T (\bar{X}(w)+1)\,dw,$$

it follows that there are constants $m(c)$ and $\sigma(c)$ such that

$$\frac{R(T) - m(c)T}{\sigma(c)\sqrt{T}}$$

converges in distribution to a standard normal distribution as $T \to \infty$ (cf. Keilson (1979, p. 121)). Call this a central limit theorem (CLT) for such a process. In this case

$$m(c) = 1 + \sum \pi(j) j$$

where π is the stationary distribution of $\{\bar{X}(w);\ w \geqq 0\}$ and a formula for evaluating $\sigma(c)$ is given in Keilson (1979). The CLT normal approximation states that $R(T)$ has a normal distribution with mean $m(c)T$ and variance $\sigma(c)^2 T$. The CLT mean is the true mean for $R(T)$ under steady state (cf. GJL). The CLT normal approximation should be increasingly accurate as T becomes large, despite values of other system parameters, including the number of terminals.

The derivation of the heavy traffic (or diffusion) approximation is detailed in § 3. In summary, the HT approximation is that $R(T)$ has a normal distribution with mean

$$N\left(1 - \frac{\mu}{\lambda N}\right) T, \qquad \frac{\mu}{\lambda N} < 1$$

and variance

$$NT\frac{\sigma^2}{\rho^2}\left[1 - \frac{1}{\rho T}(1 - e^{-\rho T})\right],$$

where

$$\sigma^2 = 2\mu\left(1 - \frac{\mu}{N\lambda}\right)$$

and

$$\rho = N\lambda + \mu.$$

The mean and variance of the HT approximation are easier to compute than those for the CLT. It is anticipated that the HT approximation should be increasingly accurate as N becomes large when heavy traffic conditions prevail, i.e. $\mu/\lambda N < 1$. It is inapplicable under other circumstances. We have conducted simulations to assess these anticipations. We report here only some of the results. More extensive simulation results are reported in Gaver and Jacobs (1985) and Pornsuriya (1984). All simulations were carried out on an IBM 3033 computer at the Naval Postgraduate School using the LLRANDOMII random number generating package (see Lewis and Uribe (1981)).

Conditional response times given the number of jobs being processed at the time of arrival of the tagged job were simulated; the tagged job required T time units of processing. For each initial condition, 500 replications were done. Sample moments and relative frequencies were computed for each initial condition giving conditional response time sample moments, and selected response time relative frequencies, i.e., estimated probabilities of response times in selected ranges. Unconditional sample moments and relative frequencies were then computed by multiplying each conditional moment or relative frequency by the appropriate stationary probability of there being j jobs present at the time of arrival of the tagged job and then summing over all possible j. The stationary probability is of the form $k\lambda\pi(j)$ where k is chosen so that the probabilities sum to 1 (cf. Kelly (1979)). A detailed description of the simulation program can be found in Pornsuriya (1984).

Simulated and approximating means and standard deviations for $N = 10$, $\lambda = 25$ and $\mu = 100$ appear in Table 1. When $N = 10$, $\lambda = 25$ and $\mu = 100$, it follows from (3.6) that approximately $(10)(1 - (100/250)) = 6$ jobs are being processed along with the tagged job; thus the traffic is moderate in this case. The HT mean is lower than the

TABLE 1
Simulated mean and standard deviation for $R(T)$ and their approximating values.

$N=10, \lambda=25, \mu=100$			
TIME T		Mean	Std. dev.
0.01	Simulation	.0606	.0503
		(.0002)*	
	CLT	.0605	.0245
	HT	.0600	.0160
0.025	Simulation	.1507	.0315
		(.0004)	
	CLT	.1513	.0288
	HT	.1500	.0314
0.05	Simulation	.3021	.0497
		(.0008)	
	CLT	.3027	.0548
	HT	.3000	.0481
0.10	Simulation	.6036	.0748
		(.0012)	
	CLT	.6053	.0776
	HT	.6000	.0706

* Standard error for the estimate of the mean.

simulated mean. As mentioned before, the CLT mean equals the true mean. The CLT standard deviation approaches the simulation value as T becomes large, as anticipated. In order to assess the degree of normality of the distribution of $R(T)$, the α-quantiles for each approximating normal distribution were computed. The relative frequency of being less than or equal to each α-quantile was computed using the simulated data. The results appear in Table 2. The HT approximation does better than the CLT. However, as expected, the CLT improves for larger values of T. Note, however, that all simulations have been carried out for the modest system size, $N=10$. If N grows to say 50, or 100, the HT approximations can be expected to improve correspondingly; they are often not bad even at the level of $N=10$.

5.2. Simulation results for Markovian model with two-terminal types. In this subsection we describe the results of a simulation of the general K-type Markovian model of § 4, in the case of $K=2$ sets of terminals. As before let $\bar{X}_i(w)$ represent the number

TABLE 2
Simulated probability (relative frequency) that the response time is less than or equal to the approximating α-quantile.

$N=10, \lambda=25, \mu=100$								
TIME	α:	.10	.25	.50	.75	.90	.95	.99
0.01	CLT	.04	.15	.45	.85	1.0	1.0	1.0
	HT	.11	.22	.43	.72	.91	.98	1.0
0.025	CLT	.08	.19	.45	.80	.98	1.0	1.0
	HT	.10	.22	.44	.73	.92	.98	1.0
0.05	CLT	.09	.22	.45	.77	.95	.99	1.0
	HT	.11	.22	.44	.71	.91	.97	1.0
0.10	CLT	.10	.24	.47	.76	.93	.98	1.0
	HT	.11	.23	.45	.70	.89	.95	1.0

of other jobs of type i being processed when the tagged job has acquired exactly w units of processing. As before the response time for the tagged job requiring T units of work is

$$R(T) = \int_0^T [\bar{X}_1(w) + \bar{X}_2(w) + 1]\, dw.$$

The process $\{(\bar{X}_1(w), \bar{X}_2(w)); w \geqq 0\}$ is Markovian. Hence again $R(T)$ satisfies a central limit theorem as $T \to \infty$. The normal approximation for the distribution of $R(T)$ resulting from the central limit theorem will again be referred to as CLT.

The mean term $(m_1 + m_2)T$ for the heavy traffic approximation was computed by solving the system of equations (4.20) for m_1 and m_2. The variance term for the approximation was computed by solving the system of stochastic differential equations (4.23) as detailed in Arnold (1974, Cor. (8.2.4)). The fundamental matrix (Arnold (1974, p. 129)) was found by computing Laplace transforms of the system of defining differential equations and then inverting the solution. The approximating variance term was found by computing the variance of the solution of the s.d.e. As in the case of one-terminal type, it is a linear combination of exponentials and constant terms. Its exact form is uninformative and will not be given here.

Conditional response times, given the number of jobs of each type being processed at the time of arrival of the tagged job, were simulated. The tagged job was always taken to be a Type 1 job. For each initial condition, 300 applications were carried out. Sample moments and probabilities (relative frequencies) were computed for each initial condition giving conditional sample moments and probabilities (relative frequencies). Unconditional sample moments and probabilities were computed in a similar manner to that in the one-terminal type simulation; see Pornsuriya (1984).

TABLE 3

Simulated means and standard deviations for $R(T)$ and their approximating values.

$N_1 = 5$, $N_2 = 5$, $\lambda_1 = 20$, $\lambda_2 = 30$, $\mu_1 = 50$, $\mu_2 = 100$			
TIME T		Mean	Std. dev.
0.01	Simulation	0.0713 (0.0001)*	0.0135
	CLT	0.0707	0.0204
	HT	0.0710	0.0132
0.025	Simulation	0.1783 (0.0003)	0.0263
	CLT	0.1766	0.0322
	HT	0.1775	0.0253
0.0375	Simulation	0.2664 (0.0005)	0.0353
	CLT	0.2650	0.0395
	HT	0.2663	0.0325
0.050	Simulation	0.3570 (0.0005)	0.0414
	CLT	0.3533	0.0456
	HT	0.3550	0.0384
0.0625	Simulation	0.4439 (0.0006)	0.0478
	CLT	0.4416	0.0510
	HT	0.4438	0.0435

* Standard error of the estimate of the mean.

TABLE 4
Simulated means and standard deviations for $R(T)$ and their approximating values.

$N_1=5$, $N_2=5$, $\lambda_1=10$, $\lambda_2=40$, $\mu_1=25$, $\mu_2=125$			
TIME		Mean	Std. dev.
0.01	Simulation	0.0717	0.0134
		(0.0001)*	
	CLT	0.0717	0.0237
	HT	0.0720	0.0133
0.025	Simulation	0.1788	0.0280
		(0.0004)	
	CLT	0.1793	0.0375
	HT	0.1799	0.0271
0.0375	Simulation	0.2690	0.0376
		(0.0005)	
	CLT	0.2684	0.0459
	HT	0.2699	0.0359
0.0500	Simulation	0.3588	0.0456
		(0.0006)	
	CLT	0.3585	0.0530
	HT	0.3599	0.0433
0.0625	Simulation	0.4481	0.0527
		(0.0007)	
	CLT	0.4481	0.0529
	HT	0.4498	0.0496

* Standard error of the estimate of the mean.

Values of the simulated means and standard deviations and their approximating values for $R(T)$ for various cases in which $N_1=5$ and $N_2=5$ appear in Tables 3–4. Once again the CLT mean is the true steady-state mean for $R(T)$. The means and standard deviations of $R(T)$ for each T differ surprisingly little for the two cases. This suggests that perhaps the two-type terminal model can often be satisfactorily approximated by a one-type model in which the arrival rate and service rates are the average arrival and service rates in the two-type model. Values of the simulated means and standard deviations and their approximating values for the approximate one-type model with $N=10$, $\lambda=25$ and $\mu=75$ appear in Table 5. The values for the approximate one-type model are acceptably close to those for the two-type model. Note that the quality of the HT approximation is generally quite good, even though the system sizes N_1 and N_2 can hardly be called "large."

To assess the quality of the normal approximation to the distribution of $R(T)$ for the two-type model, the α-quantiles for each two-type approximating normal distribution were computed. The relative frequency of being less than or equal to each α-quantile was then computed, using the simulated data. The results appear in Tables 6–7. From the heavy traffic approximation to the mean it follows that approximately 7 jobs are being processed with the tagged job. Thus, all the cases considered are really moderate traffic cases. The tables indicate that the HT approximation tends to describe the quantiles better than does the CLT. However, as is expected, the CLT improves with larger T.

Appendix. Heavy traffic approximation by convergence-of-semigroups methodology. The purpose of this appendix is to outline a mathematical framework upon which the heavy traffic approximations of this paper may be rigorously based. The approach is

TABLE 5

Simulated means and standard deviations for $R(T)$ and their approximating values for the one-type model.

$N=10, \lambda=25, \mu=75$ TIME T		Mean	Std. dev.
0.1	Simulation	0.0701 (0.0002)*	0.0139
	CLT	0.0701	0.0206
	HT	0.0700	0.0135
0.025	Simulation	0.1766 (0.0005)	0.0263
	CLT	0.1752	0.0325
	HT	0.1750	0.0258
0.0375	Simulation	0.2620 (0.0008)	0.0361
	CLT	0.2628	0.0398
	HT	0.2625	0.0330
0.0500	Simulation	0.3501 (0.0009)	0.0430
	CLT	0.3504	0.0460
	HT	0.3500	0.0390
0.0625	Simulation	0.4388 (0.0011)	0.0478
	CLT	0.4380	0.0514
	HT	0.4375	0.0441

* Standard error of the estimate of the mean.

TABLE 6

Simulated probability (relative frequency) that the response time is less than or equal to the approximating α-quantiles.

$N_1=5, N_2=5, \lambda_1=20, \lambda_2=30, \mu_1=50, \mu_2=100$ TIME T	α:	0.10	0.25	0.50	0.75	0.90	0.95	0.99
0.010	CLT	0.041	0.137	0.433	0.838	0.998	1.0	1.0
	HT	0.109	0.221	0.443	0.719	0.915	0.977	1.0
0.0250	CLT	0.062	0.170	0.425	0.765	0.969	0.997	1.0
	HT	0.103	0.223	0.438	0.708	0.911	0.975	1.0
0.0375	CLT	0.079	0.191	0.429	0.754	0.952	0.991	1.0
	HT	0.118	0.229	0.447	0.713	0.905	0.969	0.999
0.0500	CLT	0.075	0.181	0.417	0.724	0.939	0.986	1.0
	HT	0.106	0.222	0.431	0.693	0.897	0.965	0.998
0.0625	CLT	0.081	0.198	0.436	0.745	0.932	0.980	1.0
	HT	0.115	0.238	0.451	0.719	0.898	0.960	0.997

to use an analytical theory of convergence of semigroups of operators apparently first applied to queueing problems by Burman (1979) in a regrettably unpublished thesis. See also Lehoczky and Gaver (1981) where the technique is used to obtain results concerning a data-voice traffic sharing multichannel system. The theory of semigroups of operators is introduced in Feller (1971), and detailed in Dynkin (1965); the convergence ideas are discussed in Trotter (1974) and Kato (1976). The basic notion is that the state variable of a process, say the work time process of § 4.1, $\{X_i(w; N), w \geqq 0\}$, is one of a sequence of birth-and-death Markov processes indexed by system size N.

TABLE 7

Simulated probability (relative frequency) that the response time is less than or equal to the approximating α-quantiles.

		$N_1=5,\ N_2=5,\ \lambda_1=10,\ \lambda_2=40,\ \mu_1=25,\ \mu_2=125$						
TIME T	α	0.10	0.25	0.50	0.75	0.90	0.95	0.99
0.010	CLT	0.027	0.119	0.447	0.906	1.0	1.0	1.0
	HT	0.110	0.230	0.458	0.737	0.931	0.982	1.0
0.0250	CLT	0.061	0.178	0.451	0.824	0.991	0.999	1.0
	HT	0.121	0.236	0.464	0.734	0.930	0.985	1.0
0.0375	CLT	0.068	0.191	0.441	0.790	0.979	0.999	1.0
	HT	0.117	0.239	0.451	0.725	0.927	0.983	1.0
0.0500	CLT	0.081	0.189	0.430	0.776	0.975	0.997	1.0
	HT	0.117	0.232	0.444	0.729	0.935	0.985	1.0
0.0625	CLT	0.085	0.199	0.442	0.772	0.960	0.995	1.0
	HT	0.121	0.237	0.456	0.737	0.923	0.979	1.0

Given such a sequence of Markov processes, $\langle\{X_i(w;\ N)\}\rangle$, each with its appropriate state space, S_N, it is desired to show that the corresponding sequence of probability transition functions converges to that of some limiting process that has state space S_∞; in the present case $S_\infty=\mathbb{R}_+^K$. The limiting process under the normalization of $\{X(w;\ N)\}$ chosen will be a particular diffusion process, namely, in the present heavy traffic situation the multivariate Ornstein-Uhlenbeck.

The Trotter-Kato theory of convergence deals with the convergence of infinitesimal operators A_N of the normalized processes $\{X_i(w;\ N)\}$: if $A_Nf\to A_\infty f$ in sup norm for a suitable class of test functions (e.g., $f(z)$: $\mathbb{R}^K\to\mathbb{R}_1$ m-times continuously differentiable, $m\geqq 3$, that vanish identically outside a bounded subset of $\mathbb{R}^K$ and further such that the functions, f, together with their first and second derivatives do not increase faster than some fixed power of z) it can be concluded that the semigroups converge, and hence the Markov probability transition functions themselves converge.

We now proceed with the formal calculation of the limiting generator for our normalized process. Invoke (4.4) so

$$\text{(A1)}\qquad Z_i^N(w)=\frac{X_i^N(w)-N_im_i(w)}{\sqrt{N_i}}=\frac{X_i^N(w)-Nl_im_i(w)}{\sqrt{l_i}\sqrt{N}}.$$

By definition, for $z=(z_1, z_2, \cdots, z_K)$ and f in the above class

$$\text{(A2)}\qquad A_Nf(\underline{z})=\lim_{\Delta\to 0}\{E[f(\underline{Z}^N(w+\Delta))|\underline{Z}^N(w)=\underline{z}]-f(\underline{z})\}\frac{1}{\Delta}.$$

Given $Z_i^N(w)=z_i$, and $C_i(w,\ w+\Delta)$ represents the change in $X_i^N(w)$,

$$\text{(A3)}\qquad \begin{aligned} Z_i^N(w+\Delta)&=\frac{X_i^N(w)+C_i(w,\ w+\Delta)-N_im_i(w+\Delta)}{\sqrt{N_i}}\\ &=\frac{C_i(w)}{\sqrt{N_i}}+z_i+\sqrt{N_i}\,m_i'(w)\Delta+o(\Delta).\end{aligned}$$

Consequently, for $\underline{z}$ such that $z_i > -\sqrt{N_i}\, m_i(w)\Delta - (1/\sqrt{N_i})$,

$$
\begin{aligned}
A_N f(\underline{z}) = \lim_{\Delta\to 0}\Bigg[\sum_{i=1}^{K}\Bigg\{ & f\Bigg(z_1,\cdots,z_i+\frac{1}{\sqrt{l_i}\sqrt{N}}-\sqrt{l_i}\sqrt{N}\, m_i'(w)\Delta,\cdots,z_K\Bigg)\lambda_i(N)\Delta \\
& + f\Bigg(z_1,\cdots,z_i-\frac{1}{\sqrt{l_i}\sqrt{N}}-\sqrt{l_i}\sqrt{N}\, m_i'(w)\Delta,\cdots,z_K\Bigg)\mu_i(N)\Delta\Bigg\} \\
& + f(z_1-\sqrt{l_1}\sqrt{N}\, m_1'(w)\Delta,\cdots,z_K-\sqrt{l_K}\sqrt{N}\, m_K'(w)\Delta) \\
& \times\Bigg(1-\sum_{i=1}^{K}\{(\lambda_i(N)+\mu_i(N))\Delta\}\Bigg) - f(z_1,z_2,\cdots,z_K)+o(\Delta)\Bigg]\frac{1}{\Delta}
\end{aligned}
\tag{A4}
$$

where for simplicity (and generality) we abbreviate

$$
\lambda_i(N) = \lambda_i[N_i - N_i m_i(w) - \sqrt{N_i}\, z_i]\Bigg(\sum_{j=1}^{K}(N_j m_j(w)+\sqrt{N_j}\, z_j)\Bigg) \tag{A5}
$$

$$
\sim(\lambda_i N)\Bigg[l_i(1-m_i(w))-\frac{\sqrt{l_i}}{\sqrt{N}}z_i\Bigg]\Bigg(N\sum_{j=1}^{K}\Bigg(l_j m_j(w)+\frac{\sqrt{l_j}}{\sqrt{N}}z_j\Bigg)\Bigg) \tag{A6}
$$

and

$$
\mu_i(N) = \mu_i(N_i m_i(w)+\sqrt{N}\, z_i) \sim \mu_i N\Bigg(l_i m_i(w)+\frac{\sqrt{l_i}}{\sqrt{N}}z_i\Bigg). \tag{A7}
$$

Upon passage to the limit via Taylor series expansion it is seen that

$$
\begin{aligned}
A_N f(\underline{z}) = \sum_{i=1}^{K}\Bigg\{ & f\Bigg(z_1,\cdots,z_i+\frac{1}{\sqrt{l_i}\sqrt{N}},\cdots,z_K\Bigg)\lambda_i(N) \\
& + f\Bigg(z_1,\cdots,z_i-\frac{1}{\sqrt{l_i}\sqrt{N}},\cdots,z_K\Bigg)\mu_i(N)\Bigg\} \\
& + f(z_1,\cdots,z_i,\cdots,z_K)\Bigg[-\sum_{j=1}^{K}(\lambda_j(N)+\mu_j(N))\Bigg] \\
& - \sum_{j=1}^{K} f'_{z_j}\sqrt{l_j}\sqrt{N}\, m_j'(w).
\end{aligned}
\tag{A8}
$$

Note that no specific normalization has been utilized up to this point. Now, however, invoke the HTN of (4.4) and utilize (A6) and (A7) specifically; allowing N to become large,

$$
\begin{aligned}
A_N f(z) &\sim \sum_{i=1}^{K}\Bigg\{f'_{z_i}\frac{\lambda_i(N)-\mu_i(N)}{\sqrt{l_i}\sqrt{N}}+\frac{1}{2}f''(z_i)\frac{\lambda_i(N)+\mu_i(N)}{l_i N}-f'_{z_i}\sqrt{l_i}\sqrt{N}\, m_i'(w)\Bigg\} \\
&\sim \sum_{i=1}^{K} f'_{z_i}\Bigg\{\sqrt{N}\Bigg[\lambda_i'\sqrt{l_i}\,(1-m_i(w))\sum_{j=1}^{K} l_j m_j(w)-\mu_i\sqrt{l_i}\, m_i(w)-\sqrt{l_i}\, m_i'(w)\Bigg] \\
&\quad + \lambda_i'\sqrt{l_i}\,(1-m_i(w))\Bigg(\sum_{j=1}^{K}\sqrt{l_j}\, z_j\Bigg)-\lambda_i' z_i\sum_{j=1}^{K} l_j m_j(w)-\mu_i z_i+O(N^{-1/2})\Bigg\} \\
&\quad + \frac{1}{2}\sum_{i=1}^{K} f''_{z_i}\Bigg\{\lambda_i'(1-m_i(w))\times\sum_{j=1}^{K} l_j m_j(w)+\mu_i m_i(w)+O(N^{-1/2})\Bigg\}.
\end{aligned}
\tag{A9}
$$

Choose the functions m_i so that they satisfy the system of differential equations (4.5). Let $N\to\infty$. Then for f in the above class of functions, the operator A_N converges

to yield

$$A_\infty f(z) = \sum_{i=1}^{K} f'_{z_i}\left\{\lambda'_i\sqrt{l_i}\,(1-m_i(w))\sum_{j=1}^{K}\sqrt{l_j}\,z_j - \lambda'_i z_i \sum_{j=1}^{K} l_j m_j(w) - \mu_i z_i\right\}$$
$$+\frac{1}{2}\sum_{i=1}^{K} f''_{z_i}\left\{\lambda'_i(1-m_i(w))\left(\sum_{j=1}^{K} l_j m_j(w)\right) + \mu_i m_i(w)\right\}.$$

The operator A_∞ is the infinitesimal operator of the diffusion whose stochastic differential equation is (4.6) (cf. Arnold (1974, p. 152)). The Trotter-Kato theorem can now be applied to assert that the semigroups converge (cf. Burman (1979)).

REFERENCES

L. ARNOLD (1974), *Stochastic Differential Equations: Theory and Applications*, John Wiley, New York.

D. BURMAN (1979), *An analytic approach to diffusion approximations in queueing*, Unpublished Ph.D. dissertation, Dept. Math., New York University, New York.

E. A. CODDINGTON AND N. LEVINSON (1955), *Theory of Ordinary Differential Equations*, McGraw-Hill, New York.

E. G. COFFMAN, R. R. MUNTZ AND H. TROTTER (1970), *Waiting time distributions for processor-sharing systems*, J. Assoc. Comput. Mach., 17, pp. 123-130.

E. B. DYNKIN (1965), *Markov Processes*, Vol. 1, Springer-Verlag, Berlin.

W. FELLER (1957), *An Introduction to Probability Theory and Its Applications*, Vol. I, 2nd ed., John Wiley, New York.

——— (1971), *An Introduction to Probability Theory and Its Applications*, Vol. 2, 2nd ed., John Wiley, New York.

D. P. GAVER, P. A. JACOBS AND G. LATOUCHE (1984), *The normal approximation and queue control for response times in a processor-shared computer system model*, Naval Postgraduate School Technical Report, NPS 55-84-001.

D. P. GAVER AND P. A. JACOBS (1985), *Processor-shared time-sharing models in heavy traffic*, Naval Postgraduate School Technical Report, NPS-85-004.

D. L. IGLEHART (1965), *Limiting diffusion approximations for the many-server queue and repairman problem*, J. Appl. Probab., 2, pp. 429-441.

T. KATO (1976), *Perturbation Theory for Linear Operators*, Springer-Verlag, Berlin.

J. KEILSON, *Markov Chain Models—Rarity and Exponentiality*, Springer-Verlag, New York, 1979.

F. P. KELLY (1979), *Reversibility and Stochastic Networks*, John Wiley, New York.

J. P. LEHOCZKY AND D. P. GAVER (1981), *Diffusion approximations for the cooperative service of voice and data messages*, J. Appl. Probab., 18, pp. 660-671.

P. A. W. LEWIS AND L. URIBE (1981), *The new naval postgraduate school random number package*—LLRANDOMII, Naval Postgraduate School Technical Report, NPS-81-005.

D. MITRA (1981), *Waiting time distributions from closed queueing network models of shared processor systems*, Bell Laboratories Report.

D. MITRA AND J. A. MORRISON (1983), *Asymptotic expansions of moments of the waiting time in closed and open processor-sharing systems with multiple job classes*, Adv. Appl. Probab., 15, pp. 813-839.

J. A. MORRISON AND D. MITRA (1985), *Heavy-usage asymptotic expansions for the waiting time in closed processor-sharing systems with multiple classes*, Adv. Appl. Probab., 17, pp. 163-185.

T. J. OTT (1984), *The sojourn time distribution in the* M/G/1 *queue with processor sharing*, J. Appl. Probab., 21, pp. 360-378.

S. PORNSURIYA (1984), *Normal approximations for response time in a processor-shared computer system model*, M. S. thesis, Naval Postgraduate School, Monterey, CA.

V. RAMASWAMI (1984), *The sojourn time in the* GI/M/1 *queue with processor sharing*, J. Appl. Probab., 21, pp. 437-442.

H. F. TROTTER (1974), *Approximation and Perturbation of Semigroups*, P. L. Butzer and B. Szokefalvi-Nagy, eds., Conference on Linear Operators and Approximations, 2nd 1974, Mathematisches Forschungsinstitut Oberwolfach Proceedings, Birkhauser, Basel.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
017

SOME OBSERVATIONS ABOUT THE RANDOMNESS OF HARD PROBLEMS*

DUNG T. HUYNH†

Abstract. In this note we investigate some connections between hard languages and random languages. We show that there exist languages that are both hard and random. We also show that every EXPTIME-hard language is polynomial-time weakly random.

Key words. complexity, hardness, immunity, language, randomness, P, EXPTIME

1. Introduction. Following Church's definition of random sequences [4] (which have been studied further in [5], [9] in the subrecursive setting), Wilber introduced in [11] the notion of polynomial-time (P-) random languages. Intuitively, a language L is P-random if no polynomial-time algorithm that serves as an acceptor for L can be correct with a probability greater than $\frac{1}{2}$. It is shown in [11] that there exists a P-random language in EXPTIME that is P-isomorphic to an EXPTIME-complete language that is not P-random. This means that P-random languages are not invariant under polynomial-time isomorphisms. This phenomenon is somewhat unsatisfactory. Moreover, with this strong notion of P-randomness, it seems unlikely, as noted in [11], that one can show that natural random languages exist.

In this paper we will show that the notions of random languages and hard (immune) languages are independent. The idea is that the randomness of a language is invariant under slight variations, whereas immunity is a property that can rely on a very sparse subset of the language under consideration. We also show that with a somewhat weaker notion of P-random languages that is invariant under polynomial-time isomorphisms every EXPTIME-hard language is P-random. This result has an interesting consequence: Any polynomial-time algorithm with yes/no answers that "tries to recognize" an EXPTIME-hard language must be wrong on an exponentially dense set of input instances. (This improves substantially Theorem 4.3 in [12] which states that the density of instances with wrong answers cannot be $O(\log\log(n))$.)

The second result mentioned above can also be related to a classical approach in mathematics as explained in the following. In various areas of mathematics there is an interesting approach to solve hard problems: instead of trying to obtain correct answers on all input instances (which is not feasible for provably hard problems), one develops easily decidable invariants that allow oneself to obtain correct answers on many (but not all) input instances. For example, consider the isomorphism problem for groups, which is known to be undecidable. A polynomial-time decidable invariant for this problem may be obtained as follows: from the presentations of two groups G_1, G_2 in the input, one adds commutative relations and checks in polynomial time whether the resulting Abelian groups are isomorphic; in case of a negative outcome one concludes that G_1, G_2 are not isomorphic, whereas they may be nonisomorphic in the other case. (See also [6] for an invariant for the equivalence problem for context-free grammars, and [7] for its polynomial-time version.)

The above idea about polynomial-time computable invariants for computationally intractable problems can be formalized in the complexity-theoretic fashion as follows: For a language $L \subset \Sigma^*$, $L_1 \in P$ is said to be a polynomial-time invariant for L if $L_1 \supset L$.

* Received by the editors April 3, 1985, and in revised form November 19, 1985.
† Computer Science Department, Iowa State University, Ames, Iowa 50011.

Thus, a polynomial-time algorithm for L_1 is also a polynomial-time algorithm "for" L that gives wrong answers on instances in $L_1 - L$. From our main result, it follows that if L is EXPTIME-hard, then $L_1 - L$, must have exponential density, where L_1 is any polynomial-time invariant for L.

2. Preliminaries. In this note, $\mathbf{N}$ denotes the set of nonnegative integers. All languages are subsets of Σ^*, where Σ is the binary alphabet $\{0, 1\}$. For a string $w \in \Sigma^*$, $|w|$ denotes its length. For a set $S \subset \Sigma^*$, card (S) denotes the cardinality of S, $S^c := \Sigma^* - S$ denotes its complement. $\Sigma^* = \{w_i | i \geqq 1\}$ is linearly ordered as follows: the index i of w_i is the integer with binary representation $1w_i$. From this ordering we define a linear ordering on $\Sigma^* \times \Sigma^*$ as follows: (w_i, w_j) precedes (w_k, w_l) iff either $i < k$, or $i = k$ and $j < l$.

For a set $S \subset \Sigma^*$, the *census function* of S, denoted by d_S, is a function from $\mathbf{N}$ into $\mathbf{N}$ defined by:

$$d_S(n) := \text{card}\,(\{w \in S: |w| \leqq n\}).$$

S is said to have *subexponential density* if for every $\varepsilon > 0$, $d_S(n) = o(2^{n^\varepsilon})$. If S and S^c do not have subexponential density, then S is said to have *symmetric density.* If S does not have subexponential density, it is said to have *exponential density.*

We use the multi-tape deterministic Turing machine (DTM) as our computational model. Time complexity classes are defined as usual. EXPTIME $:= \bigcup_{c>0} \text{DTIME}\,(2^{cn})$. We consider only polynomial-time bounded many-one reductions ($\leqq_m^p$). A language is EXPTIME-hard if it is EXPTIME-hard w.r.t. $\leqq_m^p$.

A transducer is a DTM with an extra write-only output tape. A transducer computes a value y on an input string x if there is an accepting computation on input x for which y is the output string. F-DTIME $(t(n))$ denotes the class of functions from Σ^* into Σ^* that are computable by $t(n)$-time bounded transducers.

3. P-random languages and P-bi-immune languages. Following [4], P-random languages have been introduced in [11].

DEFINITION 1. A language L is said to be DTIME $(t(n))$ *random* iff for all $L_1 \in \text{DTIME}\,(t(n))$, the census function $d_{L'}$ of $L' := (L \cap L_1) \cup (L^c \cap L_1^c)$ satisfies the condition that $\lim_{n\to\infty} 2^{-(n+1)} d_{L'}(n)$ exists and is $\frac{1}{2}$. L is P-*random* iff it is DTIME (n^k) random for all $k \in \mathbf{N}$.

The above definition intuitively means that no polynomial-time "acceptor" for a P-random language can be right more than 50% of the time. In some sense, this suggests that such a language is hard. On the other hand, almost everywhere hard languages have been extensively studied in complexity theory (cf., e.g. [2]). In the following we want to compare these two notions.

DEFINITION 2. A language L is said to be DTIME $(t(n))$- *immune* iff L is infinite and it does not contain any infinite subset in DTIME $(T(n))$. L is said to be DTIME $(t(n))$- *bi-immune* iff both L and L^c are DTIME $(t(n))$-immune. L is said to be P-(*bi-*)*immune* iff L is DTIME (n^k)-(bi-)immune for all $k \in \mathbf{N}$.

Thus, P-immune languages are hard in the sense that they contain only finite subsets in P. The first observation is that a P-random language is not necessarily P-immune.

LEMMA 3. *Let L be a* P-*random language. Then the union or difference of L with any subexponentially dense set is again* P-*random.*

Proof. Obvious. □

PROPOSITION 4. *There is a* P-*random language that is not* P-*immune.*

Proof. Let L be a P-random language. By Lemma 3, $L \cup 1^*$ is again P-random. However, it is obviously not P-immune. □

Next, we want to show that a P-bi-immune language is not necessarily P-random. It has been observed in [9] that there exist almost everywhere hard (immune) languages with subexponential (or even polynomial) density. Such languages cannot be random. In the following we show that for "any" given function d from $\mathbf{N}$ into $\mathbf{N}$ there is a P-bi-immune language in EXPTIME that has d as census function and is not P-random.

LEMMA 5 [1], [8]. *Let $d : \mathbf{N} \to \mathbf{N}$ be a strictly monotone increasing function with $d(0) = 0$, $n \leqq d(n) \leqq 2^{n+1} - (n+2)$ for all $n \geqq 1$ such that $0^n \mapsto d(n)$ is computable in polynomial-time. Then there exists a P-bi-immune language in* EXPTIME *that has d as its census function.*

Proof sketch. For the sake of completeness we reproduce a construction in [1] here. Let $\{T_i | i \in \mathbf{N}\}$ be an enumeration of transducers. S will be a list of indices of transducers to be diagonalized. The desired language will be constructed in stages.

Stage O:
$L := \varnothing$;
$S := \{0\}$;
Stage n > O:
$S := S \cup \{n\}$;
if there is some $i \in S$ and $x, y \in \Sigma^*$ (x precedes y in the linear ordering on Σ^*) with $|x| \leqq |y| = n$ such that there are two accepting computations of T_i on x and y, respectively, that perform at most 2^n steps and $T_i(x) = T_i(y)$
then let i_0 be the least i with this property and x_0, y_0 be the smallest corresponding pair (x, y);
if x_0 does not belong to L **then** $L := L \cup \{y_0\}$;
$S := S - \{i_0\}$;
Add to L the smallest strings of length n which are distinct from x_0, y_0 (if they have been found) so that $d_L(n) = d(n)$;
go to next stage;
end.

It can be shown that L is P-bi-immune (even P-strongly-bi-immune (cf. Definition 8 and Fact 9 below)) and $L \in$ EXPTIME. □

From Lemma 5 we obtain

PROPOSITION 6. *Let $d : \mathbf{N} \to \mathbf{N}$ be a strictly monotone increasing function with $d(0) = 0$, $n \leqq d(n) \leqq 2^{n+1} - (n+2)$ for all $n \geqq 1$ such that $0^n \mapsto d(n)$ is computable in polynomial-time. Then there is a language L in* EXPTIME *with census function d so that L is* P*-bi-immune but not* P*-random.*

Proof. Let L be the language constructed in the proof of Lemma 5. Observe that, by construction, for all $n \geqq 1$, the first $d(n) - d(n-1)$, except possibly one or two, strings of length n that begin with 0 are in L. Obviously, L is not P-random. □

From Proposition 6 we obtain

COROLLARY 7. *There is a* P*-bi-immune language L with census function d_L satisfying $d_L(0) = 0$, $d_L(1) = 1$ and $d_L(n) = 2^n$ for all $n \geqq 2$ so that L is in* EXPTIME *and is not* P*-random.*

Propositions 4 and 6 and Corollary 7 reveal the fact that the notions of P-random languages and P-immune languages are independent. From these results, it seems interesting to show that there exist languages in EXPTIME that are both P-bi-immune and P-random. In the following we show that such languages do exist. For technical

convenience, we introduce the notion of "strong bi-immunity" (cf. [1]), which has been motivated by a construction in [3].

DEFINITION 8. A function $f:\Sigma^*\to\Sigma^*$ is said to be *one-one almost everywhere* ($1-1$ a.e.) on a set S iff there are at most finitely many pairs $x, y\in S$ such that $x\neq y$ and $f(x)=f(y)$. A language L is said to be DTIME $(t(n))$ *strongly-bi-immune* iff every function $f:\Sigma^*\to\Sigma^*$ in F-DTIME $(t(n))$ that satisfies $f(L)\cap f(L^c)=\varnothing$ is 1-1 a.e. on both L and L^c.

The difference between the notions of DTIME $(t(n))$-bi-immune languages and DTIME $(t(n))$-strongly-bi-immune languages is as follows. DTIME $(t(n))$-bi-immune languages can also be defined as in Definition 8 by weakening the requirement on f: $f^{-1}(w)$ has to be finite for all w.

FACT 9. *If L is* DTIME $(t(n))$ *strongly-bi-immune, then it is* DTIME $(t(n))$ *bi-immune.*

In the following we will show that there exist languages in EXPTIME that are simultaneously P-strongly-bi-immune and P-random. The existence of such languages will have an interesting consequence in the next section. We introduce a technical notion. A language L_1 is said to be obtained from L by polynomial variation if their symmetric difference has polynomial density.

LEMMA 10. *Let L be a language in* EXPTIME *that is* P*-random. Then there is a language L_1 obtained from L by polynomial variation so that L_1 is* P*-strongly-bi-immune.*

Proof. Consider the diagonalization in the proof of Lemma 5. At stage n, at most one string must be added to L and at most one string must be added to L^c in order to make L P-strongly-bi-immune. It can easily be seen that by polynomial variation, any P-random language L in EXPTIME (whose existence has been shown in [11]) can be transformed, by the same method, into a P-strongly-bi-immune language that is still in EXPTIME. □

From Lemmas 3, 10 and Fact 9, we obtain as corollary

THEOREM 11. *There is a language in* EXPTIME *that is both* P*-bi-immune and* P*-random.*

4. P-weakly-random languages. As pointed out in [10, p. 52], the notion of recursive invariance plays a fundamental role in recursive function theory: almost all concepts are recursively invariant. In complexity theory, many notions concerning recursive languages are invariant under polynomial-time isomorphisms. Unfortunately, the notion of P-randomness, as defined above, is not polynomially invariant. Indeed, it has been shown in [11] that there is a P-random language that is polynomially isomorphic to an EXPTIME-complete language that is not P-random. This means that P-randomness is a too strong notion in studying properties of complete languages. Moreover, with respect to the above notion of P-randomness, it seems unlikely, as noted in [11], that one can show that certain natural problems are P-random. To avoid these difficulties, we introduce a weaker notion of P-randomness that is polynomially invariant, and show that every EXPTIME-hard language is P-random under this new notion.

DEFINITION 12. A language L is said to be DTIME $(t(n))$ *weakly-random* iff for every $L_1\in$ DTIME $(t(n))$ the language $L':=(L\cap L_1)\cup(L^c\cap L_1^c)$ has symmetric density. L is P*-weakly-random* iff L is DTIME (n^k) weakly-random for all $k\in\mathbf{N}$.

PROPOSITION 13. *If L is* P*-weakly-random, then $f(L)$ is* P*-weakly-random for every polynomial-time isomorphism f.*

Proof. Obvious. □

From Lemma 10 we obtain

THEOREM 14. *Every* EXPTIME*-hard language is* P*-weakly-random.*

Proof. Let L be an EXPTIME-hard language. Let L_1 be the language constructed in Lemma 10: L_1 is P-strongly-bi-immune and P-random, and $L_1 \in$ EXPTIME. Since L is EXPTIME-hard, there is a polynomial-time reduction f from L_1 to L. f must be 1-1 a.e. on L_1 and L_1^c, because L is P-strongly-bi-immune. Therefore, $f(L_1) \subset L$ and $f(L_1^c) \subset L^c$ both are not subexponentially dense, because L_1 has symmetric density. Now, it is obvious that L is P-weakly-random. □

Remark 15. The set $K := f(L_1) \cup f(L_1^c)$ in the proof of Theorem 14 is also a polynomial complexity core for L (cf. [8]): any DTM that recognizes L requires a nonpolynomial number of steps on almost all instances in K. On the other hand, if we allow algorithms to give wrong answers, then the proof of Theorem 14 implies that any polynomial-time bounded DTM that "tries to recognize" L must give wrong answers on an exponentially dense subset of K. (In a fashion similar to [8], this result can be generalized for other time and space complexity classes.)

Acknowledgments. The author is grateful to two anonymous referees for helpful suggestions that greatly improve the presentation of this paper.

REFERENCES

[1] J. L. BALCAZAR AND U. SCHÖNING, *Bi-immune sets for complexity classes*, Math. Systems Theory, to appear.

[2] L. BERMAN, *On the structure of complete sets: almost everywhere complexity and infinitely often speedup*, Proc. 17th IEEE Symposium on Foundations of Computer Science, 1976, pp. 76-80.

[3] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of* NP *and other complete sets*, this Journal, 6 (1977), pp. 305-322.

[4] A. CHURCH, *On the concept of random sequence*, Bull. AMS, 46 (1940), pp. 130-135.

[5] R. A. DI PAOLA, *Random sets in subrecursive hierarchies*, J. ACM, 16 (1969), pp. 621-630.

[6] G. HOTZ, *Eine Neue Invariante für Kontext-Freie Sprachen*, Theor. Comp. Sci., 11 (1980), pp. 107-116.

[7] D. T. HUYNH, *Remarks on the complexity of an invariant of context-free grammars*, Acta Informatica, 17 (1982), pp. 89-99.

[8] D. T. HUYNH, *Exponentially dense complexity cores for provably intractable problems*, manuscript, 1984.

[9] A. R. MEYER AND E. M. MCCREIGHT, *Computationally complex and pseudo-random zero-one valued functions*, in Theory of Machines and Computations, Kohavi and Paz, eds., Academic Press, New York, 1971, pp. 19-42.

[10] H. ROGERS, *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York, 1967.

[11] R. E. WILBER, *Randomness and the density of hard problems*, Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 335-342.

[12] Y. YESHA, *On certain polynomial-time truth-table reducibilities of complete sets to sparse sets*, this Journal, 12 (1983), pp. 411-425.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
018

PROBABILISTIC ANALYSIS OF TWO HEURISTICS FOR THE 3-SATISFIABILITY PROBLEM*

MING-TE CHAO† AND JOHN FRANCO‡

Abstract. An algorithm for the 3-Satisfiability problem is presented and a probabilistic analysis is performed. The analysis is based on an instance distribution which is parameterized to simulate a variety of sample characteristics. The algorithm assigns values to variables appearing in a given instance of 3-Satisfiability, one at a time, using the unit clause heuristic and a maximum occurring literal selection heuristic; at each step a variable is chosen randomly from a subset of variables which is usually large. The algorithm runs in polynomial time and it is shown that the algorithm finds a solution to a random instance of 3-Satisfiability with probability bounded from below by a constant greater than zero for a range of parameter values. The heuristics studied here can be used to select variables in a Backtrack algorithm for 3-Satisfiability. Experiments have shown that for about the same range of parameters as above the Backtrack algorithm using the heuristics finds a solution in polynomial average time.

Key words. satisfiability, probabilistic analysis, heuristics

1. Introduction. This paper is concerned with the probabilistic performance of two heuristics for the 3-Satisfiability problem (3-SAT). 3-SAT is the problem of determining whether all of a collection of 3-literal disjunctions (clauses) of Boolean variables are *true* for some truth assignment to the variables. This problem is NP-complete so there is no known polynomial time algorithm for solving it. 3-SAT is a special case of the Satisfiability problem (SAT) which is the problem of determining whether all of a collection of clauses are *true* for some truth assignment to the variables contained in those clauses.

The analysis is based on an equally likely instance distribution which has been used in other studies of algorithms for this problem. This model has two parameters: n, the number of clauses, and r, the number of variables from which clauses are composed. The model (which we refer to as $M(n, r, 3)$) is described in greater detail in the next section. In [7] it was shown that, under $M(n, r, 3)$, if $\lim_{n,r\to\infty}(n/r) > 5.2$ then random instances have no solution with probability approaching 1. In [2] it was reported that, according to experiments, random instances have no solution with probability approaching 1 if $\lim_{n,r\to\infty}(n/r) > 4$. If $\lim_{n,r\to\infty}(n/r) = B$ and B is a constant less than 4 then the average number of truth assignments satisfying a random instance of 3-SAT is exponential in n; however, the probability that a random truth assignment satisfies a random instance tends to zero as $n \to \infty$ so, although random instances have many solutions on the average, these instances are not so trivial that a random probe is likely to result in a satisfying truth assignment. Still, it may be argued that the model generates instances with more satisfying assignments than one would expect in a practical setting. We respond by pointing out that perhaps the greatest value of the results presented here is that they may be compared with results obtained for other heuristics for 3-SAT under the same model.

Results obtained previously and in this paper under model $M(n, r, 3)$ are as follows. In [1] it was shown that Backtracking solves 3-SAT in exponential average time for all limiting ratios of n to r which are constant. In [7] it was shown that a

* Received by the editors January 7, 1985, and in revised form September 7, 1985. This work was supported in part by Air Force grant AFOSR 84-0372.

† Case Western Reserve University, Department of Computer Engineering and Science, Cleveland, Ohio 44106.

‡ Indiana University, Department of Computer Science, Bloomington, Indiana 47405.

variant of the Davis–Putnam Procedure [3] which searches for all solutions to the given instance requires exponential time in probability under $M(n, r, 3)$ for all limiting ratios of n to r which are constant. But, in [5] it was shown that the Pure-Literal heuristic can be used to solve random instances of 3-SAT in polynomial time with probability approaching 1 when $\lim_{n,r\to\infty}(n/r)<1$. In this paper it is shown that the Unit-Clause heuristic and a maximum occurring literal selection heuristic can be used to solve random instances of 3-SAT in polynomial time with probability bounded from below by a constant when $\lim_{n,r\to\infty}(n/r)<2.9$. A similar analysis shows that the Unit-Clause heuristic alone solves random instances in polynomial time with bounded probability when $\lim_{n,r\to\infty}(n/r)<2.66$. These results are useful because they indicate the effectiveness of the two heuristics when used in a Backtrack algorithm for 3-SAT. Experiments suggest that Backtracking, using the two heuristics to determine which literal to consider at each step, will verify in polynomial average time that a solution exists for about the same range of limiting ratios of n to r.

There are a number of papers which investigate the probabilistic performance of SAT; these papers present results which are closely related to the results obtained for 3-SAT. These results are based on the constant-density model for SAT: construct each of n clauses independently by placing each of r variables independently in a clause with probability p and complementing those variables in each clause with probability $\frac{1}{2}$. Average case results using the constant-density model or variation are in [1], [8], [9], [10] and [11]. Probabilistic results using the constant-density model are in [6]. According to the results in [6], when the average number of literals in a clause is 3, random instances of SAT are nearly always proven to have no solutions in polynomial time.

2. 3-Satisfiability and the probabilistic model. The following terms are used to describe 3-SAT. Let $V=\{v_1, v_2 \cdots v_r\}$ be a set of r Boolean variables. Associated with each variable v_i is a positive literal, denoted by v_i, and a negative literal, denoted by $\bar{v}_i$ and literal v_i has value *true* iff the variable v_i has value *true* and literal $\bar{v}_i$ has value *true* iff the variable v_i has value *false*. The literals v_i and $\bar{v}_i$ are said to be complementary. If l is a literal then comp (l) is the literal which is complementary to l. A clause is a subset of the set of all literals associated with the variables of V such that no two literals in the subset are complementary. A truth assignment to V is an assignment of truth values to every variable in V. A clause c is satisfied by truth assignment t if at least one literal in c has value *true* under t. Let $A_i(V)$ denote the set of i-literal clauses that can be composed of literals associated with the variables of V. An instance I of 3-SAT is a collection of clauses chosen from $A_3(V)$ and the problem is to find a truth assignment to V which satisfies all clauses in I, if one exists, and to verify that no such truth assignment exists otherwise. A truth assignment which satisfies all clauses in I is said to be a solution to I.

The probabalistic model used for analysis is presented by describing the method used to construct random instances. A random instance of 3-SAT contains n clauses chosen uniformly, independently and with replacement from $A_3(V)$. The distribution associated with this construction is referred to as $M(n, r, 3)$.

3. The algorithm SC_1. The algorithm we consider, called SC_1, takes as input a collection of clauses I and outputs "a solution exists" or "cannot determine whether a solution exists". SC_1 contains a single loop. At each iteration of the loop a literal is chosen and some clauses and literals are removed from I. Let $C_i^{I,\sigma}(j)$, for all $1\leqq i\leqq 3$, denote the collection of clauses in I containing exactly i literals at the end of the jth iteration where σ denotes the sequence of chosen literals. We shorten $C_i^{I,\sigma}(j)$ to $C_i(j)$.

Then $C_i(0) = \varnothing$ for all $1 \leqq i \leqq 2$ and $|C_3(0)| = n$. If the $j+1$st chosen literal is l then the lines

Remove from I all clauses containing l
Remove from I all occurrences of comp (l)

have the following effect:

$$\forall 1 \leqq i \leqq 2 \; C_i(j+1) = \{c: c \in C_i(j) \text{ and } l \notin C_i(j) \text{ and } \text{comp}(l) \notin C_i(j) \text{ or } c \cup \{\text{comp}(l)\} \in C_{i+1}(j)\}$$
$$C_3(j+1) = \{c: c \in C_3(j) \text{ and } l \notin C_3(j) \text{ and } \text{comp}(l) \notin C_3(j)\}.$$

In what follows card $(v, C_3(j))$ is the number of clauses in $C_3(j)$ which contain the literal v and card $(\bar{v}, C_3(j))$ is the number of clauses in $C_3(j)$ which contain the literal $\bar{v}$. Clauses in $C_1(j)$ are said to be unit clauses. Finally, var (l) is the variable associated with literal l.

```
SC1(I):
    j ← 0
    Repeat
      If |C1(j)| = 0 Then Begin
        Choose v randomly from V
        V ← V − {v}
        If card(v̄, C3(j)) > card(v, C3(j)) Then l ← v̄ Else l ← v
        End
      Else Begin
        Choose l randomly from C1(j)
        V ← V − {var(l)}
        End
      Remove from I all clauses containing l
      Remove from I all occurrences of comp(l)
      j ← j + 1
    Until I is empty or there exist two complementary unit clauses in I
    If I is empty Then Output ("a solution exists")
      Else Output ("cannot determine whether a solution exists")
```

The reader may be disturbed by the loop condition since it is more natural to break out of the loop when a null clause has been created in I than when two complementary unit clauses exist in I. We have chosen to write $SC_1(I)$ as above because, in our opinion, the analysis is slightly easier and more natural. Our results, of course, hold in either case.

SC_1 runs in less than $O(r^2 n)$ time since I must be empty after r iterations of the loop and the remove and *card* operations need look at no more than $r * n$ literals. An instance I of SAT has a solution if SC_1 run on I outputs "a solution exists": one solution to I may be found by assigning the value *true* to the variables whose positive literals were chosen and the value *false* to all other variables.

4. Analysis of SC_1. In this section it is shown that if instances are generated according to $M(n, r, 3)$ and $\lim_{n,r\to\infty}(n/r) < 2.9$ then for some $\varepsilon > 0$, the probability that SC_1 outputs "a solution exists" is greater than ε.

The following theorem will be used to show how the collections of clauses in $C_i(j)$ are distributed.

THEOREM 1. *Let V_{r-j} be the subset of variables associated with unchosen literals after j literals have been chosen. Suppose for all $1 \leqq i \leqq 3$ the clauses in $C_i(j)$ are independent and are equally likely to be any clause in $A_i(V_{r-j})$. Then for all $1 \leqq i \leqq 3$ the clauses in $C_i(j+1)$ are independent and equally likely to be any clause in $A_i(V_{r-j-1})$.*

Proof. Either the variable v is chosen randomly from V_{r-j} if $|C_1(j)| \neq 0$ or it is chosen randomly from $C_1(j)$. Consider the first case. Let c_1 and c_2 be two clauses in $C_i(j+1)$, let $\hat{c}_1$ and $\hat{c}_2$ be the two clauses in $C_i(j)$ or $C_{i+1}(j)$ from which c_1 and c_2 were derived after the $j+1$st literal was shown and let x, x_1, x_2, y, y_1 and y_2 be arbitrary clause configurations of i literals. Then

$$\begin{aligned}\text{pr}(c_1 = x) &= \text{pr}(\hat{c}_1 = x \text{ or } \hat{c}_1 = x \cup \{v\} \text{ and } \bar{v} \text{ was chosen or } \hat{c}_1 = x \\ &\qquad \cup \{\bar{v}\} \text{ and } v \text{ was chosen}) \\ &= \text{pr}(\hat{c}_1 = y \text{ or } \hat{c}_1 = y \cup \{v\} \text{ and } \bar{v} \text{ was chosen or } \hat{c}_1 = y \\ &\qquad \cup \{\bar{v}\} \text{ and } v \text{ was chosen}) \\ &= \text{pr}(c_1 = y).\end{aligned}$$

Also,

$$\begin{aligned}&\text{pr}(c_1 = x_1 \text{ and } c_2 = x_2) \\ &\quad = \text{pr}(\hat{c}_1 = x_1 \text{ and } \hat{c}_2 = x_2 \text{ or } \hat{c}_1 = x_1 \text{ and } \hat{c}_2 = x_2 \cup \{v\} \text{ and } \bar{v} \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = x_1 \text{ and } \hat{c}_2 = x_2 \cup \{\bar{v}\} \text{ and } v \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = x_1 \cup \{v\} \text{ and } \hat{c}_2 = x_2 \text{ and } \bar{v} \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = x_1 \cup \{\bar{v}\} \text{ and } \hat{c}_2 = x_2 \text{ and } v \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = x_1 \cup \{v\} \text{ and } \hat{c}_2 = x_2 \cup \{v\} \text{ and } \bar{v} \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = x_1 \cup \{\bar{v}\} \text{ and } \hat{c}_2 = x_2 \cup \{\bar{v}\} \text{ and } v \text{ was chosen}) \\ &\quad = \text{pr}(\hat{c}_1 = y_1 \text{ and } \hat{c}_2 = y_2 \text{ or } \hat{c}_1 = y_1 \text{ and } \hat{c}_2 = y_2 \cup \{v\} \text{ and } \bar{v} \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = y_1 \text{ and } \hat{c}_2 = y_2 \cup \{\bar{v}\} \text{ and } v \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = y_1 \cup \{v\} \text{ and } \hat{c}_2 = y_2 \text{ and } \bar{v} \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = y_1 \cup \{\bar{v}\} \text{ and } \hat{c}_2 = y_2 \text{ and } v \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = y_1 \cup \{v\} \text{ and } \hat{c}_2 = y_2 \cup \{v\} \text{ and } \bar{v} \text{ was chosen or} \\ &\qquad\qquad \hat{c}_1 = x_1 \cup \{\bar{v}\} \text{ and } \hat{c}_2 = y_2 \cup \{\bar{v}\} \text{ and } v \text{ was chosen}) \\ &\quad = \text{pr}(c_1 = y_1 \text{ and } c_2 = y_2).\end{aligned}$$

And so on.

Consider the second case. The $j+1$st chosen variable is equally likely to be any of $r-j$ variables and is selected independently of clauses in $C_i(j)$ for all $2 \leqq i \leqq 3$. Hence for all $2 \leqq i \leqq 3$ we may use the proof of the first case. For $i = 1$ the result follows from the independence and equal likelihood of the unit clauses.

COROLLARY 1. *For all $0 \leqq j \leqq r$ and $1 \leqq i \leqq 3$ all clauses in $C_i(j)$ are independent and equally likely to be any clause in $A_i(V_{r-j})$.*

Proof. By induction on j. The basis step holds because of the assumed distribution on instances given to SC_1. The induction step holds because of Theorem 1.

Because of Corollary 1 a system of differential equations for finding the expected number of clauses in $C_i(j)$ for all $2 \leqq i \leqq 3$ may be obtained. Let $n_i(j)$ denote the number of clauses in $C_i(j)$, let $w_i(j)$ denote the number of i—literal clauses added to $C_i(j)$ as a result of choosing the jth variable and let $z_i(j)$ denote the number of clauses eliminated from $C_i(j)$ as a result of choosing the jth variable. These three terms depend

on I and σ but this dependence is omitted from the terms for the sake of simplicity. The $w_i(j)$ term may be thought of as representing the "rate of flow" of clauses into $C_i(j)$ when the jth variable is chosen and the $z_i(j)$ term may be thought of as representing the "rate of flow" of clauses out of $C_i(j)$ when the jth variable is chosen. If the average rate of flow into $C_1(j)$ is always less than 1 the number of clauses in $C_1(j)$ will not, in probability, grow very large since at least one clause is removed from $C_1(j)$ whenever $C_1(j) \neq \varnothing$. In this case the probability that a complementary pair of clauses exists in $C_1(j)$ for some j is small. However, if the average rate of flow into $C_1(j)$ rises above 1 for a constant fraction of the values of j/r then the number of clauses in $C_1(j)$ gets large for a fraction of the values of j/r since the flow out of $C_1(j)$ is asymptotically no more than one unless $|C_1(j)|$ is large. In this case the probability that there is a complementary pair of clauses in $C_1(j)$ for some j is near 1. Since, as will be seen from the analysis below, if the expected flow into $C_1(j)$ goes above $1+\varepsilon$ for any $\varepsilon > 0$ then it stays above 1 for a constant fraction of values of j/r, the point at which $E\{w_1(j)\}$ (the expectation of $w_1(j)$—from now on all expectations will be written similarly) is around 1 is a critical one regarding the probabilistic performance of SC_1.

We now develop the differential equations for finding $E\{w_1(j)\}$, solve them and find the condition on n/r which causes $E\{w_1(j)\} < 1$. Later, it will be shown that this implies SC_1 finds a satisfying truth assignment when one exists with probability greater than some positive constant.

Clearly, for $1 \leqq i \leqq 3$

$$n_i(j+1) = n_i(j) + w_i(j+1) - z_i(j+1).$$

Taking expectations gives

$$E\{n_i(j+1)\} = E\{n_i(j)\} + E\{w_i(j+1)\} - E\{z_i(j+1)\}$$

which can be written

$$E\{n_i(j+1)\} - E\{n_i(j)\} = E\{w_i(j+1)\} - E\{z_i(j+1)\}. \tag{1}$$

For large r we can approximate (1) by

$$\frac{dE\{n_i(j)\}}{dj} = E\{w_i(j+1)\} - E\{z_i(j+1)\}. \tag{2}$$

But, for all $1 \leqq i \leqq 3$

$$\begin{aligned} E\{z_i(j+1)\} &= E\{E\{z_i(j+1) | n_i(j)\}\} \\ &= E\left\{\frac{i * n_i(j)}{r-j}\right\} = \frac{i * E\{n_i(j)\}}{r-j} \end{aligned} \tag{3a}$$

because of Corollary 1. Also,

$$\begin{aligned} E\{w_1(j+1)\} &= E\{E\{w_1(j+1) | n_2(j)\}\} \\ &= E\left\{\frac{2 * n_2(j)}{2(r-j)}\right\} = \frac{E\{n_2(j)\}}{r-j} \end{aligned} \tag{3b}$$

and

$$E\{w_3(j+1)\} = 0.$$

Finally,

$$\begin{aligned} E\{w_2(j+1)\} &= 3\frac{E\{n_3(j)\}}{2(r-j)} \\ &\quad - H_2(j+1) * \text{pr}\,(j+1\text{st chosen literal does not come from } C_1(j)) \end{aligned} \tag{4}$$

where $H_2(j+1)$ is the average number of extra clauses removed from $C_3(j)$ given the $j+1$st chosen literal does not come from $C_1(j)$. Therefore (2), for $i=3$ can be written

$$\frac{dE\{n_3(j)\}}{dj} = -\frac{3 * E\{n_3(j)\}}{r-j}. \tag{5}$$

The solution to this differential equation under the assumption that $E\{n_3(0)\}=n$ is the following.

THEOREM 2.

$$E\{n_3(j)\} = \left(1-\frac{j}{r}\right)^3 n.$$

Proof. Straightforward solution to (5).

In order to solve (2) for $i=2$ we must first find $H_2(j+1)$ and the probability that the $j+1$st chosen literal does not come from $C_1(j)$. It suffices to find a lower bound for $H_2(j+1)$ and the probability mentioned since we require only an upper bound on $E\{w_1(j)\}$.

THEOREM 3.

$$H_2(j+1) \geqq \frac{8}{9\sqrt{2\pi}} E\left\{\sum_{y=1}^{n_3(j)} \sqrt{y}\binom{n_3(j)}{y}\left(\frac{3}{r-j}\right)^y \left(1-\frac{3}{r-j}\right)^{n_3(j)-y}\right\}.$$

Proof. The probability that a particular literal appears in x clauses given the variable associated with that literal appears in y clauses is

$$\binom{y}{x}\left(\frac{1}{2}\right)^x\left(\frac{1}{2}\right)^{y-x}.$$

Hence the expected number of clauses containing the least frequently occurring literal associated with the chosen variable given y is

$$\sum_{x=0}^{\lfloor y/2\rfloor} x\binom{y}{x}\left(\frac{1}{2}\right)^y + \sum_{x=\lceil y/2\rceil}^{y} (y-x)\binom{y}{x}\left(\frac{1}{2}\right)^y = 2\sum_{x=0}^{\lfloor y/2\rfloor} x\binom{y}{x}\left(\frac{1}{2}\right)^y$$

$$= \frac{y}{2} - \frac{\frac{y+1}{2}\binom{y}{\lfloor y/2\rfloor}}{2^y} \quad \text{if } y \text{ is odd}$$

and

$$2\sum_{x=0}^{(y/2)-1} x\binom{y}{x}\left(\frac{1}{2}\right)^y + \frac{y}{2}\binom{y}{y/2}\left(\frac{1}{2}\right)^y = \frac{y}{2} - \frac{\frac{y}{2}\binom{y}{y/2}}{2^y} \quad \text{if } y \text{ is even.}$$

Let

$$G(y) = \begin{cases} \dfrac{\binom{y}{y/2}}{2^y}, & y \text{ even}, \\[2ex] \dfrac{\left(1+\frac{1}{y}\right)\binom{y}{\lfloor y/2\rfloor}}{2^y}, & y \text{ odd}. \end{cases}$$

Then

$$E\{H_2(j+1)|n_3(j)\}=\frac{1}{2}\sum_{y=0}^{n_3(j)} yG(y)\binom{n_3(j)}{y}\left(\frac{3}{r-j}\right)^y\left(1-\frac{3}{r-j}\right)^{n_3(j)-y}$$

$$>\frac{8}{9\sqrt{2\pi}}\sum_{y=0}^{n_3(j)}\sqrt{y}\binom{n_3(j)}{y}\left(\frac{3}{r-j}\right)^y\left(1-\frac{3}{r-j}\right)^{n_3(j)-y},$$

since, by Stirling's formula, $G(y)>16/9\sqrt{2\pi y}$. Taking the expectation gives the desired result.

If $E\{n_3(j)\}$ and $r-j$ are large and n/r is a constant, since $\lim_{n,r\to\infty}(E\{n_3(j)\}/(r-j))$ is bounded by a constant and since $n_3(j)$ is binomially distributed then the lower bound for $H_2(j+1)$ may be approximated by the expression

$$\frac{8}{9\sqrt{2\pi}}\beta\sqrt{\frac{3*E\{n_3(j)\}}{r-j}},$$

where β depends on the value of the expression under the large square root sign. A few values of β are as follows:

$\frac{3*E\{n_3(j)\}}{r-j}$	β
1	.7731
2	.891
4	.96
8	.983
16	.992

But, $E\{n_3(j)\}=(1-(j/r))^3 n$ so, for $1\leqq j\leqq \delta r$ where δ is any constant between zero and one, the lower bound for $H_2(j+1)$ is approximately

$$\frac{8\sqrt{3}}{9\sqrt{2\pi}}\beta\left(1-\frac{j}{r}\right)\sqrt{\frac{n}{r}}. \tag{6}$$

Only a lower bound for the probability that the $j+1$st chosen literal does not come from $C_1(j)$ still needs to be found.

THEOREM 4.

$$\Pr(j+1\text{st } \textit{chosen literal does not come from } C_1(j))\geqq 1-E\{w_1(j)\} \tag{7}$$

for all j from 1 to j_0 where $E\{w_1(j_0)\}\geqq\{w_1(j)\}$ for all $j\neq j_0$.

Proof. The flow of clauses through C_1 may be modeled as the flow through a single-server, work-conserving, nonpreemptive queueing system. In this system a unit of time corresponds to a single iteration of SC_1. At the start of every unit time interval a number of jobs (corresponding to unit clauses) arrives at the queue. The arrival rate at the start of the jth interval is $E\{w_1(j)\}$. The average service time is at most one time unit since at least one job (clause) is serviced (removed) during a unit time interval if the queue is not empty at the start of that interval. For such a system in equilibrium with constant arrival rate it is well known that the probability that the queue contains at least one job at the start of an arbitrary unit time interval is the product of arrival rate and the average service time. In this case, the product is at most the arrival rate since the average service time is at most 1. The system we consider here is not in

equilibrium, however, since the arrival rate, $E\{w_1(j)\}$ is increasing with j up to j_0. But this implies that the probability that the queue contains at least one job at the start of the jth time interval, $j \leqq j_0$, is at most the product mentioned above and, therefore, the arrival rate at the jth interval (we can add dummy jobs and increase the service time to one time interval to get a system that is in equilibrium, has at least as high a probability that the queue is not empty and that probability is $E\{w_i(j)\}$). Thus, the probability that $C_1(j) \neq \varnothing$ is less than $E\{w_1(j)\}$. The probability required is the probability that $C_1(j) = \varnothing$ and is, from the above argument, at least $1 - E\{w_1(j)\}$.

Substituting (6), (7), $(1-(j/r))^3 n$ for $E\{n_3(j)\}$ and $E\{n_2(j)\}/(r-j)$ for $E\{w_1(j)\}$ (from (3b)) into (4) and substituting the result and (3a) into (2) with i set to 2 gives

$$\frac{dE\{n_2(j)\}}{dj} = \frac{3}{2}\left(1-\frac{j}{r}\right)^2 \frac{n}{r} - \frac{2 * E\{n_2(j)\}}{r(1-j/r)} + \frac{8\sqrt{3}}{9\sqrt{2\pi}}\beta\sqrt{\frac{n}{r}}\frac{E\{n_2(j)\}}{r} - \frac{8\sqrt{3}}{9\sqrt{2\pi}}\beta\sqrt{\frac{n}{r}}\left(1-\frac{j}{r}\right). \tag{8}$$

For the moment suppose β is constant. Then the solution to (8) with boundary condition $E\{n_2(0)\} = 0$ and with α substituted for $8\sqrt{3}/9\sqrt{2\pi}$ is

$$E\{n_2(j)\} = \left(1-\frac{j}{r}\right)^2 e^{(j/r)\alpha\beta\sqrt{n/r}}\left[\frac{3}{2}\frac{n}{\alpha\beta\sqrt{n/r}}(1-e^{-(j/r)\alpha\beta\sqrt{n/r}}) + r\alpha\beta\sqrt{\frac{n}{r}}\ln\left(1-\frac{j}{r}\right) - r\left(\alpha\beta\sqrt{\frac{n}{r}}\right)^2\left(\ln\left(1-\frac{j}{r}\right)+\frac{j}{r}\right) + r\frac{(\alpha\beta\sqrt{n/r})^3}{2}\left(\ln\left(1-\frac{j}{r}\right)+\frac{j}{r}+\frac{j^2}{2r^2}\right) - \cdots\right].$$

Thus, from (3b)

$$E\{w_1(j)\} = \left(1-\frac{j}{r}\right)e^{(j/r)\alpha\beta\sqrt{n/3}}\left[\frac{3}{2}\frac{1}{\alpha\beta\sqrt{n/r}}\frac{n}{r}(1-e^{-(j/r)\alpha\beta\sqrt{n/r}}) + \alpha\beta\sqrt{\frac{n}{r}}\ln\left(1-\frac{j}{r}\right) - \left(\alpha\beta\sqrt{\frac{n}{r}}\right)^2\left(\ln\left(1-\frac{j}{r}\right)+\frac{j}{r}\right) + \frac{1}{2}\left(\alpha\beta\sqrt{\frac{n}{r}}\right)^3\left(\ln\left(1-\frac{j}{r}\right)+\frac{j}{r}+\frac{j^2}{2r^2}\right) - \cdots\right]. \tag{9}$$

The expression on the right in (9) has a maximum in the vicinity of and greater than $j = r/2$. We call the point at which the maximum occurs j_0. If $n/r = 2.9$ then $3 * E\{n_3(j_0)\}/(r-j_0) \approx 1.9$ so $\beta \approx .89$ at $j = j_0$. Since $3 * E\{n_3(j)\}/(r-j) < 3n/r < 9$, $\beta > .89$ for all $0 \leqq j < j_0$ so (8) with β set to .89 gives an upper bound on $E\{n_2(j)\}$ and therefore $E\{w_1(j)\}$ up to j_0. It can be seen from (9) that $E\{w_1(j)\}$ for $1 \leqq j \leqq j_0$ is less than 1 when $\beta = .89$ and $n/r = 2.9$.

The solution to (8) with $\beta = 0$ is an upper bound on $E\{n_2(j)\}$ in the range $j_0 \leqq j < r$. When divided by $(r-j)$ and an appropriate boundary condition is added this solution is an upper bound for $E\{w_1(j)\}$ in the range $j_0 \leqq j < r$ and has value equal to the value of $E\{w_1(j_0)\}$ at $j = j_0$. Since this bound is maximal at $j = r/2$ the maximum value of this bound in the range $j_0 \leqq j < r$ is equal to the maximum value of the first bound in the range $0 \leqq j \leqq j_0$. Hence, we get the following result.

THEOREM 5. *Given that inputs to SC_1 are distributed according to $M(n, r, 3)$,*

$$E\{w_1(j)\} < 1 \text{ for all } 0 \leqq j < r \text{ when } \lim_{n,r\to\infty} \frac{n}{r} < 2.9.$$

We now prove the main result.

THEOREM 6. *SC_1 verifies that a solution exists for satisfiable instances generated according to $M(n, r, 3)$ with probability greater than ε for some $\varepsilon > 0$ when $\lim_{n,r\to\infty}(n/r) < 2.9$.*

Proof. From Theorem 5 $E\{w_1(j)\} < 1$ for all $0 \leqq j < r$ when $\lim_{n,r\to\infty}(n/r) < 2.9$. From Corollary 1 the clauses entering $C_1(j+1)$ from $C_2(j)$ are statistically independent. Suppose all clauses entering $C_1(j+1)$ are regarded as entering $C_1(j+1)$ in some order which is decided arbitrarily. Then the probability that the qth clause entering $C_1(j+1)$ is complementary to no clause in $C_1(j+1)$ is

$$\left(1 - \frac{1}{2(r-j)}\right)^{n_1(j)+q-1}.$$

Therefore, the probability that none of the clauses entering $C_1(j+1)$ is complementary to any clause in $C_1(j+1)$ is

$$\left(1 - \frac{1}{2(r-j)}\right)^{n_1(j) * w_1(j) + w_1(j) * (w_1(j)-1)/2}$$

so the probability that no complementary pair is encountered during a run of SC_1 is

$$\sum \prod_{j=0}^{r-1} \left(1 - \frac{1}{2(r-j)}\right)^{n_1(j) * w_1(j) + w_1(j) * (w_1(j)-1)/2} \text{pr}(\cdots n_1(j), w_1(j) \cdots)$$

$$> \sum \prod_{j=0}^{r-1} \left(1 - \frac{1}{2r}\right)^{(2r/(r-j))(n_1(j) * w_1(j) + w_1(j) * (w_1(j)-1)/2)} \text{pr}(\cdots n_1(j), w_1(j) \cdots)$$

$$= \sum \left(1 - \frac{1}{2r}\right)^{2r \sum_{j=0}^{r-1} (2n_1(j) * w_1(j) + w_1(j) * (w_1(j)-1))/2(r-j)} \text{pr}(\cdots n_1(j), w_1(j) \cdots). \tag{10}$$

If the sum in the exponent of (10) is less than $\kappa n/r$ (where κ is a constant) with probability bounded from below by $\frac{2}{3}$ then (10) is bounded from below by $\frac{2}{3}(1-(1/2r))^{2n\kappa}$ which approaches a constant as r approaches infinity if the limiting ratio of n to r is constant. To show that the sum in the exponent of (10) is less than $\kappa n/r$ with probability greater than $\frac{2}{3}$ we show that the expectation of the sum is bounded from above by $\kappa n/3r$ and apply Markov's inequality.

To show that the expectation of the sum in the exponent of (10) is less than $\kappa n/3r$ we need only show that the expectation of each term in the sum is less than $\kappa n/3r^2$. Denote by $p_1(j)$ the jth term in the sum. Then

$$E\{p_1(j)\} \leqq \frac{1}{2(r-j)} \left(E\{w_1^2(j)\} + \sum_{s=0}^{n} \sum_{t=0}^{n} 2 * s * t * \text{pr}(n_1(j) = t, w_1(j) = s) \right). \tag{11}$$

The second term within parentheses is bounded by $\gamma_1(1-(j/r))n/r$ for $j < r - r^{8/9}$ and by $\gamma_2(1-(j/r))n/r$ for $j \geqq r - r^{8/9}$ where γ_1 and γ_2 are constants greater than zero. Consider the first case, $1 \leqq j < r - r^{8/9}$. Suppose SC_1 is modified so that all literals not chosen from $C_1(j)$ are chosen randomly from the set of all unchosen literals and

suppose that $\hat{n}_2(j)$ and $\hat{w}_1(j)$ have the same meaning as $n_2(j)$ and $w_1(j)$ except applied to the modified SC_1. Define $n_l = E\{\hat{n}_2(j)\} - n^{3/4}$ and $n_u = E\{\hat{n}_2(j)\} + n^{3/4}$. It is easy to see that $\hat{n}_2(j)$ is binomially distributed with mean $E\{\hat{n}_2(j)\}$ proportional to $j/r(1-(j/r))^2 n$ so the probability that $n_l \leq \hat{n}_2(j) < n_u$ is greater than $1 - 2e^{-n^{3/2}/E\{\hat{n}_2(j)\}}$ from [4] and this is greater than $1 - e^{-\sqrt{n}}$ since $E\{\hat{n}_2(j)\} < n$. The double sum of (11) can be bounded from above by using $\hat{w}_1(j)$ for $w_1(j)$. We do so and split the result into three parts:

$$
\begin{aligned}
&\sum_{s=0}^{n}\sum_{t=0}^{n}\sum_{u=0}^{n_l} 2 * s * t * \operatorname{pr}(n_1(j)=t,\ \hat{n}_2(j)=u,\ \hat{w}_1(j)=s) \\
&\qquad + \sum_{s=0}^{n}\sum_{t=0}^{n}\sum_{u=n_l}^{n_u} 2 * s * t * \operatorname{pr}(n_1(j)=t,\ \hat{n}_2(j)=u,\ \hat{w}_1(j)=s) \\
&\qquad + \sum_{s=0}^{n}\sum_{t=0}^{n}\sum_{u=n_u}^{n} 2 * s * t * \operatorname{pr}(n_1(j)=t,\ \hat{n}_2(j)=u,\ \hat{w}_1(j)=s) \\
&\qquad < \frac{4n^2}{e^{\sqrt{n}}} + 2 * E\{\hat{w}_1(j)\} * E\{n_1(j)\}
\end{aligned} \tag{12}
$$

in the limit since $n/r < 2.9$ and $|n_u - n_l| \to 0$. But $E\{\hat{w}_1(j)\}$ may be shown to be proportional to $(1-(j/r))n/r$ by solving (8) with $\beta = 0$ and dividing by $r-j$. Also, $E\{n_1(j)\}$ is bounded by a constant for all $1 \leqq j \leqq r$ since $E\{w_1(j)\} < 1$ and at least one clause is removed from $C_1(j)$ if $C_1(j) \neq \phi$. So (12) is less than $\gamma_1(1-(j/r))n/r$ where γ_1 is a constant greater than zero. Now consider the case $r - r^{8/9} \leqq j < r$. In this range $E\{\hat{w}_1(j)\}$ is proportional to $(1-(j/r))n/r$ and is decreasing with increasing j. Clearly, in this range

$$
\begin{aligned}
&\sum_{s=0}^{n}\sum_{t=0}^{n} 2 * s * t * \operatorname{pr}(n_1(j)=t,\ w_1(j)=s) \\
&\qquad < 2 * E\{\hat{w}_1(j)\} * E\{n_1(r-r^{8/9})\} < \gamma_2\left(1-\frac{j}{r}\right)\frac{n}{r}
\end{aligned}
$$

as $r \to \infty$.

We now need to find a bound on $E\{w_1^2(j)\}$. Let $\hat{w}_1(j)$ be as before. Clearly, $E\{w_1^2(j)\} \leqq E\{\hat{w}_1^2(j)\}$. But $\hat{w}_1(j)$ is distributed binomially; hence $E\{\hat{w}_1^2(j)\} = \sigma^2(\hat{\omega}_1(j)) + (E\{\hat{w}_1(j)\})^2 < E\{\hat{w}_1(j)\} + (E\{\hat{w}_1(j)\})^2$ and

$$
E\{w_1^2(j)\} < \gamma_3 * \left(1-\frac{j}{r}\right)\frac{n}{r}.
$$

Let $\gamma = \max\{\gamma_1, \gamma_2\}$. Substituting $\gamma(1-(j/r))n/r$ for the double sum in (11) and then $\gamma_3 * (1-(j/r))n/r$ for $E\{w_1^2(j)\}$ in the resulting inequality gives

$$
E\{p_1(j)\} \leqq \left(\frac{\gamma_3+\gamma}{2}\right)\frac{n}{r^2} = \frac{\kappa}{3} * \frac{n}{r^2}.
$$

From this the expectation of the sum in the exponent of (10) is less than $\kappa n/3r$. By Markov's inequality the probability that the sum is greater than $\kappa n/r$ is less than $\frac{1}{3}$. Therefore, the probability that the sum is less than $\kappa n/r$ is greater than $\frac{2}{3}$. Thus (10) is greater than $\frac{2}{3}(1-(1/2r))^{2n\kappa}$ which approaches $\frac{2}{3}e^{-n\kappa/r}$ as r approaches infinity. Let $\varepsilon = \frac{2}{3}(1-(1/2r))^{2n\kappa}$.

The Unit-Clause heuristic and the maximum occurring literal heuristic have been incorporated into a Backtrack algorithm for 3-SAT and experiments run. This algorithm is not useful for solving instances of 2-SAT (two literals per clause) since, as is well

known, 2-SAT can be solved in polynomial time. The algorithm is

BA(I):
  If there exist two complementary unit clauses in I Then return *UNSAT*
  Else If $I = \phi$ Then return *SAT*
  Else If there is a unit clause in I Then Begin
    While there is a unit clause $\{l\}$ in I Do Begin
      Remove from I all clauses containing l
      Remove from I all occurrences of comp (l)
      End
    Return *BA*(I)
    End
  Else Begin
    Choose a variable v which is present in I
    If card ($\bar{v}$, C_3) > card (v, C_3) Then $l \leftarrow \bar{v}$ Else $l \leftarrow v$
    $I_1 = \{c: c \in I$ and l, comp (l) $\notin c$ or $c \cup \{l\} \in I\}$
    $I_2 = \{c: c \in I$ and l, comp (l) $\notin c$ or $c \cup \{\text{comp } (l)\} \in I\}$
    If *BA*(I_1) = *SAT* Then return *SAT*
    Else If *BA*(I_2) = *SAT* Then return *SAT* Else return *UNSAT*

Algorithm *BA* was run on random instances of 3-SAT generated according to $M(n, r, 3)$ with n/r set to 2.4, 2.6, 2.8, 3.0, 3.2, 3.4 and 3.6 for r ranging from 10 to 200 in steps of 10. At each data point the average number of calls to *BA* per instance was computed for 100 instances. The results are presented in Fig. 1. Note that for

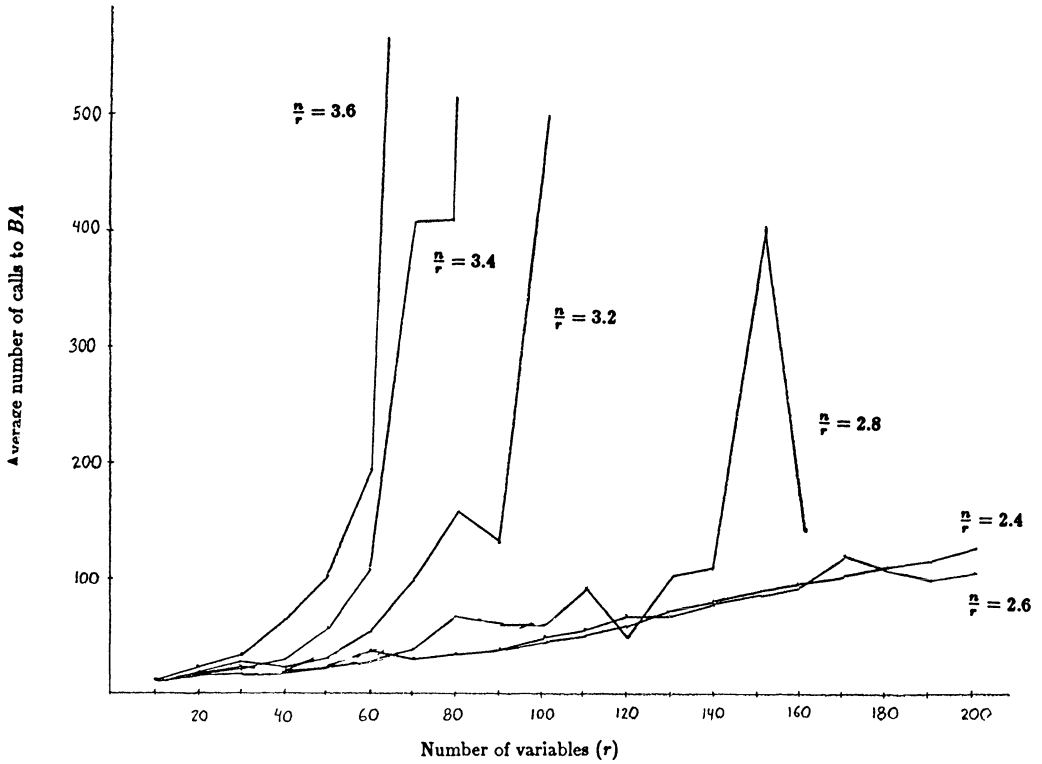


FIG. 1. *Average case performance of BA with n/r fixed.*

$n/r \leqq 2.6$ the performance curves are practically straight lines, for $n/r = 2.8$ there are occasional peaks and for $n/r \geqq 3.0$ the performance curves rise dramatically. Upon looking at the performance of individual instances for the case $n/r = 2.8$ it was noted that the peaks were due to a few runs that required many calls to *BA*.

5. A modification to SC_1. In this section we discuss why, in SC_1, if $C_1(j) = \varnothing$ then the $j+1$st chosen literal is chosen only on the number of occurrences of that literal and its complement in $C_3(j)$ and not in $C_2(j)$. Suppose that the $j+1$st literal is chosen on the number of times it occurs in $C_3(j)$ and $C_2(j)$ if $C_1(j) = \varnothing$. Assume the most optimistic case: the literal appears in more clauses of both $C_3(j)$ and $C_2(j)$ than its complement (then the "flow" into $C_1(j+1)$ is minimized since the number of two and three literal clauses removed due to the $j+1$st chosen literal is maximized). Let $E\{w_1^*(j)\}$ denote the new average "flow" of clauses into $C_1(j)$. Then

$$E\{w_1^*(j)\} = E\{w_1(j)\} - H_1(j)(1 - E\{w_1^*(j)\})$$

where $H_1(j)$ is the extra number of clauses removed from the "flow" into $C_1(j)$ when the chosen literal is not a unit clause and $1 - E\{w_1^*(j)\}$ is the probability (to within $O(1/r)$) that the chosen literal is not a unit clause. So

$$E\{w_1^*(j)\} = \frac{E\{w_1(j)\} - H_1(j)}{1 - H_1(j)}.$$

Thus $E\{w_1^*(j)\} < 1$ is equivalent to $E\{w_1(j)\} < 1$ and no benefit is gained by considering the number of occurrences of the chosen literal in $C_2(j)$.

6. Conclusions. We have presented an algorithm for 3-SAT based on the Unit-Clause and maximum occurring literal heuristics and have shown that this algorithm finds a solution to a random instance of 3-SAT in polynomial time with probability bounded from below by a constant under $M(n, r, 3)$ when $\lim_{n,r\to\infty}(n/r) < 2.9$. Experiments indicate that a Backtrack algorithm containing these two heuristics performs extremely well probabilistically over the same range of values of the limiting ratio of n/r. The method used to get these results has the advantages of providing intuition and being general enough to be used on other algorithms for 3-SAT and other NP-complete problems. The method can be used to show that the Unit-Clause heuristic alone finds a solution to a random instance of 3-SAT in polynomial time with probability bounded from below by a constant under $M(n, r, 3)$ when $\lim_{n,r\to\infty}(n/r) < 2.66$: the analysis is the same as presented here except that $\beta = 0$.

The results are interesting because they may be compared with similar results obtained for other 3-SAT algorithms under the same model (see the introduction) and they indicate the degree to which solutions to random instances of 3-SAT are "clumped."

REFERENCES

[1] C. A. BROWN AND P. W. PURDOM, *An average time analysis of backtracking*, this Journal, 10 (1981), pp. 583-593.

[2] M. T. CHAO, *Probabilistic analysis and performance measurement of algorithms for the satisfiability problem*, Ph.D. dissertation, Case Western Reserve University, Cleveland, OH, 1984.

[3] M. DAVIS AND H. PUTNAM, *A computing procedure for quantification theory*, J. Assoc. Comput. Mach., 7 (1960), pp. 201-215.

[4] P. ERDÖS AND J. SPENCER, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.

[5] J. FRANCO, *Probabilistic analysis of the pure literal heuristic for the satisfiability problem*, Ann. Oper. Res., 1 (1984), pp. 273-289.

[6] J. FRANCO, *Sensitivity of probabilistic results on algorithms for* NP*-complete problems to input distributions*, Inform. Process. Lett., to appear.

[7] J. FRANCO AND M. PAULL, *Probabilistic analysis of the Davis-Putnam Procedure for solving the satisfiability problem*, Discrete Appl. Math., 5 (1983), pp. 77-87.

[8] A. GOLDBERG, *Average case complexity of the satisfiability problem*, Proc. 4th Workshop on Automated Deduction, 1979, pp. 1-6.

[9] A. GOLDBERG, P. W. PURDOM AND C. A. BROWN, *Average time analysis of simplified Davis-Putnam procedures*, Inform. Process. Lett., 15 (1982), pp. 72-75.

[10] P. W. PURDOM, *Search rearrangement backtracking and polynomial average time*, Artificial Intelligence, 21 (1983), pp. 117-133.

[11] P. W. PURDOM AND C. A. BROWN, *The pure literal rule and polynomial average time*, this Journal, 14 (1985), pp. 943-953.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
019

WORST CASE BOUND OF AN LRF SCHEDULE FOR THE MEAN WEIGHTED FLOW-TIME PROBLEM*

TSUYOSHI KAWAGUCHI† AND SEIKI KYAN†

Abstract. This paper studies the problem of scheduling a set of n independent tasks on m identical processors so as to minimize mean weighted flow-time. The problem is known to be NP-complete for $m \geqq 2$ and to be NP-complete in the strong sense for m arbitrary.

The worst case behavior of a heuristic algorithm which requires time $O(n \log n)$ is investigated, and it is shown that the mean weighted flow-time obtained by the algorithm does not exceed $(\sqrt{2}+1)/2 \cong 1.207$ times that of an optimal schedule. Moreover the bound $(\sqrt{2}+1)/2$ is best possible.

Key words. independent tasks, identical processors, mean weighted flow-time, LRF schedule, worst case behavior

AMS(MOS) subject classifications. 68C25, 68C15, 90B35

1. Introduction. The MWFT (*Mean Weighted Flow-time*) *problem* is stated as follows [2]. A set of n tasks is to be processed on m identical processors. Tasks $i(i=1, \cdots, n)$ require processing times t_i and have positive weights w_i. The tasks are independent. In other words, there exists no precedence relation among them. The objective is to find a schedule which minimizes the cost $\Sigma\, w_i f_i$, where f_i is the finishing time of task i.

When $m=1$, the problem can be solved using the algorithm of Smith in which tasks are sequenced in nonincreasing order of w_i/t_i [7]. However, the MWFT problem is known to be NP-complete for $m \geqq 2$ and to be NP-complete in the strong sense for m arbitrary [4]. An approximation algorithm has been proposed for the problem of $m \geqq 2$ by Sahni [6]. However this algorithm is efficient only for small m because the algorithm requires time $O(n(n^2/\varepsilon)^{m-1})$ to find an approximate solution whose cost does not exceed $1+\varepsilon$ times that of an optimal schedule for the problem of $m \geqq 3$. Moreover, Chandra and Wong [1] have studied the problem of minimizing $\Sigma\, c_j^2$ where $c_j(j=1, \cdots, m)$ denote finishing times of processors j. Their result implies that if tasks have the same w_i/t_i, then $\Sigma\, w_i f_i$ of processing tasks in nonincreasing order of their processing times does not exceed 1.04 times that of an optimal schedule.

Our paper investigates the worst case behavior of a heuristic algorithm which finds a feasible schedule in time $O(n \log n)$ for an arbitrary problem instance of the MWFT problem.

DEFINITION. An LRF (*Largest Ratio First*) *schedule* is a schedule obtained by the following algorithm which will be called the LRF *heuristic* in this paper.

(i) A priority list is constructed, where task i precedes task j if $w_i/t_i > w_j/t_j$, and precedence relations among tasks with the same w_i/t_i are arbitrary.

(ii) During execution, whenever a processor becomes free for assignment, the list is scanned for the first unexecuted task, which is then assigned to the processor.

DEFINITION. Since the LRF heuristic imposes no priority rule among tasks with the same w_i/t_i, we may have more than one LRF schedule for a problem instance. An LRF schedule maximizing the cost $\Sigma\, w_i f_i$ will be called a *maximal* LRF *schedule* in this paper.

* Received by the editors December 12, 1982, and in revised form December 16, 1985.

† Department of Electronic Engineering and Computer Science, University of the Ryukyus, Nishihara, Okinawa, 903-01 Japan.

Let M_L and M^* denote the cost of a maximal LRF schedule and that of an optimal schedule respectively. Then Eastman et al. [3] have shown that $M_L < M_1/m + (m-1)\sum w_i t_i/m$ and $M^* \geqq M_1/m + (m-1)\sum w_i t_i/2m$ where M_1 denotes the cost of processing tasks in nonincreasing order of their ratios w_i/t_i on a single processor. These bounds are useful for evaluating the value of M_L/M^* for an individual problem instance. However as for an upper bound of M_L/M^*, these bounds only show that $M_L/M^* < 3/2$.

Our paper shows that an arbitrary problem instance of the MWFT problem satisfies

$$M_L/M^* \leqq (\sqrt{2}+1)/2 \cong 1.207. \tag{1}$$

Moreover a problem instance is given for which the ratio of M_L to M^* attains to $(\sqrt{2}+1)/2$, and in which all tasks have distinct values of w_i/t_i. Thus the worst case bound on the LRF heuristic never takes a smaller value than $(\sqrt{2}+1)/2$ even if any priority rule is imposed among tasks with the same w_i/t_i.

First it is shown in § 2 that (1) holds for tasks with the same w_i/t_i. Next using the induction on the number of distinct values of ratios w_i/t_i, § 3 proves that (1) holds for an arbitrary problem instance of the MWFT problem.

2. Bound of M_L/M^* for tasks with the same w_i/t_i. If tasks have the same w_i/t_i, then an LRF schedule is a schedule obtained by applying step (ii) of the LRF heuristic to an arbitrary list of tasks. Therefore for convenience sake, an LRF schedule and a maximal LRF schedule for tasks with the same w_i/t_i will be called *any list schedule* and a *maximal schedule* respectively in this section. The main result of this section is as follows: *if tasks have the same w_i/t_i, then*

$$M_L/M^* \leqq (\sqrt{2}+1)/2, \tag{2}$$

where M_L and M^* denote the cost $\sum w_i f_i$ of a maximal schedule and that of an optimal schedule respectively.

We shall use the following assumption and notations in this section.

ASSUMPTION. Tasks have the same w_i/t_i and are numbered in nonincreasing order of their processing times, that is,

$$t_1 \geqq \cdots \geqq t_n. \tag{3}$$

Notation

$\lfloor \alpha \rfloor$: the greatest integer $\leqq \alpha$,
t_i: processing time of task i,
w_i: weight of task i,
$T \triangleq \sum_{i=1}^{n} t_i^2$,
f_i: finishing time of task i in some schedule,
c_j: finishing time of processor j in some schedule,
M_L: cost $\sum_{i=1}^{n} w_i f_i$ of a maximal schedule,
M^*: cost $\sum_{i=1}^{n} w_i f_i$ of an optimal schedule,
C_L: cost $\sum_{j=1}^{m} c_j^2$ of a maximal schedule,
C^*: cost $\sum_{j=1}^{m} c_j^2$ of an optimal schedule.

(As will be shown in Proposition 2, an optimal (maximal) schedule of $\sum w_i f_i$ minimizes (maximizes) $\sum c_j^2$ if tasks have the same w_i/t_i.)

In the proof of (2), it is necessary to divide a set of tasks into two subsets according to their processing times. The succeeding notation is used for this reason.

$B_i \triangleq \sum_{k=i+1}^{n} t_k/(m-i) \ (0 \leqq i \leqq m-1)$,
$p \ (0 \leqq p \leqq m-1)$: the smallest index i such that $t_{i+1} \leqq B_i$,

$v \triangleq p/m (0 \leqq v < 1)$,

$z \triangleq \begin{cases} 0 & \text{if } p = 0, \\ \sum_{i=1}^{p} t_i / \sum_{i=p+1}^{n} t_i & \text{if } p > 0, \end{cases}$

$x_j (1 \leqq j \leqq m)$: total processing times of tasks which are included in the last $n-p$ tasks and are processed on the jth processor in some schedule.

2.1. Preliminary results. We give some preliminary results needed in order to prove (2).

PROPOSITION 1. *An optimal schedule for the* MWFT *problem is included in the set of list schedules, that is, permutation schedules.*

Proof. It is clear from [5, Thm. 4-1]. □

PROPOSITION 2. *If $w_i/t_i = \alpha$ for all tasks, then*

$$\sum_{i=1}^{n} w_i f_i = \alpha \left(\sum_{j=1}^{m} c_j^2 + T \right) \Big/ 2 \tag{4}$$

in any list schedule [3, p. 270].

The above proposition implies that an optimal (maximal) schedule of $\Sigma w_i f_i$ minimizes (maximizes) Σc_j^2 if tasks have the same w_i/t_i.

PROPOSITION 3. *If $p > 0$ then*

$$t_p > B_p \quad \textit{and} \quad z > v/(1-v). \tag{5}$$

Proof. First $t_p > B_{p-1}$ from the definition of p. Moreover since $(m-p+1)(m-p) \times (B_{p-1} - B_p) = (m-p)t_p - \sum_{i=p+1}^{n} t_i = (m-p+1)(t_p - B_{p-1}) > 0$, we have $B_{p-1} > B_p$ and so $t_p > B_p$. Furthermore this result, together with the definition of v and z yields the second inequality of (5). □

PROPOSITION 4. *Each of the first p tasks is assigned to the end of a processor in any list schedule.*

Proof. Let k denote the smallest index of tasks that violate the proposition in some list schedule S. Then we can assume without loss of generality that each i of the first $k-1$ tasks is assigned to the end of the ith processor respectively and task k is processed on the kth processor. Since this assumption implies $\sum_{j=k}^{m} c_j \leqq (m-k+1)B_{k-1} < (m-k+1)t_k$ and $c_k > t_k$, there exists a processor which has a smaller finishing time than t_k. Therefore each task starts at an earlier time than t_k because S is a list schedule. This contradicts our first assumption that task k is not assigned to the end of a processor in S. □

PROPOSITION 5. *Each of the first p tasks occupies a processor by itself in an optimal schedule S^*.*

Proof. By Proposition 4, we can assume without loss of generality that each i of the first p tasks is processed on the ith processor, respectively. Also let k denote the smallest index of tasks that violate the proposition in S^*. That is, $x_1 = \cdots = x_{k-1} = 0$ and $x_k > 0$. Since these assumptions imply $\sum_{j=k}^{m} c_j = (m-k+1)B_{k-1} < (m-k+1)t_k$ and $c_k > t_k$, there exists a processor which has a smaller finishing time than t_k. Assume without loss of generality that the mth processor is such a processor. If all tasks but task k processed on the kth processor are removed to the mth processor, then the cost C^* decreases because $(t_k + x_k)^2 + c_m^2 > t_k^2 + (x_k + c_m)^2$ by $t_k > c_m$ and $x_k > 0$. This contradicts that S^* is an optimal schedule, and so the proposition holds. □

PROPOSITION 6. *Given nonnegative numbers $c_1, \cdots, c_m, c_i', c_j'$ such that $c_i + c_j = c_i' + c_j'$ and $|c_i - c_j| \geqq |c_i' - c_j'|$, we have*

$$c_1^2 + \cdots + c_i^2 + \cdots + c_j^2 + \cdots + c_m^2 \geqq c_1^2 + \cdots + (c_i')^2 + \cdots + (c_j')^2 + \cdots + c_m^2.$$

This proposition has been shown in [1, Lemma 1].

2.2. Bound of M_L/M^*. We have the following theorem for a problem instance in which tasks have the same w_i/t_i.

THEOREM 1. *For a problem instance with $v=0$ and $z=0$,*

$$M_L/M^*-1 \leqq \text{maximum of } H_1(a) \quad \text{subject to } 0 \leqq a \leqq 1 \tag{6}$$

where

$$H_1(a) \triangleq (-a^2+a)/(2-a). \tag{7}$$

For a problem instance with $v>0$ and $z>v/(1-v)$,

$$\begin{aligned} &M_L/M^*-1 \leqq 1/5 \quad \text{or} \\ &M_L/M^*-1 \leqq \text{maximum of } H_2(a) \quad \text{subject to } 0 \leqq a \leqq 1 \end{aligned} \tag{8}$$

where

$$H_2(a) \triangleq [-a^2+a\{2z+2-1/(1-v)\}]\{2z^2/v+(2-a)/(1-v)\}^{-1}. \tag{9}$$

Moreover this theorem yields the following corollary.

COROLLARY 1. *For any problem instance in which tasks have the same w_i/t_i,*

$$M_L/M^* \leqq (\sqrt{2}+1)/2. \tag{10}$$

Specially for a problem instance with $v=0$,

$$M_L/M^* \leqq 4-2\sqrt{2}. \tag{11}$$

Moreover these bounds are best possible.

Firstly, Theorem 1 is proved in the next subsection. Secondly (10) and (11) are derived from Theorem 1 in § 2.4. Before proceeding with these proofs, we give two problem instances showing that (10) and (11) are best possible.

An instance with $M_L/M^*=(\sqrt{2}+1)/2$.

$m = m^* + \lfloor (1+\sqrt{2})m^* \rfloor$,

$n = mn^* + m^*$,

$t_i = w_i = 1/n^*$ for $1 \leqq i \leqq mn^*$,

$t_i = w_i = 1+\sqrt{2}$ for $mn^*+1 \leqq i \leqq mn^*+m^*$,

where m^* denotes some integer, and n^* is such a integer that can be divided by $\lfloor (1+\sqrt{2})m^* \rfloor$.

Let M_L be the cost of a list schedule in which the first mn^* tasks are earliest processed. Also let M^* denote the cost of a list schedule in which the last m^* tasks are earliest processed. Then

$$\begin{aligned} M_L &= (1+\sqrt{2})(2+\sqrt{2})m^* + (m/2)(1+1/n^*), \\ M^* &= (1+\sqrt{2})^2 m^* + (m/2)\{m/\lfloor (1+\sqrt{2})m^* \rfloor + 1/n^*\}. \end{aligned}$$

Therefore the ratio of M_L to M^* tends to $(1+\sqrt{2})/2$ as $m^* \to \infty$ and $n^* \to \infty$.

An instance with $v=0$ and $M_L/M^*=4-2\sqrt{2}$.

$m = m^* + \lfloor \sqrt{2}\, m^* \rfloor$,

$n = mn^* + m^*$,

$t_i = w_i = 1/n^*$ for $1 \leqq i \leqq mn^*$,

$t_i = w_i = (2+\sqrt{2})/2$ for $mn^*+1 \leqq i \leqq mn^*+m^*$,

where m^* denotes some integer, and n^* is such a integer that can be divided by $\lfloor \sqrt{2}\, m^* \rfloor$.

First we have $v=0$ for this instance because $B_0 \triangleq \sum_{i=1}^{n} t_i/m \geqq (2+\sqrt{2})/2$. Moreover let M_L be the cost of a list schedule in which the first mn^* tasks are earliest processed. Also let M^* denote the cost of a list schedule in which the last m^* tasks are earliest processed. Then

$$M_L = (5+3\sqrt{2})m^*/2 + (m/2)(1+1/n^*),$$

$$M^* = (3+2\sqrt{2})m^*/2 + (m/2)(m/\lfloor\sqrt{2}\, m^*\rfloor + 1/n^*).$$

Therefore the ratio of M_L to M^* tends to $4-2\sqrt{2}$ as $m^* \to \infty$ and $n^* \to \infty$.

2.3. Proof of Theorem 1. In addition to (3), we make the following assumptions without loss of generality in the proof of Theorem 1.

(I) Processing times of tasks are normalized by $\sum_{i=p+1}^{n} t_i/m$, that is, $\sum_{i=p+1}^{n} t_i = m$.

(II) Each i of the first p tasks is assigned to the end of the ith processor respectively in any list schedule.

(III) $x_{p+1} \geqq \cdots \geqq x_m$.

It is clear that (I) has no effect on the ratio M_L/M^*. Moreover validity of (II) and (III) is guaranteed by Proposition 4.

LEMMA 1. *If tasks have the same w_i/t_i, then*

$$M_L/M^* = 1 + (C_L - C^*)/(C^* + T). \tag{12}$$

Proof. This lemma is easily derived from Propositions 1 and 2. □

LEMMA 2. *For a problem instance with fixed m, p and $(t_1, \cdots, t_p)$,*

$$C^* \geqq C_0 \triangleq \sum_{j=1}^{p} t_j^2 + m^2/(m-p). \tag{13}$$

Proof. Proposition 5, together with Cauchy's inequality, yields (13). □

LEMMA 3. *The following relations hold among x_j in any list schedule.*

$$\sum_{j=1}^{m} x_j = m, \tag{14}$$

$$x_{p+1} \geqq \cdots \geqq x_m, \tag{15}$$

$$0 \leqq x_j \leqq x_m \quad \text{for } 1 \leqq j \leqq p, \tag{16}$$

$$0 \leqq x_j - x_m \leqq B_p = m/(m-p) \quad \text{for } p+1 \leqq j \leqq m. \tag{17}$$

Proof. No task starts at a later time than x_m in any list schedule. Moreover by the definition of p, none of the last $n-p$ tasks have larger processing times than B_p. Therefore the assumptions (I) to (III) lead to the lemma. □

The following lemma gives an upper bound of C_L. Note that $\sum_{j=p+1}^{n} t_j^2 < (m-p)B_p^2$ by using $\sum_{j=p+1}^{n} t_j = m$ and $t_j \leqq B_p (p+1 \leqq j \leqq n)$.

LEMMA 4. *If a problem instance with fixed m, p and $(t_1, \cdots, t_p)$ satisfies*

$$\sum_{j=p+1}^{n} t_j^2 \leqq A \quad \text{for a real number } A \text{ such that } 0 < A < (m-p)B_p^2, \tag{18}$$

then the problem instance satisfies

$$C_L \leqq C_I \quad \text{or} \quad C_L \leqq C_{II}, \tag{19}$$

where C_I and C_{II} are defined as follows:

(i) $C_I \triangleq$ *maximum of* $C(\mathbf{x})$ *subject to* (14) *and*

$$x_j = \begin{cases} x_m & (1 \leqq j \leqq p,\ p+r+2 \leqq j \leqq m), \\ x_m + B_p & (p+1 \leqq j \leqq p+r), \\ x_m + \varepsilon & (j = p+r+1), \end{cases} \tag{20}$$

where

$$C(\mathbf{x}) \triangleq \sum_{j=1}^{p} (t_j + x_j)^2 + \sum_{j=p+1}^{m} x_j^2, \tag{21}$$

$$r\text{: integer and } 0 \leqq r \leqq \lfloor A/B_p^2 \rfloor, \tag{22}$$

$$\varepsilon = \begin{cases} (A - B_p^2 \lfloor A/B_p^2 \rfloor)^{1/2} & \text{or } 0 \text{ if } r = \lfloor A/B_p^2 \rfloor, \\ B_p & \text{or } 0 \text{ otherwise.} \end{cases} \tag{23}$$

(ii) $C_{II} \triangleq$ *maximum of* $C(\mathbf{x})$ *subject to* (14) *and*

$$x_j = \begin{cases} x_m & (1 \leqq j \leqq q,\ p+1 \leqq j \leqq m), \\ 0 & (q+1 \leqq j \leqq p), \end{cases} \tag{24}$$

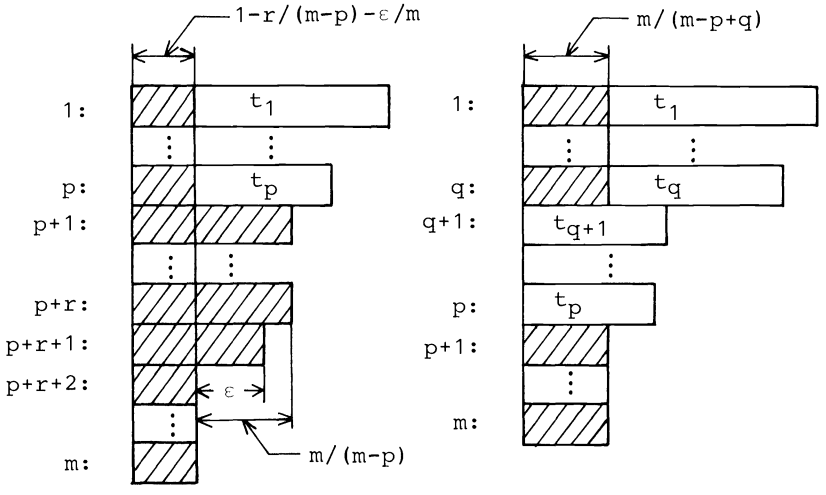
where q *is a integer such that* $0 \leqq q \leqq p$.

(Figure 1 illustrates the relations among x_j shown in (20) and (24). The hatched parts represent x_j and we should notice that $B_p = m/(m-p)$.)

Proof. It is clear from Proposition 4 that $\sum_{j=1}^{m} c_j^2$ is given by $C(\mathbf{x})$. Therefore we shall prove that the maximum of $C(\mathbf{x})$ subject to (14)-(18) is given by x_j which satisfy (20) or (24). Let $P^+ \triangleq \{x_1, \cdots, x_p\}$ and $P^- \triangleq \{x_{p+1}, \cdots, x_m\}$. First using Proposition 6, we have

$$(\#1) \qquad x_m \geqq x_1 \geqq \cdots \geqq x_p.$$

Hence it suffices to consider the following two cases.



(a) x_j given by (20). (b) x_j given by (24).

FIG. 1

Case 1. $x_j = x_m$ for all x_j of P^+. Then Proposition 6 leads to:

(#2) No more than one x_j of P^- satisfies $0 < x_j - x_m < B_p$. (Namely the remainings of P^- are equal to x_m or $x_m + B_p$.)

Therefore it follows from (18) that (20) holds among x_j if r satisfies (22) and ε is given by:

$$0 \leqq \varepsilon \leqq (A - B_p^2 \lfloor A/B_p^2 \rfloor)^{1/2} \quad \text{for } r = \lfloor A/B_p^2 \rfloor,$$

$$0 \leqq \varepsilon \leqq B_p \quad \text{for the others.}$$

Then $C(\mathbf{x})$ is a quadratic function as to ε. Thus the maximum of $C(\mathbf{x})$ is obtained by ε which satisfies (23).

Case 2. $x_j < x_m$ for some x_j of P^+. Then Proposition 6 leads to:

(#3) $x_{j+1} = x_{j+2} = \cdots = x_p = 0$ and $x_{p+1} = x_{p+2} = \cdots = x_m$.

Assume $x_j = \varepsilon (0 < \varepsilon < x_m)$ for some x_j of P^+. Then $C(\mathbf{x})$ is a quadratic function as to ε, and it follows that the maximum of $C(\mathbf{x})$ is given by $\varepsilon = 0$ or $\varepsilon = x_m$. This is a contradiction, and so (24) holds. □

In the remainder of this subsection, Theorem 1 is derived for each case of $C_L \leqq C_I$ and $C_L \leqq C_{II}$. We should notice that $C_L \leqq C_I$ always holds for a problem instance with $v = 0$. Because, for such an instance, Proposition 6 directly leads to (#2) without passing through (#1) in the proof of Lemma 4.

2.3.1. $C_L \leqq C_I$. It will be shown that

$$\frac{C_I - C_0}{C_0 + T} \leqq \begin{cases} \text{maximum of } H_1(a) \text{ subject to } 0 \leqq a \leqq 1 & \text{if } v = 0, \\ \text{maximum of } H_2(a) \text{ subject to } 0 \leqq a \leqq 1 & \text{otherwise,} \end{cases} \tag{25}$$

where $H_1(a)$ and $H_2(a)$ are given by (7) and (9), respectively.

The above bound, together with Lemmas 1, 2 and 4 yield Theorem 1 in this case. Now let us prove (25). First we have

$$C_I = \sum_{j=1}^{p} (t_j + a)^2 + \sum_{j=p+1}^{p+r} (a + B_p)^2 + (a + \varepsilon)^2 + (m - p - r - 1)a^2,$$

where $a \triangleq 1 - \varepsilon/m - r/(m-p)$.

Moreover (22) and (23) imply $A \geqq rB_p^2 + \varepsilon^2$. Therefore,

the least upper bound of $(C_I - C_0)/(C_0 + \sum_{j=1}^{p} t_j^2 + A)$ subject to $0 < A < (m-p)B_p^2$, (22) and (23)

does not exceed

the maximum of $(C_I - C_0)/(C_0 + \sum_{i=1}^{p} t_j^2 + rB_p^2 + \varepsilon^2)$ subject to $0 \leqq r \leqq m - p$ and $0 \leqq \varepsilon \leqq B_p$.

Furthermore for fixed $m, p, \sum_{j=1}^{p} t_j$ and r, the maximum of $(C_I - C_0)/(C_0 + \sum_{j=1}^{p} t_j^2 + rB_p^2 + \varepsilon^2)$ is obtained by

$$\varepsilon = 0 \quad \text{and} \quad t_1 = \cdots = t_p.$$

Thus we have (25).

2.3.2. $C_L \leqq C_{II}$. As previously stated, a problem instance with $v = 0$ satisfies $C_L \leqq C_I$. Hence assume $v > 0$ in this case. The following lemma, together with Lemma 1, 2 and 4 yields Theorem 1.

LEMMA 5. *For* $v>0$,

$$\frac{C_{II}-C_0}{C_0+T}\leqq\begin{cases}1/5 & \text{if } z>(1+v)/\{2(1-v)\},\\ H_2(1) & \text{otherwise,}\end{cases}$$

where $H_2(1)$ *is given by substituting* $a=1$ *into* (9).

This lemma is proved by the following three claims.

CLAIM 1. *If* $v/(1-v)<z\leqq(1+v)/\{2(1-v)\}$, *then*

$$C_{II}/C_0-1\leqq G\triangleq\{2z-v/(1-v)\}\{z^2/v+1/(1-v)\}^{-1}. \tag{26}$$

Proof. First we have

$$C_{II}=C_0+m(m-p+q)^{-1}\left\{2\sum_{j=1}^{q}t_j-qm(m-p)^{-1}\right\}.$$

It is easily verified that a problem instance maximizing C_{II}/C_0 satisfies

$$t_1=\cdots=t_q \quad\text{and}\quad t_{q+1}=\cdots=t_p.$$

Let $s\triangleq q/m$. Then we have $st_1+(v-s)t_p=z$ and $C_{II}/C_0=1+N/D$ where $N\triangleq\{2st_1-s/(1-v)\}/(1-v+s)$ and $D\triangleq st_1^2+(v-s)t_p^2+1/(1-v)$. Since $t_p>1/(1-v)\geqq z+1/2$ by virtue of Proposition 3 and $z\leqq(1+v)/\{2(1-v)\}$, we have $2t_p>2z+1$. Therefore $st_1+(v-s)t_p=z$ yields $2z-2st_1>(v-s)(2z+1)$, and so $(2z+1)(1-v+s)>2st_1+1$, and it follows that $N<2z-v/(1-v)$. On the other hand, $D\geqq z^2/v+1/(1-v)$ because $st_1^2+(v-s)t_p^2\geqq\{st_1+(v-s)t_p\}^2/v=z^2/v$. Thus the claim holds. □

CLAIM 2. $C_{II}/C_0-1\leqq\frac{1}{3}$.

Proof. Let I^* be a problem instance maximizing C_{II}/C_0. First it will be shown that $q=p$ for I^*. Suppose $q<p$ for I^*. Let J^* and m^* denote a set of tasks and the number of processors, respectively, in I^*, and I' be another problem instance in which a set of tasks is $J^*-\{q+1,\cdots,p\}$ and the number of processors is $m^*-(p-q)$. Then $C_{II}(I')/C_0(I')=\{C_{II}(I^*)-(p-q)t_p^2\}/\{C_0(I^*)-(p-q)t_p^2\}>C_{II}(I^*)/C_0(I^*)$ which contradicts that I^* maximizes the ratio C_{II}/C_0. Thus we have $q=p$ for I^*, and it follows that (26) also holds for I^*. Moreover $G\leqq\frac{1}{3}$ for any v and z because $\{z^2/v+1/(1-v)\}-3\{2z-v/(1-v)\}=(z-3v)^2/v+(3v-1)^2/(1-v)\geqq 0$. Therefore the claim holds. □

CLAIM 3. *For* $v>0$, $C_0/T\leqq 1+v/\{(1-v)z^2\}$.

Proof. This is easily derived from (13), $T\geqq\sum_{j=1}^{p}t_j^2$ and $\sum_{j=1}^{p}t_j^2\geqq(mz)^2/p$. □

If $z>(1+v)/\{2(1-v)\}$ then $z^2>2v/(1-v)$, and so Claim 3 yields $(1+T/C_0)^{-1}<3/5$. This, together with Claim 2 leads to the first inequality of Lemma 5. Otherwise using $(C_{II}-C_0)/(C_0+T)=(C_{II}/C_0-1)(1+T/C_0)^{-1}$, Claim 1 and 3 yield the second inequality of Lemma 5.

2.4. Derivation of Corollary 1 from Theorem 1. Since the maximum of $H_1(a)$ subject to $0\leqq a\leqq 1$ is $3-2\sqrt{2}$, we have (11) for a problem instance with $v=0$. Therefore it remains to show that $H_2(a)$ does not exceed $(\sqrt{2}-1)/2$ if $0<v<1$, $z>v/(1-v)$ and $0\leqq a\leqq 1$. Let $g(v,z,a)\triangleq a^2-\{2z+2+(-3+\sqrt{2})/(2(1-v))\}a+(\sqrt{2}-1)\times\{z^2/v+1/(1-v)\}$. Then $g(v,z,a)\geqq 0$ implies $H_2(a)\leqq(\sqrt{2}-1)/2$. First we have $g(v,z,1)\geqq 0$ because $g(v,z,1)$ is minimized on $z=(\sqrt{2}+1)v$ for any v and $g(v,z,1)=(\sqrt{2}+1)\{v-(2-\sqrt{2})/2\}^2/(1-v)\geqq 0$ on this curve. Partially differentiating g with respect to a, $\partial g/\partial a=2a-\{2z+2+(-3+\sqrt{2})/(2(1-v))\}$. Consider two cases.

Case 1. $z\geqq(3-\sqrt{2})/\{4(1-v)\}$. Then $0\leqq a\leqq 1$ implies $\partial g/\partial a\leqq 0$. Therefore g is a monotonically decreasing function of a, and so $g(v,z,a)\geqq g(v,z,1)\geqq 0$.

Case 2. $z<(3-\sqrt{2})/\{4(1-v)\}$. Then we can assume $v<(3-\sqrt{2})/4$ because otherwise $z\geqq(3-\sqrt{2})/\{4(1-v)\}$ by $z>v/(1-v)$. Moreover $g(v, z, a)$ is minimized on $a=a^*(v, z)\triangleq z+1+(-3+\sqrt{2})/\{4(1-v)\}$. Let $h(v, z)\triangleq g(v, z, a^*(v, z))$. Partially differentiating h with respect to z, $\partial h/\partial z=2(\sqrt{2}-1-v)z/v-(1+\sqrt{2}-4v)/\{2(1-v)\}$. Consider two subcases.

Case 2-1. $(\sqrt{2}-1)/2\leqq v<(3-\sqrt{2})/4$. Then $\partial h/\partial z<(4v^2-4v+4\sqrt{2}-5)/\{2(1-v)v\}\leqq 0$ by $z<(3-\sqrt{2})/\{4(1-v)\}$. Thus h is a monotonically decreasing function of z. Since $a^*(v, z)=1$ on $z=(3-\sqrt{2})/\{4(1-v)\}$, we have $h\geqq g(v, z, 1)\geqq 0$.

Case 2-2. $0<v<(\sqrt{2}-1)/2$. Then h is minimized on $z=v(1+\sqrt{2}-4v)/\{4(1-v)(\sqrt{2}-1-v)\}$, and $h=(3-2\sqrt{2})\{(9-5\sqrt{2})/8-v\}/\{2(1-v)^2(\sqrt{2}-1-v)\}\geqq 0$ on this curve. This completes the proof of $g(v, z, a)\geqq 0$. Thus (10) is obtained.

3. Worst case bound of an LRF schedule. In this section we prove the following theorem.

THEOREM 2. *Let M_L and M^* denote the cost $\sum w_i f_i$ of a maximal* LRF *schedule and that of an optimal schedule respectively. Then*

$$M_L/M^*\leqq(\sqrt{2}+1)/2 \tag{27}$$

for an arbitrary problem instance of the MWFT *problem.*

Moreover there exists a problem instance which satisfies $M_L/M^*>(\sqrt{2}+1)/2-\delta$ for any $\delta>0$, and in which all tasks have distinct values of w_i/t_i.

The second half of Theorem 2 states that the worst case ratio of M_L/M^* never takes a smaller value than $(\sqrt{2}+1)/2$ even if any priority rule is imposed among tasks with the same w_i/t_i. The following theorem, together with an instance given in Corollary 1 proves the second half of Theorem 2.

THEOREM 3. *Let λ denote the ratio M_L/M^* for a problem instance in which tasks have the same w_i/t_i. Then there exists another problem instance which satisfies $M_L/M^*>\lambda-\delta$ for any $\delta>0$, and in which all tasks have distinct values of w_i/t_i.*

3.1. Proof of Theorem 2. We shall prove (27) by the induction on the number of distinct values of ratios w_i/t_i. The following notation is used in the proof of (27).

$\lambda\triangleq(\sqrt{2}+1)/2$,

$M(S)$: mean weighted flow-time of some schedule S,

$J\triangleq\{T_1, \cdots, T_n\}$: a set of n tasks which have t_i and w_i as processing times and weights, respectively, where $w_1/t_1\geqq\cdots\geqq w_n/t_n$,

$r_i\triangleq w_i/t_i$,

ν: the number of distinct values of ratios r_i,

q: the number of T_i whose ratios are equal to r_1,

$J'\triangleq\{T_1', \cdots, T_n'\}$: a set of n tasks whose processing times are t_i and whose weights are given by

$$w_i'=\begin{cases}\varepsilon w_i & \text{for } 1\leqq i\leqq q,\\ w_i & \text{for } q+1\leqq i\leqq n,\end{cases}$$

where $\varepsilon\triangleq r_{q+1}/r_q<1$,

$r_i'\triangleq w_i'/t_i$,

S_L: an LRF schedule for J,

S^*: an optimal schedule for J,

S'_L: a schedule for J' in which T'_i is put in the position of T_i determined by S_L, respectively,

$S^{*\prime}$: a schedule for J' in which T'_i is put in the position of T_i determined by S^*, respectively,

x: the part of $M(S_L)$ comprised of costs of the first q tasks,

y: the part of $M(S_L)$ comprised of costs of the last $n-q$ tasks,

x^*: the part of $M(S^*)$ comprised of costs of the first q tasks,

y^*: the part of $M(S^*)$ comprised of costs of the last $n-q$ tasks.

Since $r_1 = \cdots = r_q > r_{q+1}$ and $r'_1 = \cdots = r'_q = r'_{q+1}$, the number of distinct values of ratios r'_i is $\nu - 1$. Therefore in what follows, we shall prove that if (27) holds for J', then (27) also holds for J.

From the definition of x, y, x^* and y^*, we obtain

$$M(S_L) = x + y \quad \text{and} \quad M(S^*) = x^* + y^*, \tag{28}$$

$$M(S'_L) = \varepsilon x + y \quad \text{and} \quad M(S^{*\prime}) = \varepsilon x^* + y^*. \tag{29}$$

Moreover, if $r'_i > r'_j$ then $r_i > r_j$, and so T_i precedes T_j in S_L, and it follows that T'_i precedes T'_j in S'_L. Thus S'_L is an LRF schedule for J'. Therefore applying the inductive hypothesis,

$$M(S'_L) \leqq \lambda M(S^{*\prime}). \tag{30}$$

Furthermore the first q tasks of J are earliest processed in S_L, and it follows from Corollary 1 that x does not exceed λ times the minimum of $\sum_{i=1}^{q} w_i f_i$. Therefore

$$x \leqq \lambda x^*. \tag{31}$$

If $y \leqq \lambda y^*$ then by (28) and (31), the proof of (27) is completed. Otherwise, $(1-\varepsilon)x^* y > (1-\varepsilon)xy^*$ from $\varepsilon < 1$ and (31). Therefore by (28) and (29), $M(S'_L)M(S^*) = (\varepsilon x + y)(x^* + y^*) > (\varepsilon x^* + y^*)(x + y) = M(S^{*\prime})M(S_L)$. This, together with (30) yields $M(S_L) \leqq \lambda M(S^*)$. Thus the proof of (27) is completed again.

3.2. Proof of Theorem 3. Let I be a problem instance in which tasks $T_i (i = 1, \cdots, n)$ have processing times t_i and weights $w_i = \alpha t_i$ for a positive number α, and assume $M_L(I)/M^*(I) = \lambda$. Also let $[i]$ denote the index of the task processed in the ith order in an optimal schedule for I. Moreover we assume without loss of generality that tasks T_i are processed according to numerical order in a maximal schedule.

Consider another problem instance I' in which tasks $T'_i (i = 1, \cdots, n)$ have processing times t_i and weights $\alpha(1+\varepsilon_i)t_i$ where $\varepsilon_1 > \cdots > \varepsilon_n > 0$ and $\varepsilon_1 - \varepsilon_n \leqq \delta/\lambda$. Also let the number of processors in I' be equal to that in I. Then

$$M_L(I') = \Sigma\, \alpha(1+\varepsilon_i)t_i f_i > (1+\varepsilon_n)\alpha\, \Sigma\, t_i f_i = (1+\varepsilon_n)M_L(I).$$

Moreover if $M^*(I')$ is the cost of processing T'_i in the same order as T_i in an optimal schedule for I, then

$$M^*(I') = \Sigma\, \alpha(1+\varepsilon_{[i]})t_{[i]} f_{[i]} < (1+\varepsilon_1)\alpha\, \Sigma\, t_{[i]} f_{[i]} = (1+\varepsilon_1)M^*(I).$$

Thus we have $M_L(I')/M^*(I') > (1+\varepsilon_n)\lambda/(1+\varepsilon_1) > \{1-(\varepsilon_1-\varepsilon_n)\}\lambda \geqq \lambda - \delta$.

Acknowledgment. The authors wish to thank the referees for helpful suggestions.

REFERENCES

[1] A. K. CHANDRA AND C. K. WONG, *Worst-case analysis of a placement algorithm related to storage allocation*, this Journal, 4 (1975), pp. 249–263.

[2] E. G. COFFMAN, JR., *Computer and Job-Shop Scheduling Theory*, John Wiley, New York, 1976.

[3] W. L. EASTMAN, S. EVEN AND I. M. ISAACS, *Bounds for the optimal scheduling of n jobs on m processors*, Management Sci., 11 (1964), pp. 268–279.

[4] M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability—A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[5] R. MCNAUGHTON, *Scheduling with deadlines and loss functions*, Management Sci., 6 (1959), pp. 1–12.

[6] S. SAHNI, *Algorithms for scheduling independent tasks*, J. Assoc. Comput. Mach., 23 (1976), pp. 116–127.

[7] W. E. SMITH, *Various optimizers for single-stage production*, Naval Res. Logist. Quart., 3 (1956), pp. 59–66.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986

© 1986 Society for Industrial and Applied Mathematics

ON MAINTAINING DYNAMIC INFORMATION IN A CONCURRENT ENVIRONMENT*

UDI MANBER†

Abstract. This paper considers the amount of cooperation required for independent asynchronous processes to share a simple dynamic data structure. We present a scheme for designing efficient concurrent algorithms to add and remove elements from a shared pool of elements. The efficiency is measured mainly by the number of non-local operations that a process may have to make. Non-local operations may involve writing into a shared variable, locking, or sending a message, hence they introduce interference (or require cooperation). We derive upper and lower bounds on the interference in the worst case. Applications to distributed computation are also discussed.

Key words. asynchronous processes, concurrent data structures, distributed computing, load balancing, lower bounds, multisets, queues, trees

AMS(MOS) subject classification. 68B20, 68C05, 68C25

1. Introduction. This paper studies the amount of cooperation required for independent asynchronous processes to share a simple dynamic data structure. Consider, for example, a problem that we want to solve by using k processes (or processors) in parallel. Assume that the problem can be divided into n independent subproblems ($n \gg k$). We would like to assign subproblems to processes in a way that minimizes the (parallel) running time. In many cases, there is no way to predict the sizes of the subproblems or the relative speed of the processes. Hence, the best strategy is to allocate the subproblems dynamically. In this paper we study data structures for the type of concurrent dynamic allocation described above. We define an abstract data structure, called a *concurrent pool*, consisting of a multiset of elements (corresponding to subproblems in the example above) and the operations *add* and *remove*. Concurrent pools are used for load balancing, resource management, garbage collection, and more.

We present a scheme for designing efficient concurrent algorithms for the operations listed above. The efficiency is measured mainly by the number of nonlocal operations that a process may have to make (we also have to make sure that the local computation is efficient). In general, non-local operations may involve writing into a shared variable, locking, or sending a message, hence they introduce interference (or require cooperation). Our goal is to derive upper and lower bounds on the interference in the worst case.

The scheme consists of two parts. The first part is an efficient sequential algorithm for another, more complicated, data structure, which includes a split operation. The second and main part is a concurrent algorithm to locate elements distributed among the processes. The algorithm is based on a tree traversal. The processes traverse a tree, whose leaves point to the locations of the elements, leaving marks indicating where they have already searched so that other processes minimize their search. The traversal is complicated since processes proceed in an unpredictable order and since elements can be moved, added, and removed.

* Received by the editors July 10, 1984, and in final revised form October 15, 1985. A preliminary version of this paper appeared in the Sixteenth Annual ACM Symposium on the Theory of Computing, Washington, DC, April 30–May 2, 1984. Copyright 1984, Association for Computing Machinery, Inc. This article was typeset at the University of Wisconsin-Madison using a *troff* program running under UNIX. The final copy was produced on January 31, 1986. This research was supported in part by the National Science Foundation under Grants MCS-8303134 and DCR-8451397.

† Department of Computer Science, University of Wisconsin, Madison, Wisconsin 53706.

We also present a model of concurrent computation, based on a shared memory model, and prove lower bounds on the amount of interference. The gap between the upper bounds achieved by our algorithm and the lower bounds is small (a factor of $O(k^{\varepsilon})$, where k is the number of processes and $\varepsilon > 0$).

The algorithms presented in this paper served as a basis for the design of DIB—a package for Distributed Implementation of Backtracking [2]. DIB runs on an experimental multicomputer with no shared memory. Applications to distributed computation are discussed in section 6.

2. The problem. We want to develop an efficient data structure to represent *multisets.* We make no assumptions on the type of elements in the multisets. The data structure should support the following operations:

Remove(M, y): if M is not empty then choose an arbitrary element of M (any element will do), delete it from M, and assign it to y.

Add(M, x): add the element x to the multiset M. Whether or not x is already in M has no effect on this operation.

Data structures that support the operations listed above will be called *pools.* Pools can be used to store pointers to all waiting (independent) tasks in a multicomputer system. When a processor (or process) becomes available it removes a task from the pool and performs it; during its execution it may generate more tasks, which are added to the pool. Pools can also represent available resources in a system, available blocks of memory, independent events in a simulation, etc.

Pools can be very efficiently implemented in a sequential environment by stacks or queues, in which case every operation takes constant number of steps. (Stacks and queues are even more powerful than what we need since we are satisfied with arbitrary order of removals and we do not require LIFO [or FIFO] ordering.) If there are only two processes we can use a double-ended queue and let each process add and remove from a different end. Each operation still takes constant time; moreover, the only time one process may interfere with the other process is when the queue is almost empty. In this case interference in unavoidable since the processes must compete for the same elements. (Double-ended queues were in fact used in that way in the design of a parallel garbage collection algorithm with two processes, a list process, and a garbage collector [6]. This algorithm was generalized to many processes in [10] where a shared pool was used but there was no discussion on efficient implementation.) Obviously only one process can add or remove from one end of a queue at any given time, so a better data structure is required in order to achieve a high level of concurrency. In addition, we would like the design to be adaptable to a distributed environment. We assume that, although sometimes the multiset may be small or even empty, in general the number of elements involved is much greater than the number of processes. When the number of elements is small interference is unavoidable.

The processes perform add and remove operations in an arbitrary order and frequency. It is possible, for example, that only one process adds while all the others remove, or that all processes only add.

3. The model. A shared memory model, similar to the one described in [7], is used throughout the paper, except for section 6 which deals with distributed models. We use a shared memory model because of its simplicity. It allows us to gain some insight to the required cooperation of asynchronous processes that may be harder to get from a more complicated model. Lower bounds for this model should apply to other models with more expensive means of communication. We assume that the memory is

unbounded, and, in particular, we do not deal with memory allocation and garbage collection problems (nor do we use them in the lower bound proofs). These problems add complexity to the algorithms, which we ignore, but solutions to them usually strongly depend on the particular concurrent (or distributed) environment.

We assume a random access memory shared by many autonomous asynchronous reliable processes. A process is a state machine with its own local memory. In one atomic step a process can either read one variable of the shared memory, write into one variable of the shared memory, or lock part of the shared memory, and change its own state (a more general operation, test and set, is allowed in [7]). Several processes can read the same variable at the same time; however, if a process writes into a shared variable then no other process is allowed to access that variable at that time. Hence, some kind of locking is required. We use the regular notion of read and write locks (see for example [1,4]). A write lock gives exclusive access to its holder while a read lock only prevents writers from accessing the variable. If a process is denied access to a variable that is locked by another process then it waits until the lock is removed. We call such occurrences *collisions*. We make no assumptions on the implementation of locks (e.g., we do not assume that the allocation of the locks is according to a first come first serve policy). Our main goal in designing the algorithm is to minimize the number of collisions.

We believe that the definition of collision captures the simplest form of interference (or cooperation depending on the point of view) among asynchronous processes. This definition is also primitive enough to enable us to prove lower bounds. We do not consider in this paper issues of fairness or starvation. We think of the processes as being servers rather than customers. We also do not deal with issues of fault tolerance in this paper.

We make no assumptions on the order in which the processes access the shared memory. All operations are completely asynchronous. Thus, it is possible that some processes are much faster than others or that a process "goes to sleep" for a period of time and "wakes up" later.

In order to prove lower bounds on the add and remove operations we have to define them precisely. We assume that there is a unique variable associated with each element when it is created. This variable changes its value when the element is removed or added. Removing or adding an element may involve changing more than just one variable; however, the element is "formally" removed only when the unique variable associated with it has been changed. This assumption is required to rule out the possibility of two processes removing the same element concurrently.

4. The algorithm. The algorithm was designed to work best when all the processes have approximately the same behavior (i.e., it is designed for the average case). Surprisingly, it turns out that the algorithm is not far from optimal in the worst case.

The basic idea of the algorithm is as follows. The multiset is initially partitioned among the processes. Every process maintains a segment of the multiset and performs the operations add and remove locally as long as possible. Collisions will only occur when a process P empties its local segment and it still wishes to remove. In this case, P searches the other segments for a non-empty one. Since removing an element from another process' segment may involve collisions we have to minimize such occurrences. For example, if few processes are adding and most processes only remove and if non-local removals consist of only one element, then most removals may lead to collisions. We improve this simple solution in the following way. Once a process finds a non-empty segment and interrupts the "owner", it tries to take more than just one

element at a time. Since the size of a local segment may vary substantially, we should not take a fixed number of elements but a fixed portion. If the segment contains only one element then the "foreign" process simply removes it.

The algorithm consists of two separate independent parts. The first part includes a data structure to represent a local segment and constant time sequential algorithms for the operations add, remove, and split. The split operation divides a segment into two segments such that the larger segment has at most twice as many elements as the smaller; one of these segments will be given to the asking process. The second part is a scheme for finding a non-empty "donor".

Each process uses the sequential algorithms of add and remove within its local segment until the segment becomes empty. Then, it uses the second part of the algorithm to find a non-empty segment, locks it, performs the split operation, and takes one half as its new local segment. (In a non-shared memory environment the operation of moving the data from one process to another is a major one, while the splitting of a local segment is less important; these issues are discussed in section 6.)

4.1. Splitting a local segment. In some cases, splitting a local segment is straightforward. For example, the elements may differ only in one parameter (e.g., the address where the task resides, a probabilistic choice in a simulation). The n elements can be represented as a set of values for this parameter. If the set consists of all values in a given range then a segment always corresponds to a sub-range, and splitting a segment is easy. In this section we describe general algorithms for splitting, removing, and adding to an arbitrary multiset in constant time.

We use balanced binary trees in which each node corresponds to an element of the multiset. Nodes are added level by level from left to right and are removed in the opposite order. One way of achieving constant time per add and remove is to maintain a pointer to the last inserted node at the bottom of the tree, and put at each node pointers to its two siblings, parent, and two children. The parent and children pointers are then updated as new nodes are added or removed. It is straightforward to perform add and remove in constant time. It seems, however, that in order to split the tree we need to update $O(\log n)$ pointers, where n is the number of nodes in the tree. We can reduce the $O(\log n)$ complexity to $O(1)$ using the observation that except for the bottom level all the levels are full. In order to find out whether a node is at either the left or the right end of a level while we are adding or removing, it is sufficient to know what level (depth) it is. We keep a header with pointers to the root and last inserted node and with the bottom level number and the number of nodes at that level. When we are adding (removing) and the level is full (empty) we start a new level (go higher up using the parent pointers), and update the level number and the number of nodes at that level. To split we need only to decrement the level number of the lowest level. This automatically makes the two children of the root two new roots. There are two ways to take care of the old root, both take constant number of steps. We can either insert it at the bottom or maintain two types of trees, one with an additional root, call it a 2-roots tree, and the regular 1-root tree. Splitting a 1-root tree forms a 1-root tree and a 2-roots tree, and splitting a 2-roots tree forms two 2-roots trees. In any case, we need to keep a pointer to the root. It is easy to see that in the worst case we split to a third and two thirds (provided that there are at least 2 nodes in the tree).

We also have to find the middle of the lowest level (or next to lowest level in case it is less than half full); removals and additions from the other half should start at that middle. We solve this problem by using arrays instead of a linked list representation for each level (see Figure 1). Since the size of each level is fixed the arrays will never

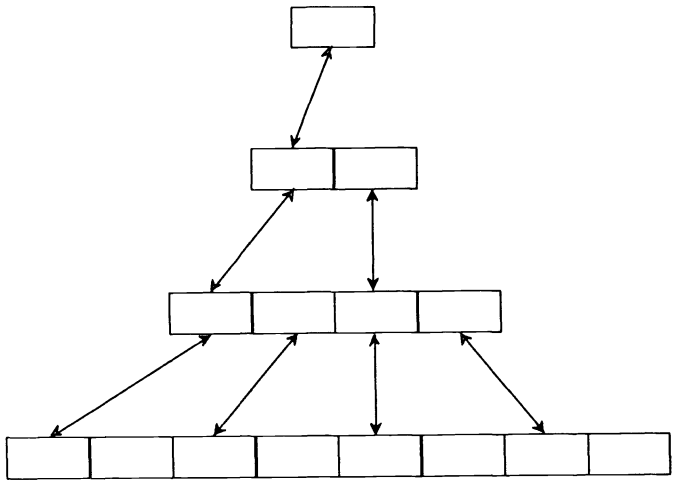


FIG. 1. *The data structure.*

have to be extended. Using arrays in this manner saves the sibling pointers and one of the children pointers (if one child is known then the other one can be easily found). The children pointers are needed in order to find the two new roots after a split and the parent pointers are needed to move up the tree when a level becomes empty.

Overall, we have an interesting data structure consisting of a collection of arrays of variable size linked into a tree, and we have shown that it can be expanded and split, all in constant time. Obviously, a process that performs split, add, or remove must lock the tree with a write lock. One drawback for using arrays in this way is that it makes memory allocation more complex.

4.2. Finding a non-empty segment. The problem of finding a non-empty segment is an interesting problem of locating dynamic entities in a concurrent environment. We want a solution that minimizes the total amount of non-local access in the worst case. We first present a simple algorithm and discuss its limitations, and then describe more efficient solutions. Since additions are always done locally we assume first that only removals take place. Additions will be considered at the end of this section. We assume that there are k processes attempting to remove n elements in an unpredictable order.

Denote the processes by P_0 to P_{k-1}, and consider them organized in a (logical) ring. Each process P_i maintains a local variable s_i, which is initially set to $(i+1) \bmod k$ (the successor of P_i in the ring). P_i starts its search for a non-empty segment from P_{s_i}. If this segment is empty then P_i continues the search in a cyclical order along the ring (excluding itself) until a non-empty segment P_j is found; s_i is then set to $(j+1) \bmod k$. In other words, all the processes search by traversing the ring in cyclical order. Notice that the algorithm does not assume a knowledge of n but it does assume a knowledge of k. A period of time during which all processes are "visited" at least once is called a *traversal.* Traversals are important as the following theorem shows.

THEOREM 1. *Assume that the maximal total number of non-local accesses (visits) performed by the algorithm during any traversal is $V(k)$. Then the total number of collisions caused by k processes to remove n elements is $O(V(k) \log n)$.*

Proof. To bound the number of collisions that occur while all processes remove all elements we look at the change in the number of elements during a traversal. It is possible that very few elements are removed during a traversal since segments may be

split and elements may be moved around, and thus may be "missed" by most visits. However, even if very few elements are removed, the elements are distributed more evenly. At some point during the traversal the number of elements in each segment was either cut by a third (as a result of a visit) or it was reduced to 0. Since there are no additions, this fact implies that the number of elements in the largest segment after a traversal is at most two thirds of what it was before the traversal. As a result, all the elements will be removed after $O(\log n)$ distinct full traversals. By the results of the previous section, each non-local access can cause only constant number of collisions. Thus, the number of collisions occurring in the algorithm is $O(V(k) \log n)$ in the worst case. □

COROLLARY 1. *The total number of collisions caused by k processes removing n elements by the ring algorithm described above is* $O(k^2 \log n)$ *in the worst case.*

Proof. It is easy to see that the maximal number of non-local accesses during a ring traversal is $O(k^2)$. The corollary follows from Theorem 1. □

It seems at first that the algorithm above can be improved by "spreading" the processes and letting each one make a traversal in a different order. We show that any order of traversals leads to $\Omega(k^2)$ non-local accesses per traversal in the worst case as long as the only way to find whether a segment is empty is to visit it. Let $s_{i1}, s_{i2}, \ldots, s_{ik}$ be the order of traversal of process i and assume for simplicity that k is even. Consider all the segments s_{ij} such that $j > \frac{1}{2}k$ (i.e., the segments at the second half of each traversal). Since there are $\frac{1}{2}k^2$ such segments one of them must appear (in the second half) at least $\frac{1}{2}k$ times. Take now the $\frac{1}{2}k$ processes that visit this segment in the second half of their traversal. We can "force" them to a total of $\frac{1}{4}k^2$ non-local accesses without any complete traversal. Hence, the only way to improve the algorithm is to use global variables that contain information about empty processes. However, we have to be careful to update these global variables efficiently. For example, we cannot afford to have only one variable that contains all the information since the processes may all collide there frequently (this is basically the queue solution).

We first solve a restrictive case of the problem and then show how to extend the solution to the general case. We assume that initially there is exactly one element in the local segment of each process. Let each process be associated with a leaf in a complete binary tree (for simplicity we assume that the number of processes is a power of 2). Each leaf initially contains the element associated with the process. When this element is removed the leaf is said to be *empty*. An internal node contains two flags, corresponding to its two children, which are set to empty if all the leaves in the child's subtree are empty. Since we assumed that there are no additions, once a node becomes empty it stays empty; however, the flag is not set immediately after the node becomes empty, hence a process may pursue an empty subtree.

Each process starts the removals from its own leaf and traverses the tree according to the procedure described below (an example follows the procedure). We assume that only one process can access a node at any given time (this can be implemented through locking). Every such access will be counted as a possible collision. A process P arrives at an internal node v always from below, and this happens when P finds one of v's subtrees to be empty. P then sets the corresponding flag in v to empty. If both flags in v are empty then P continues up the tree. Assume that P arrived from the right side. If the left flag of v is not empty then P goes down the left subtree of v directly (in one step) to a leaf that is in the same relative position in the left subtree of v as P's starting leaf is in the right subtree of v; we call this leaf the *matching descendant* of P and v. The case of reaching v from the left child is similar. This way, if many processes collide at an internal node they are spread among the leaves. Since the tree

is static the nodes can be stored in a linear array and all the index calculation required to move from one node to another can be implemented in constant time.

Example 1. The tree is given in Figure 2. First consider the case of only one process, P, initially at h. P traverses the tree in the following order:

$$h,d,i,d,b,j,e,k,e,b,a,l,f,m,f,c,n,g,o,g,c,a.$$

Let us now start with 3 processes: P at h, Q at i, and R at m, and assume that in each time unit one process completes the computation on one node. The following is an example of one possible order of execution. Each process is followed by the node it visits. A boldface node indicates that the node has been found empty in this step for the first time.

P:**h**, P:d, Q:**i**, P:i, P:**d**, P:b, Q:d, Q:b, P:**j**, P:e, P:**k**, P:**e**, Q:k, Q:e, Q:**b**, Q:a, P:b, P:a, R:**m**, R:f, R:**l**, R:**f**, R:c, P:l, P:f, R:**o**, R:g, R:**n**, R:**g**, R:**c**, P:c, Q:m, R:**a**, P:a, Q:f, Q:c, Q:a.

To analyze the algorithm we count not only collisions but actual steps. Notice that if only one process is active then although the traversal algorithm is more complicated than a straightforward sequential tree traversal the number of steps is still linear in the number of nodes (each internal node is visited twice). Let $S(k)$ denote the maximal number of steps taken by k processes traversing a tree with k leaves. The analysis of the algorithm relies on the following lemma.

LEMMA 1. *For any internal node v, all processes that visit leaves in both subtrees of v visit the left subtree first, or they all visit the right subtree first.*

Proof. Assume the contrary. Let P be a process that started in the left subtree of v and then visited the right subtree, and let Q be a process that visited the subtrees

```
Procedure Traverse (node) ;
{ initially node is the process' own leaf }

if node is a leaf then
    if node is non-empty then
        remove the element ;
    endif ;
    set the corresponding flag in node's parent to empty;
    Traverse(parent of node);
else
    if both flags of node are empty then
        if node = root then
            terminate ;
        else
            set the corresponding flag in node's parent to empty;
            Traverse(parent of node);
        endif ;
    else
        Traverse (matching descendant of initial node) ;
    endif ;
endif ;
```

FIG. 2. *Example* 1.

in the opposite order. In order to visit both subtrees P and Q must have checked first if both flags in v are empty. Let's assume that Q checked v after P (we assumed that only one process can access a node at a time). In this case, at the time Q checked v both flags in v are empty and Q continues up the tree. □

THEOREM 2. *The total number of steps taken by k processes to remove k elements using procedure Traverse is* $O(k^{1.59})$ *in the worst case.*

Proof. We use a recursive argument. For simplicity, assume that k is a power of 2. Let T be a complete binary tree rooted at r, and assume, without loss of generality, that the first process that arrives at r started at the left subtree. By Lemma 1 no process that started at the right subtree will visit the left subtree. Hence, the worst case of traversing T occurs when all the processes in the left subtree traverse it in the worst possible way, then they are dispersed to the bottom of the right subtree and then all processes traverse the right subtree together in the worst possible way. We get the following recurrence relation for $S(k)$:

$$S(2k) \leqq 3S(k) + O(k).$$

The factor 3 arises from 1) processes in the left subtree, 2) processes in the right subtree, and 3) processes from the left subtree that traverse the right subtree. Solving the recurrence relation, we get

$$S(k) = O(k^{\log_2 3}) = O(k^{1.59}).$$ □

It is possible to improve the asymptotic behavior of the algorithm by using multiway trees and distributing the processes down from an internal node in a better way. One can achieve a complexity of $O(k^{1+\varepsilon})$ for any $\varepsilon > 0$. However, while this leads to an asymptotic improvement, there is no significant improvement for practical values of k. We describe the modifications below only for the purpose of comparing them to the lower bounds in section 5. Consider a 4-way tree. Using a similar traversal procedure and similar arguments as above one can show that the worst case occurs when the processes in one subtree traverse it in the worst possible way, then they move down to a second subtree and, with the processes in this subtree, traverse it in the worst possible way, then they all move to the third and then to the fourth subtree. This leads to the recurrence relation $S(4k) \leqslant (1+2+3+4)\, S(k) + O(k)$, which gives a slightly worse solution. However, if, instead of "sending" the processes from the first and second subtrees together to the third subtree, we "spread" them and send each to the remaining subtrees (the third and the fourth) we get the recurrence relation $S(4k) \leqslant (1+2+2+4)\, S(k) + O(k)$, which turns out to yield the same bound as the binary case. Using the same technique for 6-way trees results in the recurrence relation $S(k) \leqslant (1+2+2+2+3+6)\, S(k) + O(k)$ (processes in the first tree go to the second, the second to the third and fourth, the third to the fifth and sixth, the fourth to the fifth and sixth, and the fifth to the sixth), which yields $S(k) = O(k^{\log_6 16}) = O(k^{1.55})$. Using m-ary trees we get the following theorem.

THEOREM 3. *For any* $\varepsilon > 0$ *there exists an algorithm to remove k elements by k processes using* $O(k^{1+\varepsilon})$ *steps in the worst case.*

Proof. We use a complete m-way tree such that m is a constant that depends on ε. Assume, for simplicity, that $k = m^h$ where h is the height of the tree (a leaf is at height 0), and that each leaf in the tree holds one element. Denote the processes by $P_0, \ldots, P_{k-1}$, and let P_i start at leaf i, such that the leaves are ordered from the left. Given an internal node v of height $h(v)$, define

$$index(P_i, v) = \left\lfloor \frac{(i \bmod m^{h(v)})}{m^{k(v)-1}} \right\rfloor.$$

index(P_i,v) indicates the child of v from which P_i starts to traverse the subtree rooted at v (P_i may have started in a different subtree). The data structure maintained in each internal node v is the following. There are m flags $F_1,\ldots,F_m$ and m variables $a_1,\ldots,a_m$, corresponding to the m children. There is also a list L of all the children that are not yet known to be empty. The list is maintained as a circular linked list with a pointer to it such that every time a non-empty child is selected the pointer advances. Operations on one node are assumed to be atomic; we make sure that such operations always consist of constant number of primitive steps. As in procedure Traverse above, a process always reaches an internal node v from below and then either goes up (in case v is known to be empty) or goes down directly to a matching descendant leaf in one of the subtrees. We proceed to describe how to select this subtree.

Let P_i reach v from child j ($0 \leqq j \leqq m-1$) such that *index*$(P_i,v) = d$. If F_j is not empty then P_i sets F_j to empty and removes j from L. If L is empty (i.e., v is empty) then P_i (along with all other processes reaching v from now on) goes up to v's parent; if v is the root then the algorithm terminates. If L is not empty then P_i picks the next non-empty child s from L, sets a_d to s, and goes directly to a matching descendant leaf in s; a_d indicates to all other processes P_q such that *index*$(P_q,v) = d$ that when they reach v they should continue from the matching descendant in child a_d. If F_j is empty, which implies that P_i is not the first process to reach v from j, then P_i checks a_d; if $a_d = j$, which implies that P_i is the first with index d to reach v from j, then P_i picks the next non-empty child s from L and sets a_d to s; it then follows the matching descendant in s. If $a_d \neq j$ then P_i follows the matching descendant from a_d.

To analyze the worst case behavior of the algorithm we use a recursive argument, very similar to the proof of Theorem 2. Let $S(k)$ denote the maximum number of steps taken by k processes traversing a tree with k leaves. Consider a tree rooted at r with $k' = mk$ leaves. Let P be the first process to reach r and assume that it reaches r from j. After P sets F_j to empty and removes it from L no other process will go down from r to a leaf in j. The only steps involved in traversing j result from processes that are already in j. Hence, we can assume that these processes traverse j in the worst possible way using $S(k)$ steps. The same argument holds for the second subtree that becomes empty, and the worst case occurs when it is the subtree to which the processes of j were directed; in this case the additional number of steps is at most $2S(k)$ since the processes from j were distributed in the same way as if they had started at the other subtree. We continue by selecting the subtree with the greatest number of processes, having all of them traverse it in the worst possible way, and distributing them among the other non-empty subtrees. By the selection of the next non-empty tree from L we are guaranteed that the processes with different indices are distributed equally among the subtrees. (Actually, it is not important for the proof to implement the data structure for the list L efficiently since m is a constant anyway.) To count the total number of steps we start at the last phase in which all processes traverse the last child using at most $mS(k)$ steps. The next to last phase results in at most $\lceil m/2 \rceil S(k)$ steps and so on. In addition, there are at most $O(mk)$ steps for "visiting" r. Hence, the total number of steps is at most

$$(m + \lceil m/2 \rceil + \lceil m/3 \rceil + \ldots + \lceil m/(m-1) \rceil + 1)S(k) + O(mk)$$

$$\leqq cm \log_2 m S(k) + O(mk)$$

(where c is a constant), resulting in the recurrence relation

$$S(mk) \leqq cm \log_2 m S(k) + O(mk),$$

which implies

$$S(k) = O(k^{\log_m(cm\log_2 m)}).$$

Since

$$\log_m(cm\log_2 m) = 1 + \frac{\log_2(c\log_2 m)}{\log_2 m},$$

choosing m such that

$$\varepsilon > \frac{\log_2(c\log_2 m)}{\log_2 m}$$

completes the proof. □

So far we discussed a restrictive case. In the general case we divide the algorithm into *rounds.* We make sure that in each round all segments are visited by at least one "foreign" process seeking a "donor". Since each visit splits the segment by at least a third we can have at most $O(\log n)$ rounds before all n elements are removed. Notice that it is possible that the segment of one process contains all the elements in the multiset. Hence all segments must be searched and a complete round is indeed necessary.

We modify the algorithm by replacing the flag in each node with a *round counter.* An internal node marked with a counter c implies that all of its leaves "donated" in round c. Processes run the traversal algorithm, starting with their own leaf, only when their local segments are empty and they are looking for a donor. Once they find a donor they continue their local operations until they empty their segments again, in which case they continue the traversal from the last node they visited. Instead of marking an internal node as visited, each process makes sure that the node's counter is no less than its local counter. If a process finds a node with a larger round number than its own counter then it updates its own counter and starts the traversal from the beginning. Round c is terminated when the root is marked c. When a process reaches the root it starts the traversal again from its leaf incrementing its local round number.

THEOREM 4. *Let the multiset initially contain n elements that are divided among the processes in an arbitrary way. The total number of collisions resulting from k processes executing n removals and arbitrarily many additions is* $O(S(k)\log n)$ *in the worst case.*

Proof. Assume first that no additions are performed. By Theorem 1, since a round corresponds to a traversal and the number of steps in a round corresponds to the number of visits in a traversal, at most $O(S(k)\log n)$ collisions can occur in the worst case. Additions are always performed locally; hence, the only time they can be involved in a collision is when another process (or processes) attempts to split the local segment. However, these collisions can be attributed to the foreign processes. We counted steps (visits) in the evaluation of $S(k)$ and each visit was counted as a collision whether the local segment was active or not. Hence, the upper bound is not changed and the result follows. □

If more than n removals are performed then obviously the worst case occurs when all the elements are removed and then repeatedly one element is added and all processes attempt to remove it. The algorithm is not very efficient for the case of multisets with very few elements. (The best solution in this case is probably to decrease the concurrency and let only few processes be active, assuming again that the processes are servers and that fairness is not an issue.)

5. Lower bounds. In this section we present a lower bound on the number of collisions in the worst case. Since the add operation in the algorithm described in the

previous section does not contribute to the number of collisions in the worst case we ignore additions for the lower bounds. The following scenario is considered: There are n elements in the multiset, and k processes, $P_0, P_1, \ldots, P_{k-1}$, are trying to remove all the elements. The time processes spend locally per operation is not counted; **only** collisions are counted. We use an adversary argument in the following way. Given a data structure and algorithms, we will produce a schedule for the processes that leads to many collisions. Let $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$ be the sequence of elements P_i removes provided that all other processes are not active.

LEMMA 2. *There exist i and j, $0 \leq i, j \leq k-1$, and r and s, $1 \leq r,s \leq \lfloor n/k \rfloor + 1$, such that $x_{i_r} = x_{j_s}$.*

Proof. There are altogether more than n elements in the k subsequences of size $\lfloor n/k \rfloor + 1$, hence one element must appear in two subsequences. □

THEOREM 5. *Every algorithm for concurrent removal of n elements by k processes, where $k > 2$, and $n = \Omega(k^a)$, $a > 1$, produces $\Omega(k \log n)$ collisions in the worst case.*

Proof. Find i, j, r, and s that satisfy the conditions of Lemma 2, and assume that all processes except P_i and P_j are not active. Let P_i remove all the elements up to x_{i_r} and then let P_j start removing its elements. If P_j does not change its order of removals as a result of the removals of P_i then we can cause a collision by letting them both attempt to remove x_{i_r} $(= x_{j_s})$ at the same time. Otherwise, in order to know that P_i has been active, P_j must read a variable that P_i has written. Let w be the first such common variable. The execution sequences of P_i and P_j until they access w are independent. Hence we can let them arrive at w at the same time thus producing a collision. In any case, a collision can be forced to happen after at most $\lceil 2n/k \rceil$ elements are removed. After the collision we "allow" all the processes to learn about it and change their removal orders accordingly (this probably causes more collisions but we do not count them). The number of remaining elements is now $R_1(n) \geqq n - \lceil 2n/k \rceil \geqq n(1-2/k) - 1$. We now use the same argument repeatedly to get $R_m(n) \geqq R_{m-1}(n)(1-2/k) - 1$. Solving the above, we get

$$R_m(n) \geqq n(1-2/k)^m - \frac{k}{2}(1-(1-2/k)^m) \geqq n(1-2/k)^m - k/2 \approx ne^{-2m/k} - k/2.$$

Hence, if n and k are sufficiently large and $n = \Omega(k^a)$, $a > 1$, $m = \Omega(k \log n)$ iterations are required to remove all the elements, and the theorem follows. □

In particular, Theorem 5 implies that if k is a small constant then the algorithm in section 4 is optimal; for large k there is a gap of a factor of k^ϵ between the lower and upper bounds.

6. Applications to distributed computation. If processes communicate by messages through a communication network then, conceptually, one can still regard the whole network as a shared memory [7]. However, the costs associated with communication are different. Considering all the issues involved in a distributed implementation of pools is beyond the scope of this paper. In this section we discuss only a few observations and related work.

Assume that, instead of counting collisions, we measure the complexity by counting the number of messages, where a message contains one element. Furthermore, assume that every two processors are connected and can communicate directly. The lower bound results of section 5 still hold since every time two processes need access to the same element they must communicate. However, in this case, one can get a trivial lower bound of $\Omega(n)$ on the number of messages in the worst case by the following observation. Let P be the process that initially has the smallest number of elements

in its local segment. P has at most n/k elements. If P is the only active process then it has to get the rest of the elements from the other processes, which implies $\Omega(n)$ messages.

In many cases, in particular in local area networks, the communication overhead depends more on the number of messages than their size (provided, of course, that the messages are not too large). In this case, the number of collisions is a more realistic measure than the number of bits that have to be sent since it measures in some sense the amount of "dialogue". Also, if one considers the example of dividing a large problem into subproblems that was given in the introduction, it may be possible to have a short description for a collection of subproblems. Hence, a message containing many elements can still in some cases be short. The lower bounds of section 5 still hold for this model. Moreover, they are not far from optimal since the upper bounds described in section 4 are applicable provided that sending half of a segment can be considered as one message. The internal nodes in procedure Traverse of section 4.2 can be stored in preassigned machines. A process P can access an internal node by sending an appropriate message to process Q where the node is stored. Q then performs the operation on the internal node locally, and sends out a message to P describing the outcome.

A similar problem of allocating resources in a network consisting of a tree of processors where a message can only go between neighboring processors is discussed in [3]. A probabilistic algorithm is given and its expected average time performance is analyzed.

Some of the ideas of the algorithms of section 4 have been used in the design of DIB–a package for Distributed Implementation of Backtracking [2]. DIB is a general-purpose package that allows applications that use backtrack or branch and bound to be implemented on a multicomputer. The package is currently running on the Crystal multicomputer at the University of Wisconsin–Madison. It is based on a distributed algorithm, transparent to the user, that divides the problem into subproblems and dynamically allocates them to the available machines. Initially, the problem is divided arbitrarily among the machines. When a machine finishes all its subproblems it sends a *request for work* to another machine selected by an algorithm similar to the algorithms described in section 4.2. A machine that receives such a request grants it by dividing its work into several subproblems and sending half of them to the requesting machine. Initial results of using DIB are very promising. In particular, the speedup we achieve for many simple implementations of backtracking is almost linear. The communication overhead is small and the computation load is equally divided among the machines.

The most simple case where all subproblems are independent and known in advance is similar to the definition of pools. In general, the subproblems may not be completely independent and the generation of subproblems (i.e., the partitioning) may depend on the execution of the algorithm. For example, in a typical branch and bound algorithm the branching depends on the current subproblem and the current bound may be used in solving other subproblems.

7. Extensions and further research. It is possible to support processes that are allocated and destroyed dynamically. We need to be able to insert and delete leaves from the global tree concurrently with the other operations. Efficient concurrent algorithms for insertions and deletions in external binary search trees are described in [9]; general concurrent binary search trees are discussed in [5,8]. These algorithms can be easily adapted to this application. If a dynamic tree is used then it may not be possible to find the matching descendant of a node (see algorithm Traverse in section

4.2) in constant time. However, the search for matching descendants need not interfere with the removals or additions operations; it only interferes with inserting or deleting a process.

Several open questions remain.

Can one improve the tree traversal algorithm? Although the asymptotic difference between lower and upper bounds is very small, it is not clear whether efficient solutions for small number of processes can come close to the lower bound.

We defined collisions as being either read-write or write-write conflicts. If we count only write-write conflicts, is it possible to improve the upper bounds or (more likely in our opinion) prove the same lower bounds?

In a distributed environment fault tolerance is a major issue. It is important to implement pools such that recovery from failures (e.g., a loss of a local segment) is possible. Several alternatives are now being studied and implemented in the context of the DIB project.

8. Conclusions. We presented in this paper a design of an abstract concurrent data structure. We proved upper and lower bounds on the amount of interference among competing processes under a shared memory model. Abstract data structures have proven to be very helpful in the design of sequential algorithms. Having an arsenal of data structures and efficient implementations makes it easier to formalize problems and to solve them. We hope that the same will be true for concurrent computation.

Acknowledgments. The author wishes to thank Raphael Finkel and Richard Ladner for many stimulating discussions that made this paper possible, and the anonymous referees for comments that improved the presentation of the paper.

REFERENCES

[1] C. ELLIS, *Concurrent search and insertion in AVL trees*, IEEE Trans. Comput., C-29 (1980), pp. 811–817.

[2] R. A. FINKEL AND U. MANBER, *DIB–A distributed implementation of backtracking*, Fifth International Conference on Distributed Computing Systems, Denver, May 1985, pp. 446-452.

[3] M. J. FISCHER, N. D. GRIFFETH, L. J. GUIBAS AND N. A. LYNCH, *Probabilistic analysis of a network resource allocation algorithm*, Information and Control, to appear.

[4] J. N. GRAY, *Notes on database operating systems*, in Lecture Notes in Computer Science 60, Springer-Verlag, Berlin, 1978, pp. 393-481.

[5] H. T. KUNG AND P. L. LEHMAN, *Concurrent manipulation of binary search trees*, ACM Trans. Database Systems, 5 (1980), pp. 354-382.

[6] H. T. KUNG AND S. W. SONG, *An efficient parallel garbage collection system and its proof of correctness*, 18th Annual Symposium on Foundations of Computer Science, October 1977, pp. 120-131.

[7] N. A. LYNCH AND M. J. FISCHER, *On describing the behavior and implementation of distributed systems*, Theoret. Comput. Sci., 13 (1981), pp. 17-43.

[8] U. MANBER AND R. E. LADNER, *Concurrency control in a dynamic search structure*, ACM Trans. Database Systems, 9 (1984), pp. 439-455.

[9] U. MANBER, *Concurrent maintenance of binary search trees*, IEEE Trans. Software Engineering, SE-10 (1984), pp. 777-784.

[10] I. A. NEWMAN AND M. C. WOODWARD, *Alternative approaches to multiprocessor garbage collection*, 1982 International Conference on Parallel Processing, Bellaire, MI, August 1982, pp. 205-210.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
© 1986 Society for Industrial and Applied Mathematics
021

SUMS OF DIVISORS, PERFECT NUMBERS AND FACTORING*

ERIC BACH†‡, GARY MILLER§ AND JEFFREY SHALLIT¶‡

Abstract. Let N be a positive integer, and let $\sigma(N)$ denote the sum of the divisors of N (e.g. $\sigma(6)=1+2+3+6=12$). We show computing $\sigma(N)$ is equivalent to factoring N in the following sense: there is a random polynomial time algorithm that, given $\sigma(N)$, produces the prime factorization of N, and $\sigma(N)$ can be computed in polynomial time given the factorization of N.

We show that the same result holds for $\sigma_k(N)$, the sum of the kth powers of divisors of N.

We give three new examples of problems that are in Gill's complexity class BPP: perfect numbers, multiply perfect numbers, and amicable pairs. These are the first "natural" sets in BPP that are not obviously in RP.

Key words. factoring, sum of divisors, perfect numbers, random reduction, multiply perfect numbers, amicable pairs

AMS(MOS) subject classifications. 68C25, 10A25, 10A20

1. Introduction.

Integer factoring is a well-known difficult problem whose precise computational complexity is still unknown. Several investigators have found algorithms that are much better than the classical method of trial division (see [Guy 1], [Pol], [Dix], [Len]).

We are interested in the relationship of factoring to other functions in number theory. It is trivial to show that classical functions like $\varphi(N)$ (the number of positive integers less than N and relatively prime to N) can be computed in polynomial time if one can factor N; hence computing $\varphi(N)$ is "easier" than factoring. One would also like to find functions "harder" than factoring. The first result in this area was given in Gary Miller's thesis [Mill]. Miller showed that if the Extended Riemann Hypothesis (ERH) is true, then given $\varphi(N)$ one can produce the factorization of N in polynomial time. Thus computing $\varphi(N)$ is "equivalent" to factoring. He also demonstrated a similar equivalence between factoring and two other number-theoretic functions, $\lambda(N)$ and $\lambda'(N)$ (defined below). Long pointed out that if one is willing to use randomization, the ERH assumption in the above results can be eliminated, and further showed that the calculation of orders in the multiplicative group of integers (mod N) is randomly equivalent to factoring [Long]. (Section 2 below gives a slightly more general version of these results.) Using the results of Miller and Long, a method for composite-modulus discrete logarithm problems implies a method for factoring [Bach].

In this paper, we demonstrate an equivalence between factoring and computing the function $\sigma(N)$, the sum of the divisors of N. More formally, we prove the following

THEOREM 1. *Given the factorization of N, $\sigma(N)$ can be computed in polynomial time.*

THEOREM 2. *Given $\sigma(N)$, we can produce the factorization of N in random polynomial time.*

* Received by the editors February 28, 1984, and in revised form August 28, 1985. A preliminary version of this paper was presented at the 16th ACM Symposium on the Theory of Computing in Washington, DC [BMS].

† Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706.

‡ The research of this author was sponsored in part by the National Science Foundation under grant MCS 82-04506.

§ Department of Computer Science, University of Southern California, Los Angeles, California 90089.

¶ Department of Computer Science, University of Chicago, Chicago, Illinois 60637.

Theorem 1 is easy to prove; for if

$$N = p_1^{e_1} \cdots p_r^{e_r}$$

then

$$\sigma(N) = \frac{p_1^{e_1+1}-1}{p_1-1} \cdots \frac{p_r^{e_r+1}-1}{p_r-1} \tag{0}$$
$$= (p_1^{e_1}+\cdots+1)\cdots(p_r^{e_r}+\cdots+1).$$

(e.g. [HW, Thm. 275]). Thus $\sigma(N)$ can be computed in polynomial time. □

In §§ 3 and 4 below, we will prove Theorem 2.

Section 5 discusses extensions to $\sigma_k(N)$, the sum of the kth powers of the divisors of N. Section 6 discusses some interesting corollaries, including three examples of natural problems in Gill's complexity class BPP that are not obviously in RP.

(We assume the reader is familiar with probabilistic complexity classes, as discussed in [Gill]. Recall that BPP is the class of languages recognized in polynomial time by a probabilistic Turing machine, with *two*-sided error probability bounded by a constant away from 1/2. RP is the class of languages recognized in polynomial time by a probabilistic Turing machine with *one*-sided error.)

A few words about notation: we use N to denote a number to be factored, and p and q represent prime divisors of N. The factorization of N is given by

$$N = p_1^{e_1} \cdots p_k^{e_k}.$$

We use $p^e \| N$ to mean $p^e | N$ but $p^{e+1} \nmid N$, i.e. p^e is the highest power of p dividing N. By $v_p(N)$ we mean the exponent of the highest power of p dividing N; e.g. in the example of the previous sentence, $v_p(N) = e$.

If R is a ring, we use R^* to denote the group of invertible elements. For example, $\mathbb{Z}_N$ is the ring of integers (mod N), and $\mathbb{Z}_N^*$ is the group of elements relatively prime to N. By GF(q^k) we mean the Galois field with q^k elements. $N_{E/F}(a)$ is the relative norm of the element a.

By *factoring* an integer N, we mean producing the complete factorization. By *splitting* we mean finding a nontrivial divisor.

$\lambda(N)$ denotes Carmichael's lambda function. $\lambda(N)$ is the exponent of the group $\mathbb{Z}_N^*$, i.e. the least positive e for which $a^e \equiv 1 \pmod N$ for all $a \in \mathbb{Z}_N^*$. It is easy to show that

$$\lambda(N) = \operatorname{lcm}_i \{p_i^{e_i-1}(p_i-1)\}.$$

$\lambda'(N)$ is defined similarly:

$$\lambda'(N) = \operatorname{lcm}_i \{p_i - 1\}.$$

2. Splitting N given a multiple of $p-1$. Most of the equivalences between functions discussed in § 1 are proved as follows: let N be composite with prime divisors p and q. By doing computations in $\mathbb{Z}_N$, and using the Chinese remainder theorem, we get the effect of doing computations in $\mathbb{Z}_p$ and $\mathbb{Z}_q$. Given a randomly chosen $a \in \mathbb{Z}_N$, we construct a number $x_a \in \mathbb{Z}_N$ such that $x_a \equiv 0 \pmod p$, but $x_a \not\equiv 0 \pmod q$ with high probability. Thus gcd (x_a, N) gives a nontrivial divisor of N. (This is one of the few general ideas for factoring integers.)

The first half ($x_a \equiv 0 \pmod p$) is usually proved by exploiting some algebraic structure; the second half ($x_a \not\equiv 0 \pmod q$) by showing that the set of $a \in \mathbb{Z}_N^*$ for which $x_a \equiv 0 \pmod q$ is a *proper* subgroup of the group $\mathbb{Z}_N^*$.

As an example, we now show how to split N given a multiple of $p-1$. This theorem and its proof can essentially be found in [Mill] and [Long]. However, we include it here for two reasons: for completeness and to motivate the main ideas.

THEOREM 3. *There is an algorithm* S(N, M, a) *with the following properties*: *Let* N *be odd and divisible by at least two distinct primes. Let* $p|N$. *Then given* M *such that* $p-1|M$, *the algorithm* S(N, M, a) *splits* N *for at least* 50% *of the choices for* $a \in \mathbb{Z}_N$, *and terminates in time bounded by a polynomial in* log M *and* log N.

Proof. The body of Algorithm S is given below.

ALGORITHM S(N, M, a):

S1. [*Check for nontrivial GCD*]. If gcd $(a, N) = r$ and $r \neq 1$, then return r and stop.

S2. [*Set exponent*]. Set $Q \leftarrow MN$.

S3. [*Compute power using modular exponentiation algorithm*]. Let $b \equiv a^Q \pmod N$.

S4. [*Test*]. If $b \not\equiv \pm 1$ then return gcd $(b-1, N)$ and stop. Else if Q is even and $b \equiv 1$, set $Q \leftarrow Q/2$ and return to step S3.

S5. [Q *odd or* $b \equiv -1$]. Failure. Return nothing. Stop.

LEMMA A. *For at least* 50% *of all choices of* a, $1 \leqq a < N$, *Algorithm* S(N, M, a) *terminates after having produced a nontrivial divisor of* N.

Proof of Lemma A. If $a \in \mathbb{Z}_N - \mathbb{Z}_N^*$ then step S1 of the algorithm will always discover a nontrivial factor of N. Hence we may assume $a \in \mathbb{Z}_N^*$.

Let $p|N$. By assumption $p-1|M$, so $p-1|MN$. We examine two cases:

Case I. There exists at least one other prime $q|N$ such that $q-1 \nmid MN$. Then $a^{MN} \equiv a^{(p-1)kN} \equiv 1 \pmod p$ but

$$\{a \in \mathbb{Z}_N^* \mid a^{MN} \equiv 1 \pmod q\}$$

is a proper subgroup of $\mathbb{Z}_N^*$ and so $a^{MN} \not\equiv 1 \pmod q$ at for at least 50% of all choices of a. For these choices of a, step S4 produces a nontrivial divisor of N.

Case II. $q-1|MN$ for all primes $q|N$. Then $\lambda(N)|MN$, so

$$G_0 = \{a \in \mathbb{Z}_N^* \mid a^{MN} \equiv 1 \pmod N\} = \mathbb{Z}_N^*.$$

Now consider the following chain of subgroups:

$$G_1 = \{a \in \mathbb{Z}_N^* \mid a^{MN/2} \equiv 1 \pmod N\},$$
$$G_2 = \{a \in \mathbb{Z}_N^* \mid a^{MN/4} \equiv 1 \pmod N\},$$
$$\vdots$$
$$G_k = \{a \in \mathbb{Z}_N^* \mid a^{MN/2^k} \equiv 1 \pmod N\}$$

where $k = v_2(MN)$, the largest exponent of 2 dividing MN. Clearly $G_k \neq \mathbb{Z}_N^*$ since $\lambda(N)$ is even, but $MN/2^k$ is odd. Hence there exists a subscript j for which $G_{j-1} = \mathbb{Z}_N^*$ but $G_j \neq \mathbb{Z}_N^*$. We claim that $H_j \neq \mathbb{Z}_N^*$ also, where H_j is a subgroup of $\mathbb{Z}_N^*$ given by

$$H_j = \{a \in \mathbb{Z}_N^* \mid a^{MN/2^j} \equiv \pm 1 \pmod N\}.$$

We will produce an $x \in \mathbb{Z}_N^*$ not in H_j. Let $q^e \parallel N$ such that

$$b^{MN/2^j} \equiv -1 \pmod{q^e}$$

for some b; such a q must exist, for otherwise G_j would equal $\mathbb{Z}_N^*$. Let x be given by

$$x \equiv b \pmod{q^e},$$
$$x \equiv 1 \pmod{N/q^e};$$

then

$$x^{MN/2^j} \equiv -1 \pmod{q^e},$$
$$x^{MN/2^j} \equiv 1 \pmod{N/q^e}$$

and so $x \notin H_j$. Thus H_j is a *proper* subgroup of $\mathbb{Z}_N^*$.

Now we claim that for each $x \notin H_j$, step S4 of the algorithm will produce a nontrivial divisor of N. This is because $x^{MN/2^{j-1}} \equiv 1 \pmod{N}$ but $x^{MN/2^j} \not\equiv \pm 1 \pmod{N}$ implies that $x^{MN/2^j} \equiv 1 \pmod{p^f}$ for some $p \mid N$ and $x^{MN/2^j} \equiv -1 \pmod{q^e}$ for some $q \mid N$. The conclusion is that at least 50% of all $a \in \mathbb{Z}_N^*$ will lead to a splitting of N in step S4.

This completes the proof of Lemma A. □

(We remark parenthetically that algorithm S works even if step S2 is replaced by

S2′. Set $Q \leftarrow M$.

However, with S2′, the second corollary below might not be true, and this corollary is needed later in the paper.)

To complete the proof of Theorem 3, it suffices to verify that all the steps of Algorithm S can be done in polynomial time. This is left to the reader. □

We immediately get the following corollary:

COROLLARY. *The functions $\varphi(N)$, $\lambda(N)$, and $\lambda'(N)$ are randomly equivalent to factoring.*

The following corollary will be needed in § 4.

COROLLARY. *Suppose we replace step* S1 *of Algorithm* S *with*

S1′. *If* $\gcd(a, N) = r$ *and* $r \neq 1$, *then return nothing and stop.*

Then if d is a factor of N produced by the new algorithm, there is some prime $q \mid N$ such that $q \nmid d$.

Proof. Just check the proof of Theorem 3. □

3. Splitting N using $\sigma(N)$: the square-free case. In this section we assume that

$$N = p_1 p_2 \cdots p_k$$

is the product of one or more distinct primes. This case is somewhat easier than the case where N is divisible by a square, so we give our proofs in detail.

The following procedure will state that N is prime, or with high probability produce a nontrivial divisor of N.

(By iteration, if necessary, we eventually produce the complete factorization of N.)

ALGORITHM A. [*Given $\sigma(N)$ with N squarefree, try to split N.*]

A0. If $\sigma(N) = N+1$, say "prime" and stop.

A1. If N is even, output the factor 2 and stop.

Repeat until N splits:

A2. Run a single iteration of Algorithm S described in § 2 above, using an a chosen at random, and $M = \sigma(N)$. If a nontrivial divisor of N is produced, output that divisor and stop.

A3. Choose a random monic quadratic polynomial from $\mathbb{Z}_N[X]$, say, $f(X) = X^2 + bX + c$.

A4. Choose a random linear polynomial from $\mathbb{Z}_N[X]$, say, $r(X) = tX + u$ such that t and u are not both 0.

A5. [*Ensure that $r(X) \not\equiv 0 \pmod{q}$ for all primes $q \mid N$*]. If $\gcd(t, N)$ splits N, output that divisor and stop.

A6. Compute $dX + e = r(X)^{\sigma(N)} \pmod{(f(X), N)}$.

A7. If $\gcd(d, N)$ splits N, output that divisor and stop.

A8. [*Failure.*] No divisor of N has been produced on this iteration.

In our analysis of Algorithm A, we will find the following group-theoretic lemma useful:

LEMMA B. *Let G be a finite cyclic group, $|G| = n$. Let σ be the homomorphism defined by $\sigma(g) = g^r$. Then $\sigma(G)$ is also a finite cyclic group. We have $|\sigma(G)| = n/\gcd(n, r)$ and if $g' \in \sigma(G)$ then $|\sigma^{-1}(g')| = \gcd(n, r)$. Hence $\sigma(G)$ is the trivial group iff $n \mid r$.*

Proof. See, for example, [Alb, Thm. 23, p. 19]. □

Here are the ideas behind Algorithm A:

Steps A0 and A1 are self-explanatory.

In step A2, if for any p_i dividing N we have $p_i - 1 \mid \sigma(N)$ then Algorithm S will split N in polynomial time.

Hence let us assume that for all p_i we have $p_i - 1 \nmid \sigma(N)$. Pick a p_i and call it p.

Suppose $f(X)$ is a monic quadratic polynomial chosen at random from $\mathbb{Z}_N[X]$. Then a simple argument shows that with probability

$$\frac{1}{2} \cdot \frac{p-1}{p} \tag{1}$$

$f(X)$ is irreducible (mod p); so assume it is. (In practice, of course, we choose many different f and perform the algorithm on all of them. With high probability, the algorithm succeeds somewhere.)

Similarly, for a prime q, with probability

$$\frac{1}{2} \cdot \frac{q-1}{q} \tag{2}$$

$f(X)$ splits as the product of distinct linear factors (mod q), say $f(X) = (X - \beta)(X - \gamma)$ (mod q), so assume it does for some $p_j \neq p_i$ (call it q).

LEMMA C. *With probability at least $\frac{1}{2}$, $\gcd(d, N)$ splits N.*

Proof. We show that we always have $d \equiv 0 \pmod{p}$ but $d \not\equiv 0 \pmod{q}$ with probability $\geqq \frac{1}{2}$. From this we conclude that $\gcd(d, N)$ splits N with probability $\geqq \frac{1}{2}$.

To see that $d \equiv 0 \pmod{p}$ it is enough to see that

$$r(X)^{p+1} (\bmod f(X)) \in \mathbb{Z}_p.$$

Now $f(X)$ is irreducible (mod p); hence $\mathbb{Z}_p[X]/(f(X)) \cong GF(p^2)$. Now the pth power automorphism gives the conjugate of the element $r(X)$ in $GF(p^2)$, so $r(X)^{p+1} = N_{GF(p^2)/GF(p)}(r(X))$ lies in the base field $GF(p)$ (see [Mar]). Thus $d \equiv 0 \pmod{p}$.

Now let us show that $d \not\equiv 0$ with probability $\geqq \frac{1}{2}$. By the Chinese Remainder Theorem, we have the isomorphism

$$\mathbb{Z}_q[X]/(f(X)) \cong \mathbb{Z}_q[X]/(X - \beta) \oplus \mathbb{Z}_q[X]/(X - \gamma).$$

Indeed, we can make this isomorphism explicit. There exist fixed $w_1 X + w_2$ and $v_1 X + v_2 \in \mathbb{Z}_q[X]$ such that every linear $r(X) \in \mathbb{Z}_q(X]$ can be written uniquely as

$$r(X) \equiv c_1(w_1 X + w_2) + c_2(v_1 X + v_2) \pmod{q}. \tag{3}$$

Here the c_1 and c_2 are in $\mathbb{Z}_q$ and depend on $r(X)$. If c_1 and c_2 are both congruent to 0 (mod q), then step A5 of the algorithm above splits N, so we may assume that c_1 and c_2 are not both 0 (mod q).

Now

$$r(X)^{\sigma(N)} \equiv c_1^{\sigma(N)}(w_1 X + w_2) + c_2^{\sigma(N)}(v_1 X + v_2) \pmod{q}$$

so that

$$d \equiv c_1^{\sigma(N)} w_1 + c_2^{\sigma(N)} v_1 \pmod{q}.$$

It is easy to see that $w_1, v_1 \not\equiv 0 \pmod{q}$, so if $d \equiv 0$ we must have

$$-c_1^{\sigma(N)} w_1 v_1^{-1} \equiv c_2^{\sigma(N)} \pmod{q}. \tag{4}$$

We count the number of pairs (c_1, c_2) for which this can happen and show that for each $c_1 \pmod q$ at most $\frac{1}{2}$ the values c_2 satisfy (4). If $c_1 \equiv 0$, then for (4) to hold we must have $c_2 \equiv 0$. If $c_2 \not\equiv 0 \pmod q$ then we may apply Lemma B to see that for any fixed value of c_1, the number of c_2 satisfying (4) is $\gcd(q-1, \sigma(N))$. But since $q-1 \nmid \sigma(N)$, this is $\leqq (q-1)/2$. Hence the total number of nonzero pairs for which (4) can hold is $\leqq (q-1)^2/2$. Dividing this by q^2-1 (total pairs (c_1, c_2) with c_1, c_2 not both 0), we get $d \equiv 0 \pmod q$ with probability $\leqq \frac{1}{2}(q-1)/(q+1)$. Hence with probability $\geqq \frac{1}{2}$, we have $d \not\equiv 0 \pmod q$.

This completes the proof of Lemma C. □

THEOREM 4. *Suppose N is odd, squarefree, and not prime. If $\sigma(N)$ is given, then with probability at least* 1/15, *a single iteration of steps* A2 *through* A7 *splits N.*

Proof. We multiply the probabilities given in (1) and (2) (using the worst case $p=5$, $q=3$) by the likelihood that step A7 splits N to get the worst case probability 1/15. □

A brief remark is in order. Algorithm A will work even if we have a nonzero *multiple* of $\sigma(N)$ instead of $\sigma(N)$ itself. The only difference is that in step A0 we must use a random polynomial-time test on N; for example, the probabilistic test given in [SS].

4. Factoring N using $\sigma(N)$: the general case. This section serves two purposes: we generalize the algorithm in § 3 to the case when N is not necessarily squarefree, and we show how to obtain the *complete* factorization of N, using only the single quantity $\sigma(N)$. Roughly speaking, this has the following complexity-theoretic import: the function "prime factorization" is many-one polynomial-time reducible to the function σ, not just Turing-reducible as one would first suppose.

For now, assume that we merely want to split $N = p_1^{e_1} \cdots p_k^{e_k}$. The algorithm below does this, using a guess α for one of the e_i's. Since $e_i \leqq \log_2 N$, we can try all possible α's without spoiling the polynomial time bound.

ALGORITHM B. [*Try to split N given $\sigma(N)$ and α*]:
B0. If N is a prime power, output $N = p^k$ and stop.
B1. If N is even, output a relatively prime factorization $N = 2^k \cdot M$ and stop.
Repeat until N splits:
B2. Try to split N using the Algorithm S from § 2, using $M = \sigma(N)$. If a nontrivial factor is obtained, output that factor and stop.
B3. Choose a random monic polynomial $f(X) \in \mathbb{Z}_N[X]$ of degree $\alpha+1$.
B4. Choose a random polynomial $r(X) \in \mathbb{Z}_N[X]$ of degree $\leqq \alpha$.
B5. Compute $h(X) = d_\alpha X^\alpha + \cdots + d_1 X + e = r(X)^{\sigma(N)} \pmod{f(X)}$.
B6. For each i, $1 \leqq i \leqq \alpha$, let $g_i = \gcd(d_i, N)$.
B7. If for some i, $1 < g_i < N$, output g_i and stop.

We hope that $f(X)$ is irreducible (mod p), but has at least two distinct irreducible factors (mod q). If this is the case, we call $f(X)$ *suitable*, and write

$$f(X) = \prod_{i=1}^{s} f_i(X)^{d_i}$$

with each $f_i(X)$ irreducible, $\deg(f_i) = k_i$, and $s \geqq 2$. There is then a surjective ring homomorphism

$$\varphi : \mathbb{Z}_q[X]/(f(X)) \to \mathrm{GF}(q^{k_1}) \oplus \cdots \oplus \mathrm{GF}(q^{k_s}) \tag{5}$$

(by the Chinese Remainder Theorem). We let K denote the kernel of φ, and let

$$\varphi_i : \mathbb{Z}_q[X]/(f(X)) \to \mathrm{GF}(q^{k_i})$$

denote the ith projection map. The interesting fact about these projections is

LEMMA E. *Let $h(X) \in \mathbb{Z}_q[X]/(f)$ have degree $\leqq \alpha$. If for some $i<j$, $\varphi_i(h) \neq \varphi_j(h)$, then one of h's nonconstant coefficients is relatively prime to q.*

Proof. Assume that all of h's positive-degree coefficients vanish mod q. Then h is an element of $\mathrm{GF}(q)$, which is unchanged by every φ_i. The result follows by contraposition. □

We now need two probability estimates:

LEMMA F. *A monic polynomial $f(X) \in \mathbb{Z}_N[X]$ of degree $\alpha+1$ is suitable with probability at least*

$$\frac{1}{\alpha+1}\left(1-\frac{1}{\sqrt{p}}\right)\left(1-\frac{1}{q}-\frac{1}{\alpha+1}\right).$$

Proof. First, $f(X)$ is irreducible (mod p) with probability at least $(1-1/\sqrt{p})/(\alpha+1)$. Second, f is irreducible (mod q) with probability at most $1/(\alpha+1)$, and has a repeated factor (mod q) with probability exactly $1/q$ (see [Berl, p. 80] and [Carl]). □

LEMMA G. *If $f(X)$ is suitable, then $r(X) \notin K$ with probability at least $1-1/q^2$.*

Proof. By the rank-nullity theorem, $\dim_{GF(q)} K + \sum k_i = \alpha+1$. Since there are at least two positive k_i's, the result follows. □

The main result on our algorithm is

THEOREM 5. *If N is not prime, then for some $\alpha \leqq \log_2 N$, a single iteration of steps* B0 *through* B7 *splits N with probability at least $1/32(\alpha+1)$.*

Proof. If N is a prime power or even, we get a nontrivial factorization. Therefore we can assume that N is odd, with two distinct prime factors p and q. If $q-1 \mid \sigma(N)$, then by Theorem 2, step B2 will split N, so we can assume further that $q-1 \nmid \sigma(N)$.

Now let $p^\alpha \parallel N$; $\alpha \leqq \log_2 N$ as claimed. Assume for now that $f(X)$ is suitable and that $r(X) \notin K$; we will estimate the probability that for some i, $g_i \equiv 0 \pmod p$ and $g_i \not\equiv 0 \pmod q$.

First, since f is suitable, $g_i \equiv 0 \pmod p$ for all i, since $\sigma(N)$ is a multiple of $p^\alpha + \cdots + p + 1$, the annihilator of $\mathrm{GF}(p^{\alpha+1})^*/\mathrm{GF}(p)^*$.

Now consider the situation (mod q), and let $c_i = \varphi_i(r^{\sigma(N)})$. By the hypothesis that $r \notin K$, some $c_i \neq 0$; if some other $c_j = 0$, then by Lemma E we must split N at step B7. Therefore we may as well assume that all the c_i's are nonzero, or, what is the same thing, $r(X)$ is a unit mod $f(X)$. Since we have assumed that $q-1 \nmid \sigma(N)$, the map $r(X) \to \varphi(r(X)^{\sigma(N)})$ does not annihilate $\mathbb{Z}_q[X]/(f(X))^*$. The image of this homomorphism is then a direct product of nontrivial cyclic groups, say $C_1 \times C_2 \times \cdots \times C_s$. The probability that a random element $(c_1, \cdots, c_s)$ will have all components equal is at most $1/\mathrm{lcm}\,\#(C_i) \leqq 1/2$; by Lemma E, then, the probability that some $g_i \not\equiv 0 \pmod q$ is at least $\frac{1}{2}$.

Theorem 5 now follows by combining the last two paragraphs, Lemmas F and G, and the estimates $p, q \geqq 3$, $\alpha \geqq 1$. □

We now turn to the problem of complete factorization. Our first observation is that $\sigma(N)$ can be replaced by any multiple of $\sigma(N)$ with no change in the statement of Theorem 5. Since $\sigma(N_1 N_2) = \sigma(N_1)\sigma(N_2)$ for relatively prime N_1 and N_2, we can use $\sigma(N)$ to recursively factor the pieces produced by Algorithm B, *provided* they are relatively prime. Therefore we need to transform the output of Algorithm B into a list of pairwise coprime factors.

Our solution to this problem hinges on the following concept. We say that a factorization $N = N_1 N_2 \cdots N_r$ *segregates* p if $v_p(N_i) = v_p(N)$ for some i. A factorization segregates every prime if and only if the elements are pairwise relatively prime, and in this case $\sigma(N_i) \mid \sigma(N)$. The procedure below produces such a factorization, provided that some prime is segregated to begin with.

ALGORITHM R. [*Factor refinement procedure*].
[*At all times we have* $N = N_1^{e_1} \cdots N_r^{e_r}$, *possibly needing further processing.*]
R0. Let the initial factorization be $N = N_1 N_2$.
While factors remain with $\gcd(N_i, N_j) > 1$:
R1. Set $D = \gcd(N_i, N_j)$.
R2. Replace $N_i^{e_i}$, $N_j^{e_j}$ in the list by $D^{e_i+e_j}$, $(N_i/D)^{e_i}$, $(N_j/D)^{e_j}$.
R3. If necessary, remove units from the list and combine powers of equal numbers.

The properties of this procedure are given by

LEMMA D. *Algorithm* R *terminates in at most* $\log_2 N$ *iterations, with all the* N_i's *relatively prime. If the initial factorization is nontrivial and segregates some* $p \mid N$, *then on termination there are at least two factors.*

Proof. Left to the reader. □

It remains to show that using a multiple of $\sigma(N)$, we can split N and segregate some prime. This follows from the proof of Theorem 5 (recall that $q \nmid d_i$, so q is segregated) and the corollary to Theorem 3.

Thus we have completed the proof of Theorem 2, which we restate here:

THEOREM 2. *Given* $\sigma(N)$, *we can produce the complete factorization of* N *in random polynomial time.*

COROLLARY. *Computing the function* $r_4(N)$, *the number of ways to write* N *as the sum of four integer squares, is (randomly) equivalent to factoring.*

Proof. Suppose

$$N = 2^{e_1} p_2^{e_2} \cdots p_k^{e_k}.$$

Then a classical theorem of Lagrange (see [HW, Thm. 386]) says

$$r_4(N) = \begin{cases} 8\sigma(N) & \text{if } e_1 = 0, 1, \\ 24\dfrac{\sigma(N)}{2^{e_1+1}-1}, & e_1 \geqq 2. \end{cases}$$

Since computing $\sigma(N)$ is randomly equivalent to factoring, the result follows. □

Similar results can be proved for functions like $r_8(N)$.

5. Generalization to $\sigma_k(N)$. A natural generalization of $\sigma(N)$ is the sum of the kth powers of divisors of N, i.e.

$$\sigma_k(N) = \sum_{d \mid N} d^k = (p_1^{ke_1} + \cdots + p_1^k + 1) \cdots (p_r^{ke_r} + \cdots + p_r^k + 1)$$

where $N = p_1^{e_1} \cdots p_r^{e_r}$.

We also have a corresponding generalization regarding its computational complexity.

THEOREM 6. *For any fixed integer* $k \neq 0$, *computing* $\sigma_k(N)$ *is (randomly) equivalent to factoring.*

Proof. If k is negative, then

$$\sigma_k(N) = \frac{\sigma_{-k}(N)}{N}$$

so it suffices to consider positive k.

The essential idea is that the map $x \to x^{\sigma_k(N)}$ takes $\mathrm{GF}(p^{k(\alpha+1)})$ into $\mathrm{GF}(p^k)$, when $p^\alpha \| N$.

ALGORITHM C. [*Try to split N given* $\sigma_k(N)$]:

C0. If N is even or a prime power, output a factor and stop.

Set $\alpha \leftarrow 1$, and repeat until N splits:

C1. Try to split N using Algorithm S with $M = \sigma_k(N)$.

C2. [*Construct* $\mathrm{GF}(p^k)$.] Pick a random monic polynomial $h(Y) \in \mathbb{Z}_N[Y]$ of degree k; let R denote $\mathbb{Z}_N[Y]/(h(Y))$.

C3. Pick a random monic $f(X) \in R[X]$ of degree $\alpha+1$.

C4. Pick a random $r(X) \in R[X]$ of degree $\leqq \alpha$.

C5. Compute $h(X) = d_\alpha(Y)X^\alpha + \cdots + d_1(Y)X + e(Y) = r(X)^{\sigma(N)} \pmod{f(X)}$.

C6. For each i, $1 \leqq i \leqq \alpha$, and each coefficient t of $d_i(Y)$, see if $\gcd(t, N)$ splits N.

C7. [*Failure*]. If $\alpha+1 < B$, where B is a bound on the exponents in the prime factorization of N, set $\alpha \leftarrow \alpha+1$; else set $\alpha \leftarrow 1$. (We may take $B = \log_3(N)$.)

There is only one new observation to make here: we want $h(Y)$ to be irreducible modulo two distinct divisors of N, and this happens with probability about $1/k^2$. Since $k \leqq \log_2 \sigma_k(N)$, we only expect to wait a polynomial-bounded time until this happens. In all other respects, Algorithm C behaves just like Algorithm B. The details are left to the reader. □

6. Some classes of numbers that can be factored quickly. The reduction of factoring to computing $\sigma(N)$ discussed in the previous sections allows us to quickly factor those numbers N for which $\sigma(N)$ is easily computable.

Consider the equation $\sigma(N) = 2N$. Numbers satisfying this equation are known as *perfect numbers*. The Pythagoreans attributed special properties to such numbers and this led to their intense study in antiquity, culminating in Euclid's proof that numbers of the form $2^{n-1}(2^n-1)$ are perfect when the second factor is prime. In the 18th century, Euler proved that all *even* perfect numbers must be of this form. No one knows if there are any odd perfect numbers, but if there are, they must satisfy many stringent conditions (see, e.g., [teR]). We now add one more: they are all easy to factor!

More precisely, we show that the set {*perfect numbers*}, defined to be

$$\{x \in (0,1)^* : x \text{ (interpreted in binary) is perfect}\},$$

is recognizable in (two-sided) random polynomial time, i.e. is a member of the complexity class BPP.

THEOREM 7. {*perfect numbers*} $\in$ BPP.

Proof. Given N, *assume* that $\sigma(N) = 2N$. Run the algorithm of §§ 3–4 with the appropriate polynomial time bound; the result is a (purportedly complete) factorization of N. Now check to see if N is indeed perfect by using equation (0).

We end up accepting N if N is perfect, or if we accidentally produced an incorrect factorization (i.e. one where our probabilistic prime test said all the factors were prime, but some really weren't). But such an accident happens only ε of the time, and we can fix ε ahead of time.

We end up rejecting N if N is not perfect, or if we accidentally produced an incorrect factorization as above, or if the algorithm of §§ 3-4 failed to produce any factorization at all in our (pre-fixed) time bound. Again, this happens only ε of the time. □

Theorem 7 gives the first "natural" set in BPP which is not known to be in RP. Of course, it is possible to construct examples like

$$L = \{x \# y: x \text{ is prime and } y \text{ is composite}\}.$$

$L \in$ BPP, but it is somewhat "artificial", since it may be written as the product of two languages, one of which is known to be in RP, and one which is known to be in co-RP.

Nevertheless, Theorem 7 is very likely less interesting than it appears at first glance; if there are no odd perfect numbers (as is widely believed), then the clever Lucas-Lehmer test (see [Knu]) combined with the Euclid-Euler result for even perfect numbers gives a *deterministic* polynomial time algorithm to recognize {*perfect numbers*}.

However, there are well studied generalizations of perfect numbers for which no deterministic tests are known. For example, numbers such that $\sigma(N) = 3N$ are sometimes called *sous-doubles*; examples are 120 and 672. It is easy to see that an argument like that in Theorem 7 shows that {*sous-doubles*} $\in$ BPP.

A larger class is the set of *multiply perfect numbers*; i.e., those numbers N for which $N \mid \sigma(N)$. To show that {*multiply perfect numbers*} $\in$ BPP, we need the following lemma:

LEMMA J.

$$\sigma(N) < 5N \ln \ln N \quad \textit{for } N \geqq 3.$$

Proof. A well-known theorem (e.g. [HW, Thm. 329]) states that

$$\frac{\sigma(N)\phi(N)}{N^2} \geqq 1.$$

A result of Rosser and Schoenfeld [RS] is

$$\frac{N}{\phi(N)} < e^C \ln \ln N + \frac{3}{\ln \ln N}$$

for $N \geqq 3$. Here C is Euler's constant, approximately .5772.

Combining these two inequalities, we get

$$\frac{\sigma(N)}{N} < (e^C + 3) \ln \ln N$$

for $N > e^e$. From this, the result easily follows. □

Lemma J shows that we can determine if N is multiply perfect with fewer than $5 \ln \ln N$ invocations of Algorithm B. This can be done in random polynomial time, so we have proved

THEOREM 8. {*multiply perfect numbers*} $\in$ BPP.

Carmichael [Carm] found the multiply perfect numbers less than 10^9:

1, 6, 28, 120, 496, 672, 8128, 30240, 32760, 523776,

2178540, 23569920, 33550336, 45532800, 142990848, 459818240.

(We have corrected several mistakes in Carmichael's original list.) It is not known whether or not there are infinitely many multiply perfect numbers. However, there are

some density results that give upper bounds; for example, Hornfeck and Wirsing have shown [HoW] that if $m(x)$ denotes the number of multiply perfect numbers $\leqq x$, then

$$m(x) = O\left(\exp\left(\frac{c \ln x \ln\ln\ln x}{\ln\ln x}\right)\right).$$

To give still another example, consider the pairs (M, N) such that

$$\sigma(M) = \sigma(N) = M + N.$$

Such numbers are known as *amicable pairs*; the smallest pair is (220, 284). Jacob gave Esau 220 goats and 220 sheep [God], and some scholars have interpreted this as showing that the ancient Hebrews knew about $\sigma(N)$. There is an enormous literature concerning amicable pairs (see [LM]). An argument similar to those above gives

THEOREM 9. $\{$*amicable pairs*$\} \in$ BPP.

It is not known whether or not there are infinitely many amicable pairs (M, N), but Erdös conjectures that the number of such pairs with $M < N < x$ is at least $cx^{1-\varepsilon}$ [Guy2].

Using our methods, it is possible to show that many other types of numbers (for example, the "betrothed numbers" of Isaacs [Guy2, p. 33]) can be recognized in two-sided random polynomial time.[1]

In Theorems 7–9 above, we have given three sets in BPP. The two-sidedness of these sets is due to the dependence on primality testing; if we had a deterministic polynomial-time prime test, we would be able to show that $\{$*perfect numbers*$\}$, $\{$*multiply perfect numbers*$\}$, and $\{$*amicable numbers*$\}$ are in RP. No such prime test is currently known, although there is one due to Adleman, Pomerance, and Rumely [APR] which runs in time $O((\log N)^{c \log\log\log N})$.

7. Epilogue. In § 2, we showed how to split N given a multiple of $p-1$. The results on $\sigma(N)$ can be phrased similarly; if we know a multiple of $p+1$ (or p^2+p+1, etc.) we can split N. This leads to the question: for which polynomials $f(p)$ do there exist fast algorithms for splitting N? We will address this question in a future paper [BS].

The complexity of several number-theoretic functions is still open. One example is computing discrete logarithms (mod p).

Not every difficult number-theory function is equivalent to factoring; some are apparently *harder*. For example, remarks of Shanks indicate that factorization is reducible to finding the class number of an imaginary quadratic field [Shan] but no reduction in the other direction is known, nor is it even clear that this problem is in NP.

Acknowledgments. Much of the research for this paper was done while the first and third authors were graduate students at the University of California at Berkeley; they would like to express their deep appreciation to Manuel Blum, who created an environment eminently suitable to conducting research.

We are pleased to acknowledge the use of the computer algebra program VAXIMA, which allowed us to confront our early ideas with the harsh reality of specific examples.

Thanks also go to the referees, especially to one who produced a particularly thorough list of improvements.

[1] Betrothed pairs (M, N) satisfy $\sigma(M) = \sigma(N) = M + N + 1$; we note reluctantly that a pair cannot be both betrothed and amicable.

REFERENCES

[Alb] A. A. ALBERT, *Fundamental Concepts of Higher Algebra*, Univ. Chicago Press, Chicago, IL, 1956.

[APR] L. M. ADLEMAN, C. POMERANCE AND R. S. RUMELY, *On distinguishing prime numbers from composite numbers*, Ann. Math., 117 (1983), pp. 173–206.

[Bach] E. BACH, *Discrete logarithms and factoring*, Computer Science Division Report UCB/CSD/84/186, Univ. California, Berkeley 1984.

[BMS] E. BACH, G. MILLER AND J. SHALLIT, *Sums of divisors, perfect numbers, and factoring*, Proc. 16th Annual ACM Symposium on The Theory of Computing (1984), pp. 183–190.

[BS] E. BACH AND J. SHALLIT, *Factoring with cyclotomic polynomials*, 26th IEEE Symposium on Foundations of Computer Science, 1985, pp. 443–450.

[Berl] E. R. BERLEKAMP, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[Carl] L. CARLITZ, *The arithmetic of polynomials in a Galois field*, Amer. J. Math., 54 (1932), pp. 39–50.

[Carm] R. D. CARMICHAEL, *A table of multiply perfect numbers*, Bull. Amer. Math. Soc., 13 (1907), pp. 383–386.

[Dix] J. D. DIXON, *Asymptotically fast factorization of integers*, Math. Comp., 36 (1981), pp. 255–260.

[Gill] J. GILL, *Computational complexity of probabilistic Turing machines*, this Journal, 6 (1977), pp. 675–695.

[God] *Genesis*, xxxii, 14.

[Guy1] R. K. GUY, *How to factor a number*, Proc. Fifth Manitoba Conference on Numerical Mathematics, Winnipeg, 1976, pp. 49–89.

[Guy2] R. K. GUY, *Unsolved Problems in Number Theory*, Springer-Verlag, New York, 1981.

[HW] G. H. HARDY AND E. M. WRIGHT, *An Introduction to the Theory of Numbers*, Oxford University Press, Oxford, 1971.

[HoW] B. HORNFECK AND E. WIRSING, *Uber die Haufigkeit vollkommener Zahlen*, Math. Ann. 133 (1957), pp. 431–438.

[Knu] D. E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, 2nd ed. Addision-Wesley, Reading, MA, 1981, pp. 391–394.

[LM] E. J. LEE AND J. S. MADACHY, *The history and discovery of amicable numbers*, J. Rec. Math., 5 (1972), pp. 77–93, 153–173, 231–249.

[Len] H. W. LENSTRA, JR., *Elliptic curve factorization*, typewritten ms., February 14, 1985.

[Long] D. L. LONG, *Random equivalence of factorization and computation of orders*, Theoret. Comput. Sci., to appear.

[Mar] D. A. MARCUS, *Number Fields*, Springer-Verlag, New York, 1977.

[Mill] G. MILLER, *Riemann's hypothesis and tests for primality*, J. Comp. System Sci., 13 (1976), pp. 300–317.

[Pol] J. M. POLLARD, *Theorems on factorization and primality testing*, Proc. Cambridge Phil. Soc., 76 (1974), pp. 521–528.

[RS] J. B. ROSSER AND L. SCHOENFELD, *Approximate formulas for some functions of prime numbers*, Illinois J. Math., 6 (1962), pp. 64–94.

[Shan] D. SHANKS, *Class number, a theory of factorization, and genera*, Proc. of Symposia in Pure Mathematics, V. 20 (1969 Number Theory Institute), American Mathematical Society, Providence, RI, 1971, pp. 415–440.

[SS] R. SOLOVAY AND V. STRASSEN, *A fast Monte-Carlo test for primality*, this Journal, 6 (1977), pp. 84–85.

[teR] H. J. TE RIELE, *Perfect numbers and aliquot sequences*, in Computational Methods in Number Theory, Amsterdam Math. Centre Tracts, 154, 1982, pp. 141–157.

SIAM J. COMPUT.
Vol. 15, No. 4, November 1986
©1986 Society for Industrial and Applied Mathematics
022

COMPLETION OF A SET OF RULES MODULO A SET OF EQUATIONS*

JEAN-PIERRE JOUANNAUD† AND HÉLÈNE KIRCHNER†

Abstract. Abstract Church-Rosser properties are first presented, depending on an arbitrary relation R, an equivalence relation E and a reduction relation R^E used to compute normal forms of R modulo E. Terminating rewriting systems operating on equational congruence classes of terms of a free algebra are then considered. In this framework, the Church-Rosser property is proved decidable for a very general reduction relation which may take into account the left-linearity of rules for efficiency reasons, under the only assumption of existence of a complete and finite unification algorithm for the underlying equational theory, whose congruence classes are assumed to be finite. This extends previous results by Lankford and Ballantyne, Peterson and Stickel, Huet, Jouannaud. A general completion procedure for mixed sets of rules and equations is then presented that generalizes and improves Peterson and Stickel's one. In addition to computing a Church-Rosser set of rules when it terminates, it yields a semi-decision procedure for testing equality when it runs forever. Finally a post-processor is described that yields a Church-Rosser set of inter-reduced rules. All proofs, including the correctness proof of our completion algorithm, are based on the powerful proof technique of multiset induction.

Key words. Church-Rosser property, confluence, coherence, equational theories, equational deduction, term rewriting systems, reduction systems, critical pairs, complete sets of unifiers, completion procedure, multiset induction

AMS(MOS) subject classifications. 03B25, 08B05

1. Introduction.

Term rewriting systems (TRS for short) are known to be a major tool for expressing nondeterministic computations, because they are based on directed equalities, with no explicit control. In addition, they have very simple semantics, whenever the result of a computation does not depend on the choice of the rules to be applied (the so-called *confluence* property). When a TRS is not confluent, it may be transformed into an equivalent confluent one, using the Knuth-Bendix *completion procedure* [54]. This procedure can be seen as a way to compile equational specifications into confluent sets of rules, providing *rewrite programs*, which can be used in a similar way as PROLOG ones [11].

During the past several years, confluent TRS and the Knuth-Bendix completion procedure were shown to be a major tool for a wide class of problems, mainly the word problem in universal algebra [17], [18], [54], [3], [5], [81], [49], the word problem for finitely presented algebras [2], [4], [60], equivalence proofs of sets of axioms in algebra [61], [62], unification in equational theories [22], [36], [80], [45], software validation [28], inductive proofs in data types [68], [25], [34], [50], theorem proving in various logics [72], [29], [30], [24], program synthesis from specification [31], [11], computing with rewrite programs [11] or with Horn clauses with equality [26], describing PROLOG's semantics [8], constructing initial algebras and executing equational specifications as in OBJ [27], inferring types in the presence of polymorphism [67], code optimization in compilers [1] or even code generation thought of as tree rewriting [79].

* Received by the editors March 20, 1984, and in revised form January 24, 1986. A preliminary version of this paper was presented at the 11th ACM Conference on Principles of Programming Languages, Salt Lake City, January 1984. This research was supported in part by Agence pour le Développement de l'Informatique under contract 82/767 and in part by the Office of Naval Research under contract N00014-82-0333. Part of this work was done while the first author was visiting the Computer Science Laboratory of SRI-International, Menlo Park, California 94025.

† Greco-Programmation and Centre de Recherche en Informatique de Nancy, Campus Scientifique, BP 239, 54506 Vandoeuvre les Nancy, Cedex, France.

The Knuth-Bendix completion procedure is based on using *equations* as *rewrite rules* and computing *critical pairs* when left-hand sides of rules overlap. If a critical pair has distinct *irreducible forms*, a new rule must be added and the procedure recursively applies until it maybe stops. This procedure requires the *termination* property of the set of rules, which can be proved by various tools, such as the recursive path ordering [9], the generalized recursive path ordering [47], the recursive decomposition ordering [46], the recursive decomposition ordering with status [63] or polynomial orderings [55]. A full implementation of most of these techniques is described in [61].

The method was extended to handle the case of an *equational term rewriting system* (ETRS for short), i.e. a set of axioms split into a first subset R whose axioms are used as rules and a second subset E whose axioms are used as equations. This allows the inclusion of axioms, such as commutativity, that cannot be used as rules without losing the termination property. A first approach by Lankford and Ballantyne [56], [57], [58] handles the case of commutative, or more generally permutative, axioms that generate finite E-congruence classes. The case of infinite E-congruence classes is studied in [73], [32], [40]. Huet's approach [32] is restricted to sets R of left-linear rules, while Peterson and Stickel's one [73] is restricted to linear theories E for which a finite and complete unification algorithm is known. Both [58] and [73] results yield an efficient associative-commutative completion procedure, with a few more restrictions needed for applying the first approach. A complete unification algorithm is required to compute complete sets of E-overlappings, providing complete sets of E-critical pairs. These results are unified in [40] by describing at a more abstract level the underlying model of computation used in both approaches: it makes clear that two properties, namely *confluence* and *coherence*, are necessary and sufficient conditions for Church-Rosser properties of this model, assuming the so-called *E-termination* property of the rewriting relation modulo the set of equations. In addition to confluence, coherence is required to enable computations in E-congruence classes. Applying then this abstract model to ETRS, Huet's results are easily obtained as well as a more general version of Peterson and Stickel's ones: nonlinear equational theories can be handled, which was known as a hard problem in the theory of ETRS.

In this paper, we refine our abstract approach and provide simpler definitions. In addition to classical Church-Rosser and confluence notions using E-congruence classes, we also introduce new Church-Rosser, confluence and coherence notions which are parameterized by the reduction relation R^E used to reduce terms. This allows us to deal with efficient reduction relations, whereas R/E, the reduction relation induced by the rules in E-congruence classes, requires one to search the congruence class for a reducible term. As in [40], the coherence property is the necessary and sufficient condition for R/E and R^E to have the same normal forms. We then study a very general reduction relation, which is a mixture of the usual reduction relation for left-linear rules and Peterson and Stickel's reduction relation using matching modulo E for the others. This was already done in [40] and allows us to avoid useless computations of complete sets of E-critical pairs when left-linear rules are involved. This is a major improvement in practice, because using E-unification or E-matching algorithms (such as the associative-commutative ones of [82] and [37]) is known to be time consuming. On the other hand, the more powerful the rewriting relation, the weaker (and therefore the easier to ensure) the associated Church-Rosser property will be. As a consequence, given a set E of equations, the same set R rules can be Church-Rosser for some reduction relation but not for a weaker one.

For handling our definitions, a new, powerful proof technique is required, based on *multiset induction*, which permits us to prove our Church-Rosser results. In addition,

this proof technique allows us to resolve a problem left open in [73] and [40]: assume that the reduction relation induced in E-congruence classes is terminating. Then the Church-Rosser property is shown equivalent to the E-equality of normal forms of all critical pairs and is thus decidable when these pairs are finitely many, which is the case for finite ETRS. Such a result was already known when R^E is the usual rewriting relation, assuming the rules are left-linear [32]. We prove it here for the case when R^E is the mixed rewriting relation, assuming a complete and finite unification algorithm is known for E, whose congruence classes must also be finite. Our proof holds for the particular case of the Peterson and Stickel's rewriting relation, without any linearity hypothesis on rules or on equations. However, the case of infinite congruence classes remains as the last open problem of the theory of ETRS.

A new completion procedure is then derived from these Church-Rosser results. It is written in a recursive form, which makes both understanding and proof easier. This procedure improves and generalizes Peterson and Stickel's one for associative-commutative theories. Two kinds of critical pairs are to be distinguished: *confluence pairs* are processed in the usual way, whereas *coherence pairs* are processed in a more complex way. In addition, *extended rules* are sometimes used instead of coherence pairs to ensure the coherence property. These rules generalize to an arbitrary theory E the so-called *associative-commutative extensions* of [73]. Following [33], a complete proof of our algorithm is then given using multiset induction. The case where the algorithm stops without adding any rule provides in addition another proof of our Church-Rosser results. We finally define *complete sets of normalized reductions* (i.e. Church-Rosser sets of rules which are inter-reduced) and show how to transform a given Church-Rosser set of rules into a unique complete set of normalized reductions. These sets of rules can be used to check whether two specifications are actually different presentations of the same theory.

2. Abstract Church–Rosser results. The presentation of this section is strongly influenced by that in [32]: we deal with binary relations on an arbitrary set $\mathcal{S}$. As in [32], most of our proofs will use *noetherian induction* (see [6] for a precise introduction): given a noetherian relation on $\mathcal{S}$ (i.e. a relation without any infinite decreasing chain), we prove a property $\mathcal{P}$ on $\mathcal{S}$ by showing that for any t in $\mathcal{S}$, $\mathcal{P}(t)$ follows from the assumption that $\mathcal{P}$ is true for all elements in $\mathcal{S}$ which are proper descendents of t for the given noetherian relation. We use here what we call *multiset induction*, a particular case of noetherian induction, that we introduce in the following.

2.1. Preliminary definitions and notations. Let $\vdash\!\dashv^E$ be a symmetric relation on $\mathcal{S}$ and $\sim^E$ its reflexive transitive closure, called E-equality, which is obviously an equivalence relation. For simplicity, the E subscript will be omitted if no ambiguity arises.

Let $\rightarrow^R$ (R for short) be any relation on $\mathcal{S}$, called *reduction*, $\overset{+}{\rightarrow}{}^R$ and $\overset{*}{\rightarrow}{}^R$ its transitive and reflexive transitive closure respectively, and $\rightarrow^{R/E}$ (R/E for short) the relation $\sim\circ\rightarrow^R\circ\sim$ which simulates the relation induced by R in E-equivalence classes.

Let $\models\!\dashv$ be the union of $\vdash\!\dashv$ with the symmetric closure of $\rightarrow^R$, that is the smallest symmetric relation containing $\vdash\!\dashv$ and $\rightarrow^R$. Let $\overset{*}{\models\!\dashv}$ be the reflexive transitive closure of $\models\!\dashv$, called $R \cup E$-equality, which is an equivalence relation.

In the following, t (possibly with a subscript or a superscript) denotes an arbitrary element in $\mathcal{S}$ and $\rightarrow$ an arbitrary reduction on $\mathcal{S}$. We say that t is *$\rightarrow$-reducible* iff $t \rightarrow t'$ for some t'. Otherwise t is said to be *$\rightarrow$-irreducible*. A *normal form* of t, denoted $t\downarrow$, is an irreducible term t' such that $t \overset{*}{\rightarrow} t'$. We write $t\downarrow_R$ if we want to make the reduction relation R precise.

The relation R is *terminating modulo* E or E-*terminating* iff the relation R/E is noetherian, i.e. there is no sequence of the form $t_0 \sim t'_0 \to^R t_1 \cdots t_n \sim t'_n \to^R t_{n+1} \cdots$. R is simply said to be *terminating* or *noetherian* or *well-founded* if E is empty. Our particular interest in terminating relations is twofold: first, they provide normal forms for every element, and second, they allow proofs by noetherian induction.

Multiset induction. A *multiset* on $\mathcal{S}$ is a finite unordered collection of elements of $\mathcal{S}$, possibly with repetitions. For instance, $\{1, 2, 1, 3, 15, 15\}$ is a multiset on $\mathcal{N}$, the set of natural numbers. Operations on sets extend in a natural way to multisets, also called bags. Specifically, removing an element from a multiset amounts to removing one of its occurrences in the multiset. In the same way, adding an element to a multiset increases by one its number of occurrences. We can extend any relation $\to$ on $\mathcal{S}$ to a relation $\to_{\text{mult}}$ on $\mathcal{M}(\mathcal{S})$, the set of all multisets on $\mathcal{S}$, in the following way: given two multisets M and N in $\mathcal{M}(\mathcal{S})$, $M \to_{\text{mult}} N$ iff N is obtained from M by removing one of its elements, say t, and adding a finite number of elements t' (possibly with repetitions) such that $t \to t'$. As an easy consequence of Koenig's lemma, the transitive closure of $\to_{\text{mult}}$ is noetherian iff $\to$ is noetherian (see [12] for a precise proof, and [41] for a discussion about multiset orderings). What we call *multiset induction* on $\to$ is simply noetherian induction on the transitive closure of $\to_{\text{mult}}$. All our proofs are based on this technique that appears to be well-suited for proving Church-Rosser properties.

2.2. Church–Rosser definitions. We now begin studying the Church-Rosser property for such reductions on $\mathcal{S}$. Our goal is to decide $R \cup E$-equality of two terms t and t' by computing their normal forms using some reduction relation related to R and E, then checking these normal forms for E-equality (that we implicitly assume decidable for practical use). A very natural idea is to compute in the quotient set of $\mathcal{S}$ modulo E, thus to use R/E as the reduction relation. Let us recall what the Church-Rosser and confluence properties are in the context of E-equivalence classes on $\mathcal{S}$:

DEFINITION 1. R is *Church-Rosser modulo* E iff $\forall t_1, t_2 \quad t_1 \overset{*}{\leftrightarrow} t_2 \Rightarrow \exists t' \quad t_1 \overset{*}{\to}^{R/E} t'$ and $t_2 \overset{*}{\to}^{R/E} t'$. R is *confluent modulo* E iff $\forall t, t_1, t_2 \quad t \overset{*}{\to}^{R/E} t_1$ and $t \overset{*}{\to}^{R/E} t_2 \Rightarrow \exists t' \quad t_1 \overset{*}{\to}^{R/E} t'$ and $t_2 \overset{*}{\to}^{R/E} t'$.

These two properties are nothing but the classical Church-Rosser and confluence properties in the quotient set of $\mathcal{S}$ by $\sim$ and are known to be equivalent. However, R/E-reducibility may be undecidable if E-equivalence classes are infinite and is at least very inefficient if large classes must be traversed to find a reducible term. This problem can be overcome by choosing an element for representing each equivalence class, as in OBJ [27]. This can be done in a canonical way for many practical cases of reductions on terms modulo associativity and commutativity. A more general idea, the key one actually, is to compute using another relation R^E. This yields R^E-dependent definitions that have to be linked with definitions using R/E by introducing a new property called coherence. As a matter of fact, the first point of view is an instance of the second: starting from an element of $\mathcal{S}$, we first compute its canonical form and then reduce it. This is clearly a reduction on $\mathcal{S}$. As the second point of view allows better understanding and more general results, we adopt it here.

FUNDAMENTAL ASSUMPTION. In the following, $\to^{R^E}$ ($\to$ or R^E can also be used for short if no ambiguity arises) is any reduction that satisfies

$$R \subseteq R^E \subseteq R/E.$$

DEFINITION 2. A pair (t_1, t_2) is R^E-*confluent modulo* E, denoted $t_1 \downarrow t_2$, iff $\exists t'_1, t'_2 \quad t_1 \overset{*}{\to}^{R^E} t'_1$, $t_2 \overset{*}{\to}^{R^E} t'_2$ and $t'_1 \sim t'_2$. For $R^E = R/E$, we can assume $t'_1 = t'_2$.

DEFINITION 3.

1) R is *R^E-Church-Rosser modulo E* iff $\forall t_1, t_2 \quad t_1 \overset{*}{\vdash\!\dashv} t_2 \Rightarrow t_1 \downarrow t_2$.

2) R^E is *confluent modulo E* iff $\forall t, t', t'' \quad t \overset{*}{\to}^{R^E} t'$ and $t \overset{*}{\to}^{R^E} t'' \Rightarrow t' \downarrow t''$.

3) R^E is *locally confluent modulo E with R* iff $\forall t, t', t'' \quad t \to^{R^E} t'$ and $t \to^{R} t'' \Rightarrow t' \downarrow t''$.

4) R^E is *coherent modulo E* iff $\forall t, t', t'' \quad t \overset{+}{\to}^{R^E} t'$ and $t \sim t'' \Rightarrow \exists t_1'' \quad t'' \to^{R^E} t_1''$ and $t' \downarrow t_1''$.

5) R^E is *locally coherent modulo E* iff $\exists t, t', t'' \quad t \to^{R^E} t'$ and $t \vdash\!\dashv t'' \Rightarrow \exists t_1''$ $t'' \to^{R^E} t_1''$ and $t' \downarrow t_1''$.

These definitions are pictured in Fig. 2.1.

Let us first point out that the definitions of coherence and locale coherence could be given in a slightly different way, as follows:

4') R^E is coherent modulo E iff $\forall t, t', t'' \quad t \overset{*}{\to}^{R^E} t'$ and $t \sim t'' \Rightarrow t' \downarrow t''$.

5') R^E is locally coherent modulo E iff $\forall t, t', t'' \quad t \to^{R^E} t'$ and $t \vdash\!\dashv t'' \Rightarrow t' \downarrow t''$.

These definitions are simpler than the previous ones and make clear that confluence and coherence are two instances of the same general concept, which expresses that two (maybe different) relations have a "diamond" property. Nevertheless it is easy to check that 4 and 4' on one hand, 5 and 5' on the other hand are actually equivalent, provided the termination of the relation R/E is satisfied. Then the first reduction step introduced in Definition 3 arises in a very natural way: if we do not require it, we can have the following cycle for the relation R/E: $t'' \sim t \overset{+}{\to}^{R^E} t' \overset{*}{\to}^{R^E} t_1' \sim t''$, therefore $t'' \overset{+}{\to}^{R/E} t''$ which contradicts the E-termination property of R that is required anyway in the following. The reason why we gave the definitions 3 in this way is that we want to emphasize that the first R^E step from t'' required in the coherence definition plays a crucial role: if a term t is in R^E-normal form, then any term t' E-equal to t must also be in R^E-normal form, otherwise a R^E-step would apply to t by coherence. For the same reason, t is also in R/E-normal form, otherwise $t \sim t' \to^{R} t''$ for some t' and t'' and the same applies. Therefore the two sets of normal forms with respect to R^E and R/E must be the same. This remark will be used freely in the rest of the paper.

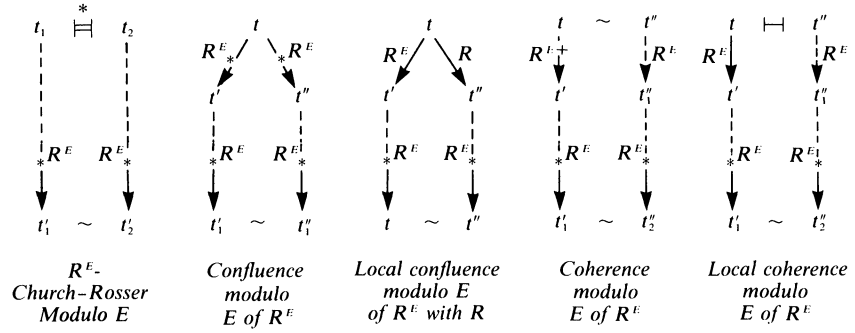


FIG. 2.1. *Main definitions. (Full arrows denote universal hypotheses while dashed arrows denote existential conclusions.)*

Before developing our results, let us give some other important consequences of our definitions:

- Since $\to^{R} \subseteq \to^{R^E} \subseteq \to^{R/E}$, the relations $\sim \circ \to^{R^E} \circ \sim$, $\sim \circ \to^{R/E} \circ \sim$ and $\to^{R/E}$ are the same.
- For $R^E = R/E$, coherence and local coherence are trivially satisfied.
- Different R^E-normal forms of a term are E-equal, by the E-confluence property. We therefore will refer to *the* R^E-normal form of a term.

• Church-Rosser modulo E and R^E-Church-Rosser modulo E are two different properties if R^E and R/E are different. R^E-Church-Rosser is actually the stronger one as shown by the following case: let $t \to^{R^E} t' \sim t'_1 \to^{R^E} t_1$. As a matter of fact, t' and t_1 can be in R^E-normal form, though not E-equal. The R^E-Church-Rosser property is actually *monotonic* with respect to inclusion of relations: this is a straightforward consequence of its definition. Therefore, R-Church-Rosser implies R^E-Church-Rosser implies R/E-Church-Rosser, this last property being precisely the *intrinsic* Church-Rosser property first defined. Choosing an adequate R^E will thus be a relevant practical problem.

Some additional comments about the choice of these definitions can be made:

• Local confluence involves both R and R^E. Using only R would give a relation that is too weak, whereas using only R^E gives one that is too strong for application to ETRS.

• Coherence and local coherence differ from the corresponding definitions in [32] by replacing R by R^E, which is a key point for applications, and by requiring a first R^E step from t'' in the coherence definition, which has been justified above. It differs from the *compatibility* property in [73] by allowing t' to be reduced into some term t'_1. This will permit a more powerful result stating an equivalence between Church-Rosser and both confluence and coherence. Local coherence differs also from the *commutation* property in [42] by allowing t' and t''_1 to be reduced to some terms t'_1 and t''_2 respectively. Handling coherence, however, requires a more powerful termination property than handling commutation. Note finally, that property P1 in [78] is a common ancestor to all of these definitions.

• It is even possible to give a slightly different form to our definitions (without changing at all either the results or the proofs) by using the relation R/E itself to compute from t' and t'' in each of our definitions, except for coherence and local coherence where a first R^E step from t'' to t''_1 is required for having a nonvacuous definition. This was actually done in [40], despite some tedious proofs stemming from a proof technique that was not as powerful as the one used here. The confluence step from the pairs (t', t'') or (t', t''_1) of the definitions can even be performed with an arbitrary reduction provided it contains R^E and is contained in R/E. This last remark is very important in practice, as we will see in §§ 3 and 4.

2.3. Role of coherence. We now show that coherence of R^E modulo E is the property that enables one to compute in E-congruence classes with the relation R^E. This was already the case with the definitions of [40], which are actually equivalent to ours, provided R is E-terminating.

PROPOSITION 4. *Assume R is E-terminating and R^E is confluent modulo E. Then $t\downarrow_{R/E} \sim t\downarrow_{R^E}$ iff R^E is coherent modulo E.*

Proof. Let us first prove the only if part. Assume $t'' \sim t \xrightarrow{+}{}^{R^E} t'$ and let $t'\downarrow_{R^E}$ be any R^E-normal form of t', which is also an R/E-normal form of t', as already pointed out, thus also of t''. Using the hypothesis, it follows that $t''\downarrow_{R^E} \sim t'\downarrow_{R^E}$. As R is E-terminating, $t''\downarrow_{R^E}$ is different from t'', otherwise $t'' \sim t \xrightarrow{+}{}^{R^E} t'\downarrow_{R^E} \sim t''$ and there would be a cycle for R/E. Therefore, $t'' \xrightarrow{+}{}^{R^E} t''\downarrow_{R^E}$ and we are done.

Let us now prove the if part by noetherian induction on R/E. Let t be any element of $\mathcal{S}$. First, the result is true if t is in R/E normal form, thus in R^E-normal form. Otherwise $t \sim t_1 \to^R t_2 \xrightarrow{*}{}^{R/E} t\downarrow_{R/E}$ and $t \xrightarrow{+}{}^{R^E} t\downarrow_{R^E}$.

By the coherence property applied to $(t, t\downarrow_{R^E}, t_1)$, there exist t'_1 and t''_1 such that $t_1 \to^{R^E} t'_1$ and $t'_1 \xrightarrow{*}{}^{R^E} t''_1 \sim t\downarrow_{R^E}$. Note that t''_1 must be in R^E-normal form since it is E-equal to a normal form.

By the confluence property applied to (t_1, t_1'', t_2), there exists t_2'' such that $t_2 \overset{*}{\to}_{R^E} t_2'' \sim t_1''$ and t_2'' must be in R^E-normal form, thus it is a R^E-normal form of t_2. We can now finish the proof by noetherian induction applied to t_2 which is a proper son of t for R/E. □

Note that we could assume R locally R^E-confluent with R modulo E instead of R^E-confluent modulo E: a simple extra induction step needs to be added to the proof for that.

This result states that we can use R^E-computations instead of R/E-computations, if we are interested in normal forms of terms and not in the intermediate steps of computation. This is usually the case in practice. Therefore coherence is exactly the property needed. As a consequence, confluence and coherence modulo E are both needed to compute with R^E, that is to decide $R \cup E$-equality by computing R^E-normal forms and checking for their E-equality.

2.4. The abstract Church–Rosser theorem. We show now that confluence and coherence modulo E can be restricted together to their local versions. This generalizes the theorem of [69] which corresponds to the case where E is the empty set, since both coherence and local coherence become vacuous in that case. For that, we also use normal form computations, since normal forms are the link between R^E-computations and R/E-computations.

THEOREM 5. *If R is E-terminating, the following properties are equivalent*:

(1) *R is R^E-Church-Rosser modulo E.*

(2) *R^E is confluent modulo E and R^E is coherent modulo E.*

(3) *R^E is locally confluent with R modulo E and locally coherent modulo E.*

(4) $\forall t, t' \quad t \overset{*}{\models\!\!=} t'$ *iff* $t\downarrow_{R^E} \sim t'\downarrow_{R^E}$.

Proof. Note that property (4) asserts that the R^E-Church-Rosser property modulo E is true when computing R^E-normal forms of t and t', provided R is E-terminating. This is perfectly similar to the usual case of an empty set E of equations: in that case the Church-Rosser property can be checked on normal forms, provided R is terminating.

(2) $\Rightarrow$ (3) and does not use the E-termination hypothesis.

(4) $\Rightarrow$ (1) is straightforward, but (4) assumes E-termination.

(1) $\Rightarrow$ (2) the confluence proof is straightforward, but the coherence proof requires the use of E-termination to exhibit the first R^E-step from t'' in the definition of coherence.

Let $t \overset{+}{\to}_{R^E} t'$ and $t \sim t''$. Then $t' \overset{*}{\models\!\!=} t''$ and by R^E-Church-Rosser property, $\exists t_1', t_2' \quad t' \overset{*}{\to}_{R^E} t_1', \quad t'' \overset{*}{\to}_{R^E} t_2'$ and $t_1' \sim t_2'$. Assume now that $t'' = t_2'$. Then $t_1' \sim t_2' = t'' \sim t \overset{+}{\to}_{R^E} t' \overset{*}{\to}_{R^E} t_1'$ and thus $t_1' \overset{+}{\to}_{R/E} t_1'$ which contradicts E-termination.

(3) $\Rightarrow$ (4): the right part of the equivalence clearly implies the left. Let us prove the converse by multiset induction on R/E: let $M = \{t_0, t_1, \cdots, t_{n+1}\}$ be a multiset of at least two elements (the one element case is straightforward) such that $t = t_0 \models\!\!= t_1 \cdots \models\!\!= t_{n+1} = t'$. The basic (one step) case being obvious, we consider the general one. Three cases are to be distinguished according to the last equality step $t_n \models\!\!= t_{n+1}$.

- $t_{n+1} \to_R t_n$: since $t_n\downarrow$ is a R^E-normal form of t_{n+1}, the result follows from the induction hypothesis applied to the multiset M-$\{t_{n+1}\}$.
- $t_n \vdash\!\!\dashv t_{n+1}$: the result follows from either the induction hypothesis applied to the multiset M-$\{t_{n+1}\}$ if t_n, and thus t_{n+1}, is R^E-irreducible, or else it follows from the local coherence of R^E and from the induction hypothesis applied to the multiset $\{t_0, \cdots, t_n, t_n'', \cdots, t_n', \cdots, t_{n+1}'\}$ which is strictly smaller than M, since

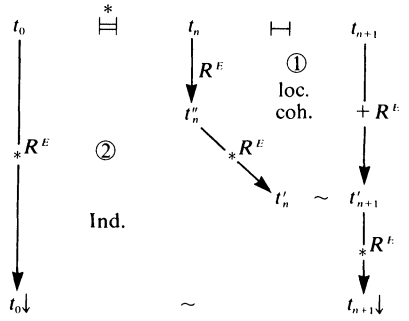


FIG. 2.2. *Church-Rosser proof using multiset induction (case 2).*

the added terms are all proper descendents of t_{n+1} for R/E. This second case is shown in Fig. 2.2, where circled numbers stand for successive steps in the proof.

- $t_n \to^R t_{n+1}$: we first apply the induction hypothesis to the multiset M-$\{t_{n+1}\}$. As t_n is reducible by R (therefore by R^E), we can apply local confluence modulo E of R^E with R. We end the proof by applying the induction hypothesis to the multiset $\{t_n\downarrow, \cdots t''_n, \cdots, t'_n, \cdots, t'_{n+1}, \cdots, t_{n+1}\downarrow\}$ as shown in Fig. 2.3. □

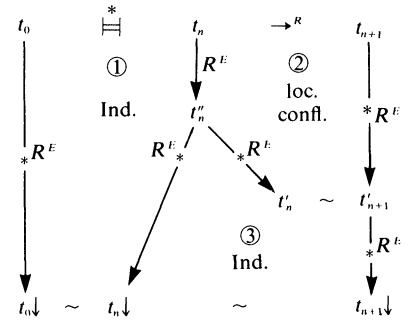


FIG. 2.3. *Church-Rosser proof using multiset induction (case 3).*

Theorem 5 is false if termination of R^E is assumed in place of E-termination of R, as proved by the counterexamples of Fig. 2.4 and Fig. 2.5, where local properties

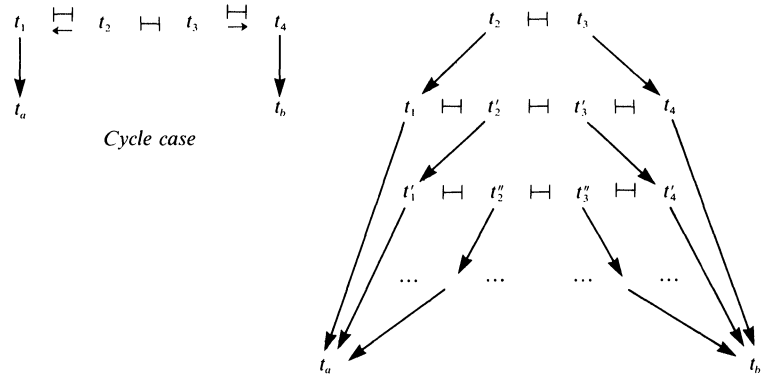


FIG. 2.4. *Crucial role of E-termination: no coherence.*

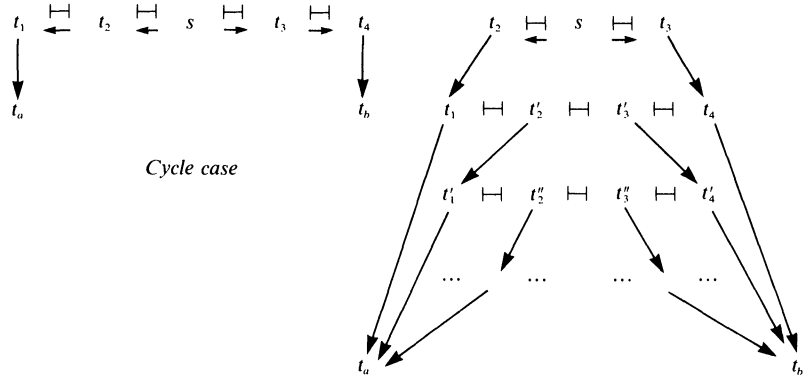


FIG. 2.5. *Crucial role of E-termination: no confluence.*

are true but global ones are not. Note that these examples are similar to [32], with added complexity to ensure that they are not coherent.

The counterexamples of Fig. 2.4 are based on contradicting the coherence property, whereas confluence is trivially satisfied. In the two first examples, $\rightarrow$ is R^E and is clearly terminating, whereas R/E has a cycle. Similarly, we get counterexamples where local confluence is true but confluence is not satisfied. In the examples of Fig. 2.5, $\rightarrow$ stands for $R^E = R$.

As indicated in [78], abstract Church-Rosser results like these apply in a wide range of areas in computer science as well as in mathematics. An interesting example dealing with cyclic covering matrices is given in [78]. A more complex example in formal language theory is given in [66]. In the following we are interested in those applications which deal with terms and term rewriting systems for which we give decision procedures for the Church-Rosser property.

3. Application to equational term rewriting systems. This section is devoted to the study of Church-Rosser results for ETRS, i.e. mixed sets of axioms E and R on a term algebra, such that axioms in E are used as equations and define an equational theory, whereas axioms in R are used as rules and define a *rewriting relation.* However, rewriting with rules in presence of equations is somewhat more complicated than rewriting with rules only: we will introduce other rewriting relations that will play the role of the reduction relation R^E of § 2. In practice, E contains those axioms that cannot be directed without losing the termination property for the rewriting relation. It can also contain other axioms causing the divergence of the completion process when used as rules. The notation used in this section is consistent with that of § 2 and will not be reintroduced. This section is divided into two different parts: the first one recalls the more or less classical background on TRS (see [35] for a more complete presentation); the second states and proves our Church-Rosser result.

3.1. Terms.

DEFINITION 6. Given a set $\mathcal{X}$ of variables and a graded set $\mathcal{F}$ of function symbols, $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denotes the *free algebra* over $\mathcal{X}$ also called the *term algebra.* Variables have arity 0 by convention. Elements of $\mathcal{T}(\mathcal{F}, \mathcal{X})$, called *terms*, are viewed as *labelled trees* in the following way: a term t is a partial application of the free monoid on natural numbers $\mathcal{N}_+^*$ into $\mathcal{F} \cup \mathcal{X}$ such that its domain $\mathcal{D}(t)$ satisfies: the empty word $\varepsilon \in \mathcal{D}(t)$, and $vi \in \mathcal{D}(t)$ iff $v \in \mathcal{D}(t)$ and $i \in [1, \text{arity}(t(v))]$. $\mathcal{D}(t)$ is the set of *occurrences* of t, $\mathcal{G}(t)$ the subset of *ground occurrences* (i.e. $t(v)$ is not a variable for $v \in \mathcal{G}(t)$), $\mathcal{V}(t)$ the

set of variables of t. The *subterm* of t at occurrence v, denoted t/v, is said to be strict if $v \neq \varepsilon$. The relation $>^{sst}$ is defined by $t >^{sst} t'$ iff t' is a strict-subterm of t; $t[v \leftarrow t']$ is the term obtained by replacing t/v by t' in t. A term t is *linear* iff for any x in $\mathcal{V}(t)$, x has only one occurrence in t.

In the following, i, j, k denote natural numbers, x, y, z denote variable symbols, a, b, c and e denote constant symbols, f and h denote function symbols, s and t denote terms, v, ν, w, o and π denote occurrences, ε denotes the outermost occurrence, and $v\nu$ denotes the concatenation of occurrences v and ν.

Example. Let $t = (x + e) + x$. Then $\mathcal{D}(t) = \{\varepsilon, 1, 11, 12, 2\}$, $t(\varepsilon) = +$ and $t(12) = e$, $\mathcal{G}(t) = \{\varepsilon, 1, 12\}$ and t is not linear.

A term with top function symbol f will often be written $f(\cdots t \cdots)$ indicating that we are interested in its first level subterm t. By convention, t and t' occur at the same place in $f(\cdots t \cdots)$ and $f(\cdots t' \cdots)$.

DEFINITION 7. A relation $\rightarrow$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is said to be *compatible* with the term structure iff $\forall t, t' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and $\forall f \in \mathcal{F}$, $t \rightarrow t' \Rightarrow f(\cdots t \cdots) \rightarrow f(\cdots t' \cdots)$. An equivalence relation $\sim$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is called a *congruence* iff it is compatible with the term structure.

Compatibility can easily be extended by induction to the case where t and t' are plugged into the same arbitrary context (a term having a free place to be filled). This remark will be used implicitly in the rest of the paper.

3.2. Substitutions, axioms, unifiers.

DEFINITION 8. *Substitutions* σ are defined to be endomorphisms of $\mathcal{T}(\mathcal{F}, \mathcal{X})$ with a finite domain $\mathcal{D}(\sigma) = \{x \mid \sigma(x) \neq x\}$. We denote by Σ the set of substitutions on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and by σt or $\sigma(t)$ the result of applying the substitution σ to the term t. Composition of substitutions σ and τ is simply written $\sigma\tau$. If $t' = \sigma t$ (resp. $\tau' = \sigma\tau$), we say that t (resp. τ) is *more general* than t' (resp. τ'), that t' (resp. τ') is an *instance* of t (resp. τ), and that σ is the *match* from t to t' (resp. τ to τ'). The *subsumption* preordering $\geqq^{sub}$ on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ (resp. Σ) is defined by: $t \geqq^{sub} t'$ (resp. $\tau \geqq^{sub} \tau'$) iff t is an instance of t' (resp. τ of τ'). We write $\tau \geqq^{sub} \tau'[\mathcal{V}]$ if it holds for the restrictions of τ and τ' to a subset $\mathcal{V}$ or $\mathcal{X}$. The equivalence associated with this preordering is *variable renaming*, i.e. bijection between two sets of variables.

Remember that substitutions are homomorphisms: we will make intensive (and implicit) use of homomorphic properties of substitutions that are often given as lemmas in the literature. We will also use the fact that the strict ordering associated with the subsumption preordering is well-founded [76].

In the following, greek letters σ, θ and τ denote substitutions and $(x \backslash t, y \backslash t', \cdots)$ is the substitution σ such that $\mathcal{D}(\sigma) = \{x, y, \cdots\}$ and $\sigma x = t$, $\sigma y = t', \cdots$.

DEFINITION 9. We call an *axiom* or *equation* any pair (t, t') of terms and write it $t = t'$. It is said to be *linear* if both t and t' are linear. Given a set E of equations, the *one step E-equality* is defined by: $t \vdash\!\dashv^{E}_{[v,\sigma,1=r]} t'$ with $v \in D(t)$, $\sigma \in \Sigma$ and $l = r \in E$ iff $t/v = \sigma l$ and $t' = t[v \leftarrow \sigma r]$ or $t/v = \sigma r$ and $t' = t[v \leftarrow \sigma l]$. We write $l \rightarrow r$ (resp. $r \rightarrow l$) if we want to make precise the fact that the equation $l = r$ is used from left to right (resp. right to left).

The reflexive-transitive closure of $\vdash\!\dashv^{E}$, denoted $\sim^{E}$, or $\sim$, is called *E-equality* or the *equational theory E*.

Any of the E-subscripts may be omitted for the sake of simplicity without any ambiguity, since different notations range over disjoint sets of characters. Note that $\vdash\!\dashv$ is actually the smallest relation on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ which is symmetric, compatible, closed under instantiation and contains all pairs (l, r) for every $l = r \in E$. As a consequence,

$\sim$ is the smallest congruence relation on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ closed under instantiation and containing all pairs (l, r).

All our definitions concerning matching extend straightforwardly to the case of an equational theory $\sim$. For instance, an *E-match* from t to t' is a substitution σ such that $t' \sim \sigma(t)$. Note that the *E-subsumption* preorderings defined on $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and Σ as previously are not always well founded if E is not empty [21]. Their associated equivalence is the composition of $\sim$ with variable renaming. This will be used in § 4.

Example. $t' = e + (y + z)$ is an instance of $t = x + x'$ with the match $\sigma = (x \backslash e, x' \backslash y + z)$. If E contains a commutativity axiom for $+$, then $\sigma' = (x \backslash z + y, x' \backslash e)$ is an E-match from t to t'. These two matches do not compare using $\geqq^{\text{sub}}$. Now $\sigma'' = (x \backslash y + z, x' \backslash e)$ is another E-match and $\sigma' \sim \sigma''$. Note that $\mathcal{D}(\sigma) = \mathcal{D}(\sigma') = \mathcal{D}(\sigma'') = \{x, x'\}$.

Our next definitions handle the general case of a nonempty set E of equations and are specialized for the standard empty case.

DEFINITION 10. An *E-unifier* (or simply *unifier*) of two terms t and t' is a substitution σ such that $\sigma(t) \sim \sigma(t')$. If E is empty there exists a *most general unifier*, called an mgu, for any pair of terms that has unifiers: all other unifiers of these two terms are actually instances of the mgu. If E is not empty, a basis for generating the set of unifiers by instantiation, whenever one exists, is called a *complete set of unifiers*. A *complete unification algorithm* computes such a basis for any two given terms. The algorithm is said to be *finite* if the basis is finite and *minimal* if any two unifiers in the basis do not compare under the E-subsumption preordering.

Standard unification is known to be solvable in linear time [71], but nonlinear algorithms are usually more efficient for many practical applications [7], [20]. Unification in equational theories is a much more difficult problem [51] and the complexity is not precisely known for most of them. However, complete and finite unification algorithms are known for many practical theories including commutativity [76], associativity and commutativity [82], [64], [19], permutativity [39] and minus theory [48]. Some of these are not minimal. However, given such a complete set of unifiers that is finite, a minimal one with respect to the subsumption preordering can always be computed by eliminating those that are instances of others. Minimal sets are usually preferred for reasons of efficiency. Most of our results however carry over to complete sets of unifiers that are infinite, except for decidability results. This enables us to apply some of our results to the associative case, for which a complete but nonfinite algorithm is available [76].

Finally, remark that the E-subsumption ordering is well-founded for all theories of practical interest that have a complete, finite unification algorithm. For the theory given in [21], the E-subsumption preordering is not well-founded, but the unification problem in this theory is nonfinite.

We write

$$\text{SU}(t, t'), \text{mgu}(t, t'), \text{SU}(t, t', E), \text{CSU}(t, t', E)$$

for (respectively) the set of unifiers of t and t', their most general unifier, the set of E-unifiers of t and t' and a complete set of E-unifiers of t and t'.

Example. Let $t = (x + y) + z$ and $t' = (e + x')$. Then $\text{mgu}(t, t') = \varnothing$, but $\text{CSU}(t, t', \{x + y = y + x\}) = \{(x' \backslash x + y, z \backslash e)\}$.

3.3. Rewrite rules. Many theoretical problems arise in equational theories which can be approached by the use of *rewrite rules* (i.e. directed equations) or more generally by the use of *mixed sets of rules R and equations E.*

DEFINITION 11. A *rewrite rule* is a pair of terms, denoted $l \to r$, such that $\mathcal{V}(r)$ is a subset of $\mathcal{V}(l)$. The *rewriting relation* $\to^{R,E}$ (or R,E) is defined as follows: $t \to^{R,E} t'$ iff there exist an occurrence v of $\mathcal{G}(t)$, a rule $l \to r$ of the set R of rewrite rules and a substitution σ such that $t/v \sim \sigma l$ and $t' = t[v \leftarrow \sigma r]$. t/v is called the *redex*. We write $t \to^{R,E}_{[v,\sigma,l\to r]}$ if we want to make precise the occurrence, the substitution and the rule involved in the rewriting. Any of these subscripts may be omitted for simplicity. The reflexive-transitive (resp. transitive) closure of $\to^{R,E}$ is denoted $\overset{*}{\to}^{R,E}$ (resp. $\overset{+}{\to}^{R,E}$) and is called the *derivation relation.*

Example. Let $(e+x') \to x'$ be a rule and $x+y = y+x$ be an equation. Then $x+(y+e)$ is in R-normal form, but $x+(y+e) \to^{R,E}_{[2]} x+y$.

As a special case of R,E, the relation $\to^{R}$ (or R) is the smallest relation which is compatible with the term structure, closed under instantiation and that contains all pairs (l, r) for every $l \to r \in R$. $\overset{+}{\to}^{R}$ and $\overset{*}{\to}^{R}$ are in addition transitive, and $\overset{*}{\to}^{R}$ is also reflexive. But they are generally not symmetric. In the following, we will be interested in those that are well founded.

Note that R,E-reducibility is *decidable* whenever the matching problem is decidable in the theory E. This is always the case if the theory E is empty. When the theory E is not empty, we will either use the relation R or the relation R,E. We will speak about *rewritings* for R and *E-rewritings* for R,E.

It must be understood that the two relations $\to^{R,E}$ and $\sim \circ \to^{R}$ are different: if $t \to^{R,E} t'$ at occurrence v, the E-equality steps may apply in t only at occurrences below v. If $t \sim \circ \to^{R} t'$, then the E-equality steps may apply at any place in t. The two relations are therefore the same only if R,E applies at the outermost occurrence.

Example. Let $+$ be a binary infix operator with the following properties: *associativity* used as an equation: $(x+y)+z = x+(y+z)$, *left-identity* used as a rule: $e+x \to x$. Let us now consider the term $t = (y+e)+z$. By associativity, $t \sim y+(e+z)$ and this last term rewrites to $y+z$. However, t is in normal form for R,E because its subterm $(y+e)$ is obviously in normal form and the whole term t itself cannot be matched with the left member of the rule using associativity, since neither $(y+e)+z$ nor $y+(e+z)$ are instances of $e+x$.

3.4. Critical pairs. Two reductions applied to a same term can sometimes overlap, yielding critical pairs:

DEFINITION 12. A nonvariable term t' and a term t *E-overlap* at occurrence v in $\mathcal{G}(t)$ with a complete set Θ of *E-overlappings* iff Θ is a CSU $(t/v, t', E)$. The usual *overlapping* associated with mgu $(t/v, t')$ is obtained for E empty. Given two rules $g \to d$ and $l \to r$ such that $\mathcal{V}(g) \cap \mathcal{V}(l) = \varnothing$ and l and g E-overlap at occurrence v of $\mathcal{G}(g)$ with the complete set of E-overlappings Θ, then the set $\{\langle\!\langle p, q \rangle\!\rangle \mid p = \theta(g[v \leftarrow r])$ and $q = \theta d$ for any θ in $\Theta\}$ is called a *complete set of E-critical pairs* of the rule $l \to r$ on the rule $g \to d$ at occurrence v (a trivial one if $v = \varepsilon$, $l = g$ and $r = d$). The set of E-critical pairs is simply a *critical pair* if E is empty. With each (E-)overlapping θ is associated the *(E)-overlapped term θg* which produces the (E-)critical pair $\langle\!\langle p, q \rangle\!\rangle = \langle\!\langle \theta(g[v \leftarrow r]), \theta d \rangle\!\rangle$ by rewriting with the rules $l \to r$ and $g \to d$.[1]

[1] Let us point out that a term t may be considered as a variable overlapping of any variable x with no occurrence in t and itself at any occurrence $v \in \mathcal{G}(t)$, $v \neq \varepsilon$, with the mgu $\sigma = (x \setminus t/v)$. Such overlappings produce variable critical pairs of an equation $g = x$ on the rule $l \to r$ by overlapping the variable x and any nonvariable strict-subterm of l. These critical pairs are needed to handle linear rules together with equations of the form $g = x$. We want to point out that they allow a generalization of results in [32] and [40] to the case where such equations are allowed. However, they will not be considered in this framework where the main interest is in decidability results that do not hold if axioms like $g = x$ can be used as equations.

Note that $l \to r$ and $g \to d$ do not play symmetric roles in this definition. Let CP(R, R), CP(E, R) and CP(R, E) be the sets of nontrivial critical pairs for, respectively: all $l \to r$ on $g \to d$ belonging both to R, all $l \to r$ and $r \to l$ for $l = r$ in E on all $g \to d$ in R, all $l \to r$ in R on all $g \to d$ and $d \to g$ for $g = d$ in E. Note that *top critical pairs* (i.e. critical pairs coming from an overlapping at the outermost occurrence ε) are not to be considered for CP(R, E) since they already belong to CP(E, R). Let also ECP(R, R) and ECP(R, E) be the sets of nontrivial E-critical pairs for respectively all $l \to r$ in R on all $g \to d$ in R, and all $l \to r$ in R on all $g \to d$ and $d \to g$ for $g = d$ in E. As previously, but for a slightly different reason that will become clear later, top critical pairs are not to be considered for ECP(R, E). The notation CP$(l \to r, g \to d)$ and ECP$(l \to r, g \to d)$ will also be used to make the rules or equations precise.

Example. Let $(x+y)+z \to x+(y+z)$ and $(e+x') \to x'$ be rules and $(y'+e) = y'$ be an equation. Then:

1) $\langle\langle p, q\rangle\rangle = \langle\langle y+z, e+(y+z)\rangle\rangle$ is a critical pair of the second rule on the first at occurrence 1, associated with the overlapped term $(e+y)+z$. Note that $q \to^R p$.

2) $\langle\langle p, q\rangle\rangle = \langle\langle z, e+(e+z)\rangle\rangle$ is an E-critical pair of the second rule on the first at occurrence ε, associated with the E-overlapped term $(e+e)+z \sim e+z$. Note that $q \overset{+}{\to}^R p$.

3) $\langle\langle p, q\rangle\rangle = \langle\langle x+y, x+(y+e)\rangle\rangle$ is a critical pair of the equation on the first rule at occurrence ε, associated with the overlapped term $(x+y)+e$. Note that $q \sim p$.

In the following, we use R-reductions as well as R,E-reductions or more generally combinations of both of them. As a consequence, we will have critical pairs as well as sets of E-critical pairs, according to the kind of reduction associated with each rule. R-reductions yield critical pairs as indicated by the critical pairs lemma:

CRITICAL PAIRS LEMMA [32]. *Assume* $t \to^R_{[\varepsilon,\sigma,g\to d]} t''$ *(or* $t \vdash\!\dashv^E_{[\varepsilon,\sigma,g\to d]} t''$*) and* $t \to^R_{[\nu,\sigma,l\to r]} t'$ *(or* $t \vdash\!\dashv^E_{[\nu,\sigma,l\to r]} t'$*) with* ν *in* $\mathcal{G}(g)$*. Then there exists a critical pair* $\langle\langle p, q\rangle\rangle = \langle\langle \theta(g[\nu \leftarrow r]), \theta d\rangle\rangle$ *of the rule (or equation)* $l \to r$ *on the rule (or equation)* $g \to d$ *at occurrence* ν *and a substitution* τ *such that* $\theta = \mathrm{mgu}(g/\nu, l)$ *and* $\sigma = \tau\theta\,[\mathcal{V}(g) \cup \mathcal{V}(l)]$*; therefore* $t' = \tau p$ *and* $t'' = \tau q$.

Note that we use the same substitution σ for both redexes t and t/ν. This is legal, since we can always assume that $\mathcal{V}(l) \cap \mathcal{V}(g) = \varnothing$ without lost of generality: just rename the variables when needed.[2]

R,E-reductions yield sets of E-critical pairs:

E-CRITICAL PAIRS LEMMA [40]. *Assume* $t \to^R_{[\varepsilon,\sigma,g\to d]} t''$ *(or* $t \vdash\!\dashv^E_{[\varepsilon,\sigma,g\to d]} t''$*) and* $t \to^{R,E}_{[\nu,\sigma,l\to r]} t'$ *with* ν *in* $\mathcal{G}(g)$ *and* $\nu \neq \varepsilon$ *if* $g \to d$ *belongs to* E*. Then there exist a critical pair* $\langle\langle p, q\rangle\rangle = \langle\langle \theta(g[\nu \leftarrow r]), \theta d\rangle\rangle$ *in a set of E-critical pairs of the rule* $l \to r$ *on the rule (or equation)* $g \to d$ *at occurrence* ν *and a substitution* τ *such that* $\theta \in \mathrm{CSU}(g/\nu, l, E)$ *and* $\sigma \sim \tau\theta[\mathcal{V}(g) \cup \mathcal{V}(l)]$*; therefore* $t' \sim \tau p$ *and* $t'' \sim \tau q$.

Note that we overlap R,E-reductions with R-reductions or with a one step E-equality. No other case has to be considered, as shown by a careful examination of our two local definitions of coherence and confluence given in § 2. In addition, we do not consider the case where R applies at an occurrence ν and R, E at the outermost occurrence ε: the E-critical pairs lemma would be false for such cases! We will see in our main theorem how to handle this case without using the lemma.

The E-critical pairs lemma is proved in a way similar to Huet's [32], but we need to make precise where E-equality steps can take place from t'' to τq: since $t'' = \sigma d$, $\tau p = \tau\theta d$ and $\sigma \sim \tau\theta$, they actually take place in the substitution part of t'' and not in

[2] Although it was originally stated for standard critical pairs, the lemma remains true for the case of variable ones with the same proof. The verification is left to the reader.

$\mathcal{G}(d)$. This remark is essential for carrying out the proofs in [73] and [40]. Here we will only need to assume that it does not take place at the outermost occurrence of t'', thanks to multiset induction.

3.5. Decidability of the Church–Rosser property for ETRS. Our next goal is to introduce a more general reduction relation R^E and to restrict local confluence modulo E of R^E with R and local coherence modulo E of R^E to a convergence check on a *finite* number of critical pairs or E-critical pairs. For this purpose, the relation R/E is not convenient, because there is no notion of critical pairs which corresponds to an overlapping between R and R/E-reductions. We therefore implicitly assume in what follows that R^E is not R/E. The two cases where $R^E = R$ and $R^E = R,E$ have already been studied in [40]. The first one requires R to be left-linear and yields Huet's results on *confluence modulo*, while the second requires the computation of E-critical pairs using a complete E-unification algorithm and yields a generalized version of Peterson and Stickel's results.

A first subgoal is to generalize both these cases by splitting the set R of rules into disjoint sets L and N with only left-linear rules in L and use for R^E the relation $L \cup N,E$ defined to be $\rightarrow^L \cup \rightarrow^{N,E}$. The relation $L \cup N,E$ was first introduced and studied in [40], but with a less powerful proof technique, yielding more restrictive results. Note that left-linear rules may be in N, which allows us to consider R and R,E as two particular cases of $L \cup N,E$. Thus letting $L = \emptyset$ yields Peterson's and Stickel's relation R,E and letting $N = \emptyset$, provided all rules are left-linear, yields the standard rewriting relation R.

In the following, R^E is used instead of the full notation $L \cup N,E$ for simplicity.

A second subgoal is to prove the decidability of the R^E-Church-Rosser property modulo E, provided R is E-terminating. This is done by proving that the R^E-Church-Rosser modulo E property is equivalent to checking that the normal forms of finitely many critical pairs are E-equal. Such a result was already known for the case where N is empty and rules in R are all left-linear. We will prove it for R^E, provided the theory E has a complete and finite unification algorithm, and only finite congruence classes if N is not empty.

Since R^E is included in R,E, the R^E-Church-Rosser modulo E property implies the R,E-Church-Rosser modulo E property. As already noted before, we must be aware of the fact that R can be R,E-Church-Rosser modulo E and not R^E-Church-Rosser modulo E for a nonempty set L of left-linear rules. Examples will be given at the end of the section.

The basic tool of the proof is multiset induction on the reduction relation $R/E \cup >^{sst/E}$, denoted $\mapsto$ in the following. The termination of $\mapsto$ is simply based first on the fact that R/E and $>^{sst}$ are each noetherian and second on the following commutation lemmas, whose use permits one to transform a $\mapsto$-derivation into a R/E-derivation followed by several steps of taking subterms. The two following lemmas are based on the fact that the relation $>^{sst}$ commutes with any relation compatible with the term structure, for instance R and $\sim$.

LEMMA 13. *$t >^{sst/E} t' \Rightarrow \exists t''$ such that $t \sim t'' >^{sst} t'$.*

Proof. Straightforward induction on the number of steps of E-equality following the strict-subterm step in $>^{sst/E}$. □

LEMMA 14. *$s \overset{*}{>}^{sst/E} t \overset{+}{\rightarrow}^{R/E} u$ implies that there is a term t' such that $s \overset{+}{\rightarrow}^{R/E} t' \overset{*}{>}^{sst} u$.*

Proof. By induction on the number n of $>^{sst}$-steps. The basic $n = 0$ case is straightforward with $t' = u$. Otherwise, let $s \overset{*}{>}^{sst/E} s_1 >^{sst/E} t \overset{+}{\rightarrow}^{R/E} u$. By Lemma 13, there exists $s_1'' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $s_1 \sim s_1'' >^{sst} t \overset{+}{\rightarrow}^{R/E} u$. Since $a >^{sst}$ step can be

performed after the reductions instead of before, we get $s_1 \sim s_1'' \overset{+}{\to}^{R/E} t' >^{sst} u$ for some t', therefore $s_1 \overset{+}{\to}^{R/E} t' >^{sst} u$. The result is then achieved by applying the induction hypothesis to $s \overset{*}{>}^{sst/E} s_1 \overset{+}{\to}^{R/E} t'$. □

As a consequence of these lemmas, we get the following termination result:

PROPOSITION 15. *Assume R is E terminating and E-congruence classes are finite. Then $\mapsto$ is well founded.*

Proof. Assume given an infinite chain of $\mapsto$. Either it contains no step of R/E-reduction and $>^{sst/E}$ would not be well founded, or Lemma 14 applies to the first consecutive steps of R/E. If the infinite chain contains an infinite number of R/E steps, R/E could not be noetherian. Thus we get that actually the chain can be written as the composition $\overset{*}{\to}^{R/E} \circ \overset{*}{>}^{sst} \circ \overset{*}{>}^{sst/E}$. Infinite chains can thus occur iff $>^{sst/E}$ is not well founded. From Lemma 13, this is not the case if E-congruence classes are finite, since the size of the terms in a given congruence class becomes bounded. □

Note that the problem of deciding whether a set of equations generates finite congruence classes only is undecidable in general [77]. Many theories of interest have this property. Some interesting ones do not, however, like equipotency $--x = x$ or even identity. These kind of axioms do not fall within the scope of our decidability results.

Let us now make some comments about the termination of R/E: assume that E contains an equation $g = d$ which is erasing, i.e. there exists a variable x of $\mathcal{V}(g)$ that is not in $\mathcal{V}(d)$. Let $l \to r$ be a rule of R, σ and σ' substitutions such that $\sigma(x) = l$ and $\sigma'(x) = r$. Then $d = \sigma d \sim \sigma g \overset{+}{\to}^{R} \sigma' g \sim \sigma' d = d$ and the E-termination property is not satisfied. Therefore we will assume in the following that equations in E are *nonerasing*, i.e. $\mathcal{V}(g) = \mathcal{V}(d)$ for any equation $g = d$ in E.

Suppose now that E contains an equation $t = x$ where x has at least two occurrences in t. The relation R/E is not well-founded: let $l \to r$ be any rule. Then l is E-equal to a term with several occurrences of l. One of them can be rewritten and the process started again with another occurrence of l. For instance, in the case of the idempotency equation $x = x + x$, for any rule $l \to r$: $l \sim l + l \to^{R} r + l \sim r + (l + l) \to^{R} r + (r + l) \cdots$. The same problem actually arises with any such axiom where x is replaced by any term with variables. As a consequence, the only allowable axioms of the form $g = x$, with respect to E-termination of R, have exactly one occurrence of x in g. Notice however that even such axioms are forbidden if $>^{sst/E}$ has to be well founded (at least if g is not x itself).

Let us now give our main result:

THEOREM 16. *Assume E to be a set of equations such that a complete and finite unification algorithm exists and E-congruence classes are finite. Let $R = L \cup N$ be an E-terminating set of rules such that all rules in L are left-linear. Let R^E be defined as $L \cup N, E$. Then R is R^E-Church-Rosser modulo E iff*:

1) *all confluence pairs $\langle\langle p, q \rangle\rangle$ in $\mathrm{CP}(L, L) \cup \mathrm{CP}(L, N) \cup \mathrm{ECP}(N, N) \cup \mathrm{ECP}(N, L)$ are R/E-confluent modulo E,*

2) *for any coherence pair $\langle\langle p, q \rangle\rangle$ in $\mathrm{CP}(L, E) \cup \mathrm{ECP}(N, E)$, $\exists q'$ such that $q \to^{R^E} q'$ and the pair (p, q') is R/E-confluent modulo E,*

3) *for any coherence pair $\langle\langle p, q \rangle\rangle$ in $\mathrm{CP}(E, L)$, $\exists p'$ such that $p \to^{R^E} p'$ and the pair (p', q) is R/E-confluent modulo E.*

Proof. For the only if part, we use Theorem 5 to show that the local properties are satisfied, in particular for critical pairs. Since these pairs are R^E-confluent modulo E, they are R/E-confluent modulo E by inclusion of R^E into R/E. The if part starts the same as the proof of Theorem 5 (part (3)⇒(4)) and we assume it. (Notice that it requires the use of Theorem 5 because we prove statement (4) instead of statement

(1) of this theorem.) The difference is that we have to prove the two properties of local coherence and local confluence using the sets of critical pairs. This is done by multiset induction on $\mapsto$. Notation are the same as in Theorem 5. In particular, t_n is the one introduced in its proof, $t\downarrow$ is a R^E-normal form of t and $p\downarrow q$ iff (p, q) is R^E-confluent modulo E.

Let us first prove local coherence, specifically;

$\forall t, t', t''$ such that $t_n \sim t$, $t \vdash\!\!\mid_{[v,\sigma,g\to d]} t''$ and $t \to^{R^E}_{[\nu,\sigma\to r]} t'$, $\exists t_1''$ such that $t'' \to^{R^E} t_1''$ and $t_1''\downarrow t'$. As in the critical pairs lemmas, we use the same substitution for both redexes. Note also that we prove a slightly more general local coherence property than the one needed in the proof of Theorem 5, because it will be needed in the rest of the proof.

Let us now discuss different cases according to the respective positions of v and ν:

- *Case* 1. v and ν are disjoint occurrences. Then the $\to^{R^E}$ and $\vdash\!\!\mid$ steps commute.
- *Case* 2. For the remaining cases, one occurrence is above another. The result is now straightforward if this occurrence (assume it is v) is not the empty occurrence ε, since we can apply multiset induction to the multiset $\{t'/v, t/v, t''/v\}$, yielding $t'/v\downarrow \sim t''/v\downarrow$, with $t''/v \neq t''/v\downarrow$, since R is E-terminating. By the way, this is exactly the coherence property and it can be lifted to (t', t, t'') by adding the missing context. We therefore assume in the following that the prefix occurrence is the outermost occurrence ε.
- *Case* 3. ν is a prefix of v (so assume $\nu = \varepsilon$) with $l \to r$ in N. It is given that $t'' \vdash\!\!\mid t$, thus (by the definition of N,E) $t'' \to^{N,E}_{[\varepsilon,l\to r]} t'$. This is the reason why critical pairs of equations on rules of N need not be considered.
- *Case* 4. ν is a prefix of v (so assume $\nu = \varepsilon$) with $l \to r$ in L, and $v \notin \mathcal{G}(l)$. Then the result follows in the same way as in case 6 below.
- *Case* 5. ν is a prefix of v (so assume $\nu = \varepsilon$) with $l \to r$ in L, and $v \in \mathcal{G}(l)$. Then the result follows classically from the Critical Pairs Lemma, using a critical pair of CP(E, L). Note that $g = d$ can be the reflexivity axiom and that no critical pair is needed for this particular case.
- *Case* 6. v is a prefix of ν (so assume $v = \varepsilon$) and $\nu \notin \mathcal{G}(g)$, thus $\nu \neq \varepsilon$. Because no overlapping occurs in this case, there exists an occurrence ω which is a prefix of $\nu = \omega\nu'$ and such that $g(\omega)$ is a variable x. Let us now define a substitution θ such that $\theta(y) = \sigma(y)$ for any y distinct from x and $\theta(x) = \sigma(x)[\nu' \leftarrow \sigma r]$. As equations are nonerasing, x occurs at least once in d. It is then easy to check that $t'' \xrightarrow{+}{}^{R^E} \theta d$ and $t' \xrightarrow{*}{}^{R^E} \theta g$ which ends this case, since $g = d \in E$. Note that our definition for coherence allows rewriting from t', which was not allowed by Peterson and Stickel (this is why equations were required to be linear in their work).
- *Case* 7. v is a strict prefix of ν (so assume $v = \varepsilon$), $\nu \neq \varepsilon$, and $\nu \in \mathcal{G}(g)$ with $l \to r$ in L. Then, the result follows classically from the Critical Pairs Lemma, using a critical pair of CP(L, E). Top critical pairs are useless here since $\nu \neq \varepsilon$.
- *Case* 8. v is a strict prefix of ν (so assume $v = \varepsilon$), $\nu \neq \varepsilon$ and $\nu \in \mathcal{G}(g)$ with $l \to r$ in N. As a consequence, $\mathcal{G}(g) \neq \emptyset$.

  Once more, there will be no need of top critical pairs since $\nu \neq \varepsilon$. By the E-Critical Pairs Lemma, there is a pair $\langle\!\langle p, q\rangle\!\rangle = \langle\!\langle \theta(g[\nu \leftarrow r]), \theta(d)\rangle\!\rangle$ in ECP(N, E) and a substitution τ such that $\sigma \sim \tau\theta[\mathcal{V}(g) \cup \mathcal{V}(l)]$, therefore $t'' \sim \tau q$ with E-equality steps taking place out of $\mathcal{G}(d)$ and $t' \sim \tau p$. By assumption, $q \to^{R^E}_{[\omega]} q'$ and the pair (p, q') is R/E-confluent modulo E, therefore $\tau q \to^{R^E}_{[\omega]} \tau q'$ and the pair $(\tau p, \tau q')$ is R/E-confluent modulo E, which implies $\tau p \overset{*}{\vdash\!\!\mid} \tau q'$ with a proof containing terms that are all smaller than $\tau q'$ or τp for $\mapsto$. This is step

1 on Fig. 3.1. Since the multiset $\{t', \cdots, \tau p, \cdots, \tau q', \tau q, \cdots, t''\}$ contains terms like τq and t'' which are not smaller than t for $\mapsto$, the induction hypothesis cannot be applied to this multiset. To overcome this difficulty, we now show that t'' is R^E-reducible to some t_1'' related to $\tau q'$ by a multiset of $R \cup E$-equal terms all smaller than t. As already noted, we know from the E-Critical Pairs Lemma that the equality steps between τq and t'' take place out of $\mathcal{G}(d)$, thus at some nonoutermost occurrences $o_1 \cdots o_n$ all different from ε. As a consequence, we may apply the (already proved) local coherence property (cases 1 to 6) for the first step of E-equality (on the three terms $\tau q'$, τq, t_1') to get a term t_2' such that $t_1' \to^{R^E} t_2' \downarrow \tau q'$. This is step 2 on Fig. 3.1. The same process applies until we reach t'' and get a term t_1'' such that $t'' \to^{R^E} t_1''$. This is step 3 on Fig. 3.1. The conclusion then follows by applying the induction hypothesis to the multiset $\{t', \cdots, \tau p, \cdots, \tau q', \cdots, t_2', \cdots, t_1''\}$.

Let us now prove local confluence, specifically:

For any t' and t'' such that $t_n \to^{R^E}_{[\nu,\sigma,1\to r]} t'$ and $t_n \to^{R}_{[v,\sigma,g\to d]} t''$, $t' \downarrow t''$. As before, the proof is by cases according to the respective positions of v and ν.

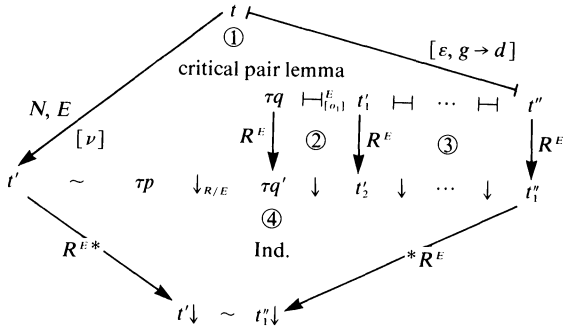


FIG. 3.1. *Local coherence proof, Case* 8.

- *Case* 1. v and ν are disjoint. Then the diagram commutes and we are done.
- *Case* 2. Neither v nor ν is the outermost occurrence ε. This case works as case 2 in the proof of local coherence.
- *Case* 3. $v = \varepsilon$. The subcase where there is no overlapping works as usual. The other subcase is proved by the Critical Pairs Lemma if R^E is L, and by the E-Critical Pairs Lemma if R^E is N,E, followed (in both cases) by multiset induction. This will use, in the first case a critical pair in CP(L, L) or CP(L, N), and in the second case an E-critical pair in ECP(N, N) or ECP(N, L). Note that top critical pairs are needed here.
- *Case* 4. $\nu = \varepsilon$ and $v \neq \varepsilon$. The subcase without overlapping works as usual. The other subcase is proved using the Critical Pairs Lemma if R^E is L. If R^E is N,E, the E-Critical Pairs Lemma does not apply and we need the use of the already proved local coherence property. Since $t_n \to^{R^E} t'$, there exists t such that $t_n \sim t \to^{R}_{[\varepsilon,l\to r]} t'$. On the other hand, $t_n \to^{R}_{[v,g\to d]} t''$. We therefore can apply local coherence n times starting from t_n, getting terms $t_n'', \cdots, t_1''$, until we reach t with the following situation:

$$t'' \downarrow t_n'' \downarrow \cdots \downarrow t_1'', \quad t \to^{R^E} t_1'' \quad \text{and} \quad t \to^{R}_{[\varepsilon,l\to r]} t'.$$

These are steps 1 and 2 on Fig. 3.2. Since the R-reduction applies now at the outermost occurrence of t, we can apply the already proved case 3 to close

the diagram, thus $t_1''\downarrow t'$. This is step 3 on Fig. 3.2. The proof is then completed by using the Church-Rosser property applied to the multiset $\{t'', \cdots, t_n'', \cdots, t_1'', \cdots, t'\}$. This is step 4 on Fig. 3.2.

The reader is encouraged to check that the critical pairs used all along the proof are exactly those defined previously. □

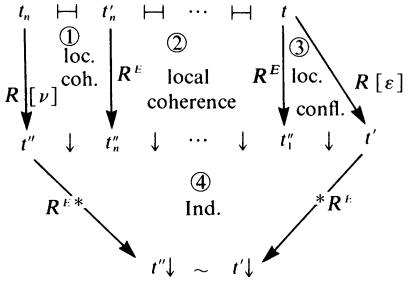


FIG. 3.2. *Local confluence proof, Case* 4.

The proof makes clear why left-linear rules are easier to handle: what happens at a variable place in an instance of rule happens once, which makes the coherence diagrams commute by simple rewriting. This is not the case for nonleft-linear rules: if an equality is applied at a variable place, the first rewriting needed in the coherence property must incorporate the same equality at the other variable places (of the same variable of course), since an equality step is not allowed here.

This Church-Rosser result has been generalized to more complex combinations of rewriting relations in [52]. A weaker result was given in [40], where it was shown that the R^E-Church-Rosser property was true under the stronger assumption that coherence pairs $\langle\langle p, q\rangle\rangle$ of rules in N on equations have the property that q is always reducible to q' at an occurrence in $\mathcal{G}(d)$. As a matter of fact, this result does not require the equational theory E to have finite congruence classes. The reason is that we do not need any induction with $>^{sst/E}$ in this case, since Case 8 in the proof of local coherence becomes vacuous and cases 2 in both local proofs are used only for convenience. On the other hand, the added requirement on these critical pairs precluded proving necessity, since this strong assumption on the way the coherence pairs become convergent cannot be a consequence of the R^E-Church-Rosser property. The question arises now whether our equivalence result is true for those theories that have infinite congruence classes. We conjecture it is. A way to prove it would be to show that we do not need the full power of the relation $\mapsto$ in the proof, i.e. the infinite $>^{sst/E}$-chains cannot occur. For doing that, a useful assumption could be that the critical pairs are of minimal size, which can be achieved by choosing an adequate E-unification algorithm. Note that such an algorithm can always be obtained from a standard complete one by checking each unifier for minimality with respect to its size.

As a corollary of the previous result we can decide the R^E-Church-Rosser property of a set R of rules modulo a set E of equations by checking for E-equality the R''-normal forms of the pairs (p, q) or (p', q) or (p, q') of the previous theorem, for any reduction relation R'' ranging between R^E and R/E.

THEOREM 17. *Let $R = L \cup N$ be an E-terminating set of rules such that L is left-linear, a complete and finite unification algorithm exists for the theory E and E-congruence classes are finite. Then the R^E-Church-Rosser property is decidable.*

Proof. As a matter of fact, the E-equality of the previous confluence and coherence pairs implies the R^E-Church-Rosser property. Conversely, the R^E-Church-Rosser property implies the desired property with R^E-normal forms instead of R''-normal forms. But we know from Theorem 4 that R/E-normal forms and R^E-normal forms are the same, thus R''-normal forms too, since R'' is included in R/E and contains R^E. □

We want to emphasize that we allow normal forms of critical pairs to be computed with any reduction relation R'' ranging between R^E and R/E instead of R^E. This is very important in practice, because it allows E-equality steps to be performed during the reduction process by R^E. This arises in a natural way in many practical cases: for computing associative-commutative (AC) critical pairs, for instance, *flattened terms* can be used to make the unification process easier. Flattening results in applying the rule $f(x, f(y), z) \to f(x, y, z)$, where x, y and z stand for (maybe empty) vectors of variables, for any AC-symbol f, whose arity is now variable. This can result in modifications of the starting terms in a same AC-equivalence class. Our result proves that it does not alter the soundness of the whole process. Of course this is not completely true for coherence pairs, since a first R^E step is needed here: only then can terms be flattened!

Let us now come back to a main practical problem: which relation R^E is the best one for efficiency of computations? Clearly, the more rules there are in N, the more often the E-matching is used, so the more inefficient the rewritings are, whereas the more powerful is the rewriting relation. We therefore want to find the most efficient rewriting relation R^E that provides a R^E-Church-Rosser ETRS.

The previous proof makes clear that the relation R^E is linked to the rules that are used to reduce p to p' or q to q' for the coherence pairs $\langle\!\langle p, q \rangle\!\rangle$. As a consequence, if a left-linear rule of N is never used with the full power of the relation N,E (recall that $\to^N \subset \to^{N,E}$) to reduce such a p or a q, it could perhaps be dropped to L. But this is only a hint: new critical pairs of the equations on this rule have to be computed now and they must be confluent modulo E. This discussion suggests that all rules should be first in N, trying then to construct an L as indicated. Clearly, some work will be needed to find good strategies for that. Let us now consider an example.

Example[3]. Let 0, 1, -1 be constants and $+$ a binary infix operator. Let

$$0+x' \to x',$$

$$1+-1 \to 0,$$

$$(x'+1)+-1 \to x'$$

be the rules of R and

$$(x+y)+z = x+(y+z), \qquad x+y = y+x$$

the equations of E.

It can be verified (using a Knuth-Bendix implementation, for example the FORMEL system [38] or the REVE system [61], [53]) that this set of rules is R,E-Church-Rosser. As these rules are all left-linear, they can all be dropped to L. In that case, the set of rules is not Church-Rosser anymore. Moreover, its completion generates an infinite set of rules, as we will see in the next section. Which rules do we really need to have in N? First of all, we need the first rule in N to make the following coherence E-critical pair convergent: $\langle\!\langle p, q \rangle\!\rangle = \langle\!\langle x+z, (x+0)+z \rangle\!\rangle$ obtained from the superposition

[3] This example is due to Etienne Paul who pointed out the previous problem.

of $0+x'$ on $x+(y+z)$ at occurrence 1 with the unifier $(y\backslash 0, x'\backslash z)$. Then q reduces to p using the rule $0+x' \to x'$ modulo commutativity. Checking the other rules is left to the reader or any Knuth-Bendix implementation.

Note that only commutativity was used in this example: the question arises whether our results can be extended to the case where each rule $l \to r$ has an associated subset E' of E that can be used for E'-rewriting. Clearly, the framework used here is powerful enough to solve this problem and see what are exactly the critical pairs to be computed for such a rewrite relation. In that case, a left-linear rule can be considered to have both nonlinear and linear status according to the respective sets E' and E-E'. Nonlinear status will require the computation of sets of E-critical pairs whereas linear status will require the computation of usual critical pairs. In this case, we need complete sets of unifiers for a subtheory E' of the whole theory E. For the last example, the subtheory is the commutative one. However a theory can have a complete and finite unification algorithm whereas, at the same time, one of its subtheories does not have one. For instance, the AC-theory has one although the associative one does not have a finite one. This remark limits the practical interest of such a method. It can, however, be interesting for speeding up reductions in some cases.

Example. Let us consider an interesting algebraic theory having constants, a unary prefix operator $-$ and a binary infix operator $+$ with the following properties:

$$
\begin{aligned}
&--x = x,\\
&-(x+y) = -y + -x,\\
&-x + (x+y) = y,\\
&(y+x) + -x = y.
\end{aligned}
$$

These axioms can be used as rules. Applying then the standard Knuth-Bendix completion procedure results in divergence for any possible orientation of the generated rules. However, the infinite set of rules can be described with a finite set of *meta-rules* [44], [49]. As an alternative, we can try to use as equations those rules causing divergence. Let us assume that the two first axioms are used as equations and the two others as rules. The equational theory associated with the first two axioms has a finite and complete unification algorithm [48]. Checking the set of rules shows that it has the R,E-Church-Rosser property [49]. This is a good example of an ETRS that does not fall within the scope of our decidability results since the congruence class of x is infinite, whereas it can be worked out using the results of [40], where sufficient conditions are given for testing the Church-Rosser property.

Before ending this section, let us point out that little is known about E-termination. A theoretical study of the problem may be found in [42] and some effective, yet complex, methods in [14] and [74]. On the other hand, some progress has also been made recently for checking the Church-Rosser property using a weaker termination property than E-termination, namely the termination of the reduction relation. Two different approaches can be used; one is reported in [70] and [43], the other in [42]. The properties used in place of coherence in these works are much stronger and involve many restrictions in practice. The second approach, however, is well suited to some equational theories like the associative-commutative one.

4. The *E*-completion procedure. Given a set of axioms $R \cup E$ defining a congruence $\overset{*}{\leftrightarrow}$, the E-completion procedure attempts to obtain a confluent and coherent set of rules that are as *inter-reduced* as possible and generate the same congruence. The E-completion procedure can be used for testing for the Church-Rosser property of a given set of rules, or as a semi-decision procedure for $R \cup E$-equality as in [32].

Completely inter-reduced sets of rules (such that left- and right-hand sides of all rules are irreducible) can be expected to have a uniqueness property, i.e. they depend upon the equational theory $R \cup E$ rather than upon a particular presentation of it. Although this goal is easily achieved for the standard completion algorithm [65], it turns out to be one of the main difficulties in the design of the general E-completion procedure. As a consequence, we will discuss several variants of our algorithm, and show that only one of them guarantees an inter-reduced result. On the other hand, we will see how to get a completely inter-reduced set of rules from a given Church–Rosser set of rules that does not have the property.

4.1. A general completion procedure for ETRS. As a main feature of our algorithm, coherence is *dynamically* ensured, unlike in Peterson and Stickel's AC-completion algorithm, where coherence is ensured by systematically adding the so-called AC-*extended rules* $f(f(x, l_2), l_2) \to f(x, r)$ with $x \notin \mathcal{V}(l_1) \cup \mathcal{V}(l_2)$, to those rules $f(l_1, l_2) \to r$ having an AC-function symbol f at the outermost occurrence of their left-hand side [73].

The E-completion procedure works on a set P of pairs, a set R of rewrite rules, a constant set E of equations and a well founded *E-reduction ordering*, i.e. a compatible quasi-ordering $\geqq$ such that its associated equivalence $\simeq$ contains $\sim$ and its associated strict order $\succ$ is well-founded. See [10] or [15] for an introduction to termination proof methods and [42] for an introduction to E-termination proof methods.

There are two ways of using the E-completion procedure, that differ in their initialization step:

- To build an R^E-Church–Rosser set of rules from a given set of axioms. In that case, it starts with an empty set R of rules and a set P of pairs initialized with those axioms that are not in E.
- To check the R^E-Church–Rosser property of an ETRS made up from a set E of equations and an E-terminating set R of rules. In that case, it starts with an empty set P of pairs and the set R itself, whose rules must be *protected*, otherwise some of them could be removed if they are reducible by some other rules as explained next. If the Church–Rosser property is not satisfied, the set of rules may then be completed as before.

During the completion process, P is updated by picking one pair at a time in order to make a new rule and by dropping in those rules whose left-hand side becomes reducible by a newly introduced rule, together with the newly computed critical pairs. Pairs in P are further compared using the E-reduction ordering $\succ$, producing rules for R. The rewriting system R is divided into a set L of left-linear rules and a set N of nonleft-linear rules. Each rule $l \to r$ is:

- *Labelled* by an integer n and denoted $n : l \to r$, $l_n \to r_n$, or simply n. The label tells us when the rule was created.
- *Marked* as soon as all its critical pairs with the axioms of E and with the previously created rules have been computed.

In addition, two particular features are added in order to ensure the coherence property: *protected rule* and *extensions*. Let $S = \{\langle\langle p, q\rangle\rangle = \langle\langle \theta(g[v \leftarrow r]), \theta d\rangle\rangle \mid \theta \in \mathrm{CSU}(g/v, l, E)\}$ be the set of E-critical pairs of a left-linear or nonleft-linear rule $n : l \to r$ on an equation $g \to d$ of E at occurrence v. Remember that the left member q of any of these E-critical pairs in S has to be R^E-reducible to ensure coherence.

- Assume first that q is R^E-reducible, but only at the outermost occurrence, by a nonleft-linear rule $k : l_k \to r_k$ such that q is an E-instance of l_k and not simply an instance. In that case, q is said to be *top-reducible* by the nonleft-linear rule k and k *protected* for coherence of n. Without protection, such a coherence critical pair $\langle\langle p, q\rangle\rangle$

could satisfy the required property at some step of the completion and not at a later step if the left-hand side of the previous nonleft-linear rule k has become reducible and if the rule has been dropped into P. As long as a rule is protected, its left-hand side is not checked for reduction and the rule cannot be removed from the current rewriting system. Note that a rule can be protected for coherence of several rules.

• Assume now that the rule n is not in L and there exists in S an E-critical pair $\langle\langle p, q\rangle\rangle$ whose right member q is R^E-irreducible. Then an *extended rule* for n is added to N, obtained from the *extended pair* $\langle\langle g[v \leftarrow l], g[v \leftarrow r]\rangle\rangle$ which must be directed from left to right, since $l > r$ and $>$ is compatible with the term structure. As Peterson and Stickel's associative-commutative ones, extended rules reduce at the outermost occurrence all right members of E-critical pairs in S, since q is an instance of $g[v \leftarrow l]$ modulo $\sim$:

$$q = \theta d \sim \theta g = \theta(g[v \leftarrow g/v]) = \theta g[v \leftarrow \theta(g/v)] \sim \theta g[v \leftarrow \theta l] = \theta(g[v \leftarrow l]),$$

using the homomorphic properties of substitutions.

As a consequence, no protection is needed for the other coherence pairs of $l \to r$ on $g \to d$ at occurrence v. Note in addition that an extended rule is a particular case of a rule in N reducing q at the outermost occurrence. They are therefore implicitly protected.

A similar problem arises with coherence pairs coming from the superposition of a rule in L with an equation. Under the same circumstances as above, nonleft-linear rules will be protected or extensions added. However, the extensions are much simpler here: it is the rule $p \to q$ or $q \to p$, according to which set, $\mathrm{CP}(E, L)$ or $\mathrm{CP}(L, E)$, the coherence pair $\langle\langle p, q\rangle\rangle$ belongs to (the right-hand side of the coherence pair must be reducible). These extensions may belong to L and/or to N but do not need any protection, since they reduce p (or q) without using any E-equality steps as before.

Let $\mathrm{Ext}(n)$ be the set of those rules added by the procedure for coherence of n or of a rule in $\mathrm{Ext}(n)$. When the left-hand side of rule n becomes reducible, rule n and all the nonprotected rules in $\mathrm{Ext}(n)$ are removed from the current rewriting system, which may require a complex data structure. More generally, when a new rule $l \to r$ is introduced by the process, the other rules are checked for simplification. A rule $l' \to r'$ is called *simplifiable* by $l \to r$ iff:

• l' is a true instance of l (and not an E-instance) or a strict subterm of l' is an instance of l (or an E-instance if $l \to r \in N$). In other words, l' is reducible at the outermost occurrence or one of its strict subterms is E-reducible.

• It can be protected (an extended rule is a particular case of a protected rule), but only for coherence of rules simplifiable by $l \to r$.

Note the links between the definition of *simplifiability* of a rule and of *top-reducibility* of a coherence pair. Actually, the problem is the same: we want some reducibility property to remain after some rules have been deleted.

The procedure E-COMPLETION is given in Fig. 4.1 in a tail recursive form that makes it easy to understand, to prove and to transform into an efficient iterative procedure.

To be sure that a rule cannot be ignored indefinitely in the selection process of the ELSE branch of the procedure E-COMPLETION, a *fairness selection hypothesis* is required for the choice of an unmarked rule in R: for any rule labelled k, there exists a recursive call such that:

• Either rule k is reduced by the newly introduced rule and removed from the current set of rules,

• or rule k is selected and CRITICAL-PAIRS (k, R, E, n) is computed.

PROCEDURE E-COMPLETION $(P, R, E, >, n)$
IF P is not empty
THEN choose a pair (p, q) in P; $p' := p\downarrow$; $q' := q\downarrow$;
CASE $p' \sim q'$ THEN $R := E$-COMPLETION $(P - \{(p, q)\}, R, E, >, n)^{\text{recursive call 1}}$
$p' > q'$ THEN $l := p'$; $r := q'$; $(P, R) :=$ SIMPLIFICATION $(P - \{(p, q)\}, R, l \to r)$;
$R := E$-COMPLETION $(P, R \cup \{n: l \to r\}, E, >, n+1)^{\text{recursive call 2}}$
$q' > p'$ THEN $l := q'$; $r := p'$; $(P, R) :=$ SIMPLIFICATION $(P - \{(p, q)\}, R, l \to r)$;
$R := E$-COMPLETION $(P, R \cup \{n: l \to r\}, E, >, n+1)^{\text{recursive call 2'}}$
ELSE STOP with FAILURE
END CASE;
ELSE IF all rules in R are marked
THEN RETURN R; STOP with SUCCESS
ELSE Choose an unmarked rule $m: l \to r$ fairly;
$(n, P, R) :=$ CRITICAL-PAIRS $(m: l \to r, R, E, n)$; Mark the rule $m: l \to r$ in $\mathcal{R}$;
$R := E$-COMPLETION $(P, R, E, >, n)^{\text{recursive call 3}}$
END IF
END IF
END E-COMPLETION

FIG. 4.1. *The E-completion procedure.*

PROCEDURE SIMPLIFICATION $(P, R, l \to r)$
$\mathcal{K} := \{k \text{ in } R \mid k \text{ is simplifiable by } l \to r\}$;
$\mathcal{K}' := \{k \text{ in } \mathcal{K} \mid k \text{ is not an extension rule}\}$;
$P := P \cup \{(l'_k, r_k) \mid k \in \mathcal{K}', l_k \to_{[l \to r]} l'_k\}$;
$R := \{l_k \to r'_k \mid l_k \to r_k \in R, k \notin \mathcal{K}, r_k \overset{*}{\to}^{(R \cup \{l \to r\})^E} r_k\downarrow = r'_k\}$;
RETURN (P, R)
END SIMPLIFICATION

FIG. 4.2. *The simplification procedure.*

PROCEDURE CRITICAL-PAIRS $(m: l \to r, R, E, n)$
IF $m: l \to r \in L$
THEN $P := \cup_{k \leqq m, k \in R} \mathrm{CP}(m, k) \cup \cup_{k \leqq m, k \in L} \mathrm{CP}(k, m) \cup \cup_{k \leqq m, k \in N} \mathrm{ECP}(k, m)$;
FOR any $\langle\langle p, q \rangle\rangle \in \mathrm{CP}(l \to r, E)$ (resp. $\mathrm{CP}(E, l \to r)$)
DO IF q (resp. p) is irreducible THEN $l' := q$; $r' := p\downarrow$; (resp. $l' := p$; $r' := q\downarrow$;)
$(P, R) :=$ SIMPLIFICATION $(P, R, l' \to r')$;
$R := R \cup \{n: l' \to r'\}$; $n := n+1$
ELSE $q \to^{R^E}_{[j]} q'$; $P := P \cup \{(p, q')\}$; (resp. $p \to^{R^E}_{[j]} p'$; $P := P \cup \{(p', q)\}$;)
IF q (resp. p) is *top-reducible* by j THEN Protect j for coherence of m END IF
END IF
END DO
ELSE $P := \cup_{k \leqq m, k \in R} \mathrm{ECP}(m, k) \cup \cup_{k \leqq m, k \in L} \mathrm{CP}(k, m) \cup \cup_{k \leqq m, k \in N} \mathrm{ECP}(k, m)$;
FOR any $g \to d \in E$, $v \in \mathcal{D}(g)$, $v \neq \varepsilon$ such that $\mathcal{U} = \mathrm{CSU}(l, g/v, E) = \varnothing$
DO IF $\exists \sigma \in \mathcal{U}$ such that σd is irreducible
THEN $l' := g[v \leftarrow l]$; $r' := g[v \leftarrow r]$; $(P, R) :=$ SIMPLICATION $(P, R, l' \to r')$;
$R := R \cup \{n: l' \to r'\}$; $n := n+1$
ELSE FOR every σ in $\mathcal{U}$
DO $q := \sigma d \to^{R'}_{[j]} q'$; $P := P \cup \{\sigma(g[v \leftarrow r]), q'\}$;
IF q is *top-reducible* by j THEN Protect j for coherence of m END IF
END DO
END IF
END DO
END IF;
RETURN (n, P, R)
END CRITICAL-PAIRS

FIG. 4.3. *The critical-pairs procedure.*

The SIMPLIFICATION procedure given in Fig. 4.2 transforms a set of rules into a quasi-inter-reduced one. Rules where the left-hand side is *simplifiable* must become new pairs, since their orientation may change. On the other hand, rules whose right-hand side only is reducible remain as rules, since their orientation does not change[4]. Note that simplifiability is a particular case of R^E-reducibility, but that some R^E-reducible rules are not simplifiable: the reason is that some R^E-reducible rules cannot be removed without losing the coherence property. This will be clear from the proof. As a consequence, the resulting set of rules will not be fully inter-reduced.

An efficient implementation of this procedure is not straightforward: it requires a careful computation of both sets $\mathcal{H}$ and $\mathcal{H}'$. Note that extended rules disappear when they become simplifiable.

The procedure CRITICAL-PAIRS described in Fig. 4.3 computes critical pairs, sets protections or adds extension rules if necessary[5] and returns a new set P of pairs of terms.

The E-COMPLETION procedure can stop with failure[6], stop with success or loop forever. In the first case, all that may be said is that every pair or rewrite rule generated so far is an equational consequence of the axioms. It is possible that trying again with another ordering would bring success. We are interested in the two remaining cases, when all pairs considered in P at every recursive call can be compared by $>$, no matter whether the algorithm stops or loops forever, yielding then a semi-decision procedure for $R \cup E$-equality. These two cases are considered in the forthcoming subsection in a way similar to [33].

4.2. A complete proof of correctness. Let us introduce the following notations, consistent with those of [33]:

- P_i and R_i denote the values of arguments P and R at the ith recursive call. Since the recursive calls of the completion procedure are tail recursive, proving properties of the procedure will be easy by induction: assuming a property is true at the ith recursive call, we will prove it for the next recursive call by checking what happens along each path from the beginning to a tail recursive call. We will use the notation *case n* to indicate that we are interested in the path to the tail recursive call *n*. Since the two recursive calls 2 and 2′ are symmetrical, we will always show the case 2 only.
- $\mathcal{R} = \bigcup_i R_i$ is the set of all rules generated during the process. $\mathcal{R}$ is split into left-linear rules $\mathcal{L}$ and nonleft-linear ones $\mathcal{N}$.
- $R_\infty = \{l \to r \text{ in } \mathcal{R} \mid \exists i \forall j > i,\ l \to r \text{ is in } R_j\}$. In other words, R_∞ is the set of those rules which are never reduced, neither on their left-hand side nor on their right-hand side by other rules. If the completion procedure stops, R_∞ is its result and is finite. If it does not stop, R_∞ is infinite and is the limit of R. R_∞ is also split into a set of left-linear rules and a set of nonleft-linear ones. Since these subsets are the limits of L and N, they are respectively denoted by L_∞ and N_∞.

According to these different sets, we consider two reduction relations: $\mathcal{R}^E$ is $\mathcal{L} \cup \mathcal{N}, E$ and R_∞^E is $L_\infty \cup N_\infty, E$. Note that R_∞ is included into $\mathcal{R}$, R_∞ / E is included into $\mathcal{R}/E$, R_∞^E is included into $\mathcal{R}^E$. Since $\mathcal{R}$ is E-terminating, so is R_∞.

As in [33], the first part of the proof consists in stating that $\mathcal{R}$ and E generate the original equational theory $\overset{*}{\leftrightarrow}$.

[4] Marked and/or protected rules remain marked and/or protected after reduction.

[5] An extension rule $l' \to r'$ of a rule $l \to r$ must be unmarked and added to L or N, according to its left-linearity. It must be protected only when the rule $l \to r$ belongs to N.

[6] As indicated in [54], some cases can be solved by adding new function symbols, or by choosing another pair as in [23].

LEMMA 18. $\forall i \geqq 0$, (a) $\sim^{R_{i+1}} \subseteq \sim^{P_i \cup R_i \cup E}$ *and* (b) $\sim^{P_{i+1}} \subseteq \sim^{P_i \cup R_i \cup E}$.

Proof. By case analysis, according to the possible tail recursive calls in *E*-COMPLETION.

Case 1. $P_{i+1} = P_i - \{(p, q)\}$ and $R_{i+1} = R_i$ and both (a) and (b) are obvious, since normalization is a particular case of equational deduction.

Case 2 *and* 2'. $(P_{i+1}, R_{i+1}) = \text{SIMPLIFICATION}\ (P_i - \{(p, q)\}, R_i, l \to r)$. Therefore, $P_{i+1} = P_i - \{(p, q)\} \cup \{(l'_k, r_k) \mid k \text{ in } \mathcal{K}'\}$ and $R_{i+1} = \{l_k \to r'_k \mid k \notin \mathcal{K}\} \cup \{l \to r\}$. If $t \sim^{R_{i+1}} t'$, *then* $t \sim^{P_i \cup R_i \cup E} t'$, since $l \sim^{P_i \cup R_i \cup E} r$ and $l_k \sim^{R_i \cup \{l \to r\} \cup E} r'_k$ and thus $l_k \sim^{P_i \cup R_i \cup E} r_k$. If $t \sim^{P_{i+1}} t'$, then $t \sim^{P_i \cup R_i \cup E} t'$, since $l'_k \sim^{R_i \cup \{l \to r\}} r_k$ and thus $l'_k \sim^{P_i \cup R_i \cup E} r_k$.

Case 3. $(P_{i+1}, R_{i+1}) = \text{CRITICAL-PAIRS}\ (m, R_i, Em)$. Then $\sim^{P_{i+1}} \subseteq \sim^{P_i \cup R_i \cup E}$, since critical pairs are computed with respect to rules of equational deduction. As new rules in R_{i+1} are extended pairs that are also computed with respect to rules of equational deduction, $\sim^{R_{i+1}} \subseteq \sim^{P_i \cup R_i \cup E}$. □

COROLLARY. $\sim^{\mathcal{R} \cup E} \subseteq \overset{*}{\vdash\!\dashv}$.

Proof. Easy consequence of Lemma 18. □

Note that the converse requires a precise proof, since some rules may have been deleted during the completion process.

The next two lemmas are an important part of the proof. They assert that the $\mathcal{R}/E$-confluence modulo E of any pair put in P is ensured.

LEMMA 19. $\forall i \geqq 0$, $\forall \langle\langle p, q \rangle\rangle \in P_i$, $\langle\langle p, q \rangle\rangle$ *will be selected at some recursive call* $j > i$.

Proof. This is equivalent to proving that there exists $k > i$ such that P_k is empty. To each recursive call i, let us associate the multiset of terms denoted $\{P_i, R_i\}$ built from the left- and right-hand sides of all pairs of terms in P_i and R_i. The number of axioms in P_i and rules in R_i are denoted by $|P_i|$ and $|R_i|$ respectively. Let us now consider the following ordering on these multisets: $\{P_i, R_i\} \gg \{P_j, R_j\}$ iff

a) $|P_i| + |R_i| > |P_j| + |R_j|$, or else

b) $|P_i| + |R_i| = |P_j| + |R_j|$ and $|P_i| > |P_j|$, or else

c) $|P_i| + |R_i| = |P_j| + |R_j|$ and $|P_i| = |P_j|$ and $\{P_i, R_i\} >_{\text{mult}} \{P_j, R_j\}$,

where $>_{\text{mult}}$ is the multiset extension of the *E*-reduction ordering $>$ used in the procedure *E*-COMPLETION. This ordering is well founded, since it is a lexicographic combination of well-founded orderings.

Let us now prove that $\{P_i, R_i\} \gg \{P_{i+1}, R_{i+1}\}$ by case analysis:

Case 1. $P_{i+1} = P_i - \{\langle\langle p, q \rangle\rangle\}$ and $R_{i+1} = R_i$. Then $|P_i| + |R_i| > |P_{i+1}| + |R_{i+1}|$.

Case 2 *or* 2'. $P_{i+1} = P_i - \{\langle\langle p, q \rangle\rangle\} \cup \{(l'_k, r_k) \mid k \in \mathcal{K}'\}$ and $R_{i+1} = \{l_k \to r'_k \mid k \notin \mathcal{K}\} \cup \{l \to r\}$. If $\mathcal{K}'$ is a strict subset of $\mathcal{K}$, $|P_i| + |R_i| > |P_{i+1}| + |R_{i+1}|$. If $\mathcal{K} = \mathcal{K}'$, $|P_i| + |R_i| = |P_{i+1}| + |R_{i+1}|$. If $\mathcal{K}$ is empty, $|P_i| > |P_{i+1}|$, else $\mathcal{K}$ has at least one element which is a $\mathcal{R}^E$-reducible term of $\{P_i, R_i\}$ and in this case $\{P_i, R_i\} >_{\text{mult}} \{P_{i+1}, R_{i+1}\}$, since each rule in $\mathcal{R}$ has been checked for termination with $>$. □

LEMMA 20. $\forall i \geqq 0$, $\forall \langle\langle p, q \rangle\rangle \in P_i$, $\exists p', q'$ *such that* $p \overset{*}{\to}^{\mathcal{R}/E} p'$, $q \overset{*}{\to}^{\mathcal{R}/E} q'$ *and* $p' \sim q'$.

Proof. From the previous lemma, we know that any pair $\langle\langle p, q \rangle\rangle$ in P_i is selected at some recursive call j. It satisfies the claimed property, since either $p\downarrow \sim q\downarrow$, or $p\downarrow \to q\downarrow$, or $q\downarrow \to p\downarrow \in R_{j+1}$. □

We are now able to state the first part of the proof of the completion procedure:

PROPOSITION 21. $\overset{*}{\vdash\!\dashv} = \sim^{\mathcal{R} \cup E}$.

Proof. From the previous lemma applied with $i = 0$, it follows that $\sim^R$, and therefore $\overset{*}{\vdash\!\dashv}$, is included into $\sim^{\mathcal{R} \cup E}$. The result then follows from the last corollary. □

The second part of the proof consists of proving the Church-Rosser property for R_∞. It will follow from the fact that the critical pairs of $\mathcal{R}$ satisfy the properties of Theorem 16.

Before stating and proving the Church-Rosser property of R_∞, we need to prove the well-foundedness of the reduction relation used in the proof. Let $\mapsto$ be the relation $\to_{\mathcal{R}/E} \cup >^{sst/E} \cup >^{sub/E}$, where $>^{sub/E}$ is the strict ordering associated with the so-called *E-subsumption preordering* defined as follows: $s \geqq^{sub/E} t$ iff $\exists s', t'$ such that $s \sim s'$, $t \sim t'$ and $s' = \sigma t'$ for some substitution σ. We assume that $>^{sub/E}$ is well founded. As already noted, this is usually the case when the theory E has a finite unification algorithm. The rest of the termination proof is based on additional commutation lemmas.

LEMMA 22. *$s \overset{*}{>}^{sub/E} t' \overset{+}{\to}_{\mathcal{R}/E} t \Rightarrow \exists s' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $s \overset{+}{\to}_{\mathcal{R}/E} s' \overset{*}{>}^{sub} t$.*

Proof. The proof is similar to that of lemma 14 and is left to the reader. □

LEMMA 23. *$s \overset{*}{>}^{sst/E} t \overset{*}{>}^{sub/E} t' \Rightarrow \exists s', s'' \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ such that $s \overset{*}{>}^{sub/E} s' \sim s'' \overset{*}{>}^{sst} t'$.*

Proof. The proof is by induction on the number of steps from s to t. If $s = t$, the result is obvious. If $t = t'$, the result comes from Lemma 13. Otherwise $s \overset{*}{>}^{sst/E} s_1 >^{sst/E} t >^{sub/E} t_2 \overset{*}{>}^{sub/E} t'$. By definition, $\exists t_1 \in \mathcal{T}$ and $\sigma \in \Sigma$ which is not a renaming modulo E, such that $t \sim \sigma t_1 >^{sub} t_1 \sim t_2$, therefore $t \sim \sigma t_2 >^{sub} t_2$. Since the same process works with the $>^{sst}$ relation as already noted in Lemma 13, we get: $s_1 \sim s_2 >^{sst} \sigma t_2 >^{sub} t_2$.

We can now make $>^{sst}$ and $>^{sub}$ commute. By definition, $\exists v$ such that $s_2/v = \sigma t_2$. Assume without lost of generality that $\mathcal{V}(t_2) \cap \mathcal{V}(s_2) = \varnothing$, therefore $\mathcal{D}(\sigma) \cap \mathcal{V}(s_2) = \varnothing$ (otherwise, we must perform a variable renaming on t_2, which eventually renames the variables of t' at the end). Now, $s_2 = \sigma(s_2[v \leftarrow t_2])$ and thus $s_2 >^{sub} s_2[v \leftarrow t_2] >^{sst} t_2$. The result is finally obtained by pushing the $>^{sst}$ step to the end using a first application of the induction hypothesis from $s_2[v \leftarrow t_2]$ to t' and then by a second application of the induction hypothesis to the whole chain except the last $>^{sst}$ steps. □

As a consequence of these lemmas, we get the following termination result:

PROPOSITION 24. *Assume R is E-terminating, E-congruence classes are finite and the strict E-subsumption preordering is well founded. Then $\mapsto$ is well founded.*

Proof. Assume given an infinite chain of $\mapsto$. From Lemmas 14, 22 and 23, it follows that actually this chain can be written as $\overset{*}{\to}_{\mathcal{R}/E} \circ \overset{*}{>}^{sub} \circ \overset{*}{>}^{sub/E} \circ \overset{*}{>}^{sst} \circ \overset{*}{>}^{sst/E}$. The proof is then similar to the proof of Proposition 15, using that the $>^{sub/E}$ relation is well founded by hypothesis. □

We may now make use of this well-founded relation $\mapsto$ to prove the main theorem of this section.

THEOREM 25. *Assume that a complete and finite unification algorithm exists for the theory E, E-congruence classes are finite and the strict E-subsumption preordering is well founded. Then R_∞ is R_∞^E-Church-Rosser modulo E, provided that the fairness selection hypothesis is satisfied.*

Proof. Let us denote by $t\!\downarrow$ the R_∞^E-normal form of t, and write $s\!\downarrow t$ iff $\exists s', t'$ such that $s \overset{*}{\to}_{R_\infty^E} s'$, $t \overset{*}{\to}_{R_\infty^E} t'$ and $s' \sim t'$, and $s\!\downarrow_{\mathcal{R}/E} t$ iff $\exists s'$ such that $s \overset{*}{\to}_{\mathcal{R}/E} s'$ and $t \overset{*}{\to}_{\mathcal{R}/E} s'$. Since R_∞ is included into $\mathcal{R}$, we may actually prove that: $t_0 \overset{*}{\mapsto}^{\mathcal{R} \cup E} t_{n+1}$ implies $t_0\!\downarrow \sim t_{n+1}\!\downarrow$. *Let* $M = \{t_0, \cdots, t_n, t_{n+1}\}$ such that $\forall i \in [0 \cdots n]$, $t_i \mapsto^{\mathcal{R} \cup E} t_{i+1}$. The proof is very similar to that of the Church-Rosser theorem 16, and works by multiset induction on $\mapsto$. Three cases have to be distinguished as usual. Each one uses a particular local property defined next, thanks to the following notation: $s_0 \Downarrow s_m$ iff $s_0 \mapsto^{\mathcal{R} \cup E} s_1 \mapsto^{\mathcal{R} \cup E} \cdots \mapsto^{\mathcal{R} \cup E} s_m$ such that $\exists j \in [0 \cdots m+1]$, $\forall i \in [0 \cdots m]$, $t_j \overset{+}{\mapsto} s_i$. As a consequence, the whole multiset $\{s_0, \cdots, s_m\}$ is strictly smaller than $\{t_j\}$, thus smaller than M. This will enable us to apply multiset induction on $\{s_0, \cdots, s_m\}$. Note in addition that $\Downarrow$ is transitive for a given t_j and that $s \mathcal{R}^E s'$, $s\!\downarrow s'$ and $s\!\downarrow_{\mathcal{R}/E} s'$ all imply $s \Downarrow s'$, provided s and s' are proper sons of t_j for $\mapsto$.

- $t_n \mapsto t_{n+1}$: if t_n is R_∞^E-irreducible, then so is t_{n+1} by the *local coherence* property of R_∞^E that we define next using the previous notations. The result follows from

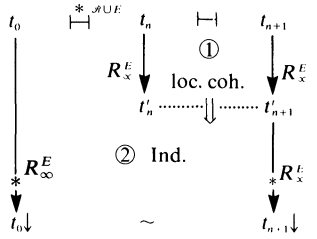


FIG. 4.4. *Church-Rosser Proof using local coherence.*

induction hypothesis applied to the multiset $\{t_0, \cdots, t_n\}$. Otherwise, *local coherence* of R_∞^E is applied first as shown on Fig. 4.4. and then induction to the multiset $\{t_0, \cdots, t_n, t'_n, \cdots, t'_{n+1}\}$ which is strictly smaller than M, since the terms between t'_n and t'_{n+1} are all proper sons of t_{n+1} for $\mapsto$ by definition of $\Downarrow$. This is step 2 in the same figure.

- $t_{n+1} \to^{\mathcal{R}} t_n$: the result follows from *reducibility of* $\mathcal{R}^E$ by R_∞^E, which then allows us to apply the induction hypothesis to the multiset $\{t_0, \cdots, t_n, \cdots, t'_{n+1}\}$ whose terms between t_n and t'_{n+1} are all smaller than t_{n+1} for $\mapsto$. This is shown on Fig. 4.5.

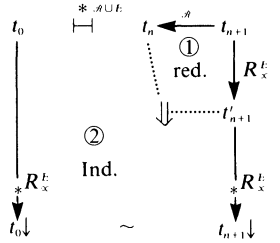


FIG. 4.5. *Church-Rosser Proof using reducibility.*

- $t_n \to^{\mathcal{R}} t_{n+1}$: we apply first the induction hypothesis on the multiset $\{t_0, \cdots, t_n\}$. Since t_n is not in $\mathcal{R}$-normal form, it is not in R_∞^E-normal form by the reducibility property, making clear step 1 in Fig. 4.6. The proof then follows from *local confluence of* R_∞^E with $\mathcal{R}$ (step 2) and induction on the multiset $\{t'_n, \cdots, t_{n+1}\}$ (step 3).

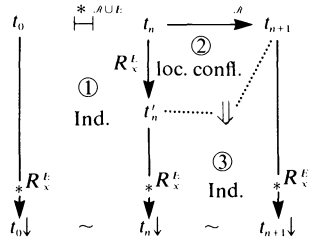


FIG. 4.6. *Church-Rosser Proof using local confluence.*

We are now left to prove the three properties of reducibility, local coherence and local confluence, assuming the Church-Rosser induction hypothesis on multisets strictly smaller than M.

Let us first prove reducibility of $\mathcal{R}^E$ by R_∞^E, specifically:

$\forall t$, $t \sim t_{n+1}$ and $t \to^{\mathcal{R}}_{[\varepsilon,\sigma,k:l\to r]} t'_1$ or $t \to^{\mathcal{R}^E}_{[v\neq\varepsilon,\sigma,k:l\to r]} t'_1$, $\exists t'_2$, $t \to^{R_\infty^E} t'_2$ and $t'_1 \Downarrow t'_2$.

Note that standard matching is required here for reduction of t at the outermost occurrence, but not for reduction of a strict subterm of t.

- *Case* 1. Assume $v \neq \varepsilon$. Then t/v is a proper son of t for $\mapsto$ and the induction hypothesis may be applied to the multiset $\{t/v, t'_1/v\}$ yielding $t/v\downarrow \sim t'_1/v\downarrow$. The property is then easily lifted to $\{t, t'_1\}$ yielding the result since $\mathcal{R}$ is E-terminating and thus t must be different from $t\downarrow$.
- *Case* 2. Assume $v = \varepsilon$ and σ is not a variable renaming. Then l is a proper son of t for $\mapsto$ and the same reasoning as in Case 1 applies. For the remainder of the proof, v can be supposed to be ε and σ to be a variable renaming, hence $t = \sigma l$ and $t'_1 = \sigma r$.
- *Case* 3. Some rule labelled k belongs to R_∞, say $l \to r'$ with $r \overset{*}{\to}^{\mathcal{R}^E} r'$, therefore $t \to^{R_\infty}_{[\varepsilon,\sigma,k:l\to r']} t'_2 = \sigma r'$ and $t'_1 \overset{*}{\to}^{\mathcal{R}^E} t'_2$, hence $t'_1 \Downarrow t'_2$.
- *Case* 4. The left-hand side l of the rule k has been reduced by another rule $l'' \to r''$, therefore $l/\nu \sim \tau l''$ for some occurrence $\nu \neq \varepsilon$ and substitution τ or $l/\varepsilon = \tau l''$, since left members of rules are not reduced at the outermost occurrence modulo E. Let $l' = l[\nu \leftarrow \tau r'']$. Since (l', r) has been dropped to P, by Lemma 20, the pair (l', r) has become convergent at some further recursive call, therefore $l' \Downarrow r$. Let $t \to^{\mathcal{R}^E}_{[\nu,\sigma\tau,l''\to r'']} t''_1$. Since $t'_1 = \sigma r$ and $t''_1 = t[\nu \leftarrow \sigma\tau r''] = \sigma(l[\nu \leftarrow \tau r''])$, we have $t'_1 \Downarrow t''_1$. We are therefore left with proving the reducibility for the pair of terms $t \to^{\mathcal{R}^E}_{[\nu,\sigma\tau,l''\to r'']} t''_1$. Let us suppose that τ is a variable renaming and ν is ε. Then l'' is an instance of l and the rule $l'' \to r''$ would never have been generated. Therefore $\nu \neq \varepsilon$ or τ is not a variable renaming, we can apply the already proved cases 1 and 2 and we are done.

Note the crucial fact of l being an instance (and not an E-instance) of l'' when l'' reduces l at the outermost occurrence. Otherwise, it would have been necessary to use the (not yet proved) local coherence property to lift the reducibility of $\tau l''$ to l in Case 2, where the induction uses the strict E-subsumption ordering (and by transivity in Case 4).

Let us now prove local coherence of R_∞^E, specifically:

$\forall t, t', t''$ *such that* $t_n \sim t$, $t \vdash\!\!\!\mapsto_{[v,\sigma,g\to d]} t''$ and $t \to^{R_\infty^E}_{[\nu,\sigma,l\to r]} t'$, $\exists t''_1$ such that $t'' \to^{R_\infty^E} t''_1$ and $t''_1 \Downarrow t''$.

The cases to be now discussed according to the respective positions of v and ν are the same as in Theorem 16. Only Cases 5, 7 and 8 differ. Numbering and notation remain exactly the same:

- *Case* 5: $\nu = \varepsilon$, with $l \to r$ in L_∞, and $v \in \mathcal{G}(l)$. By the fairness hypothesis, the critical pairs of $l \to r$ and $g \to d$ have been computed at some iteration j. By

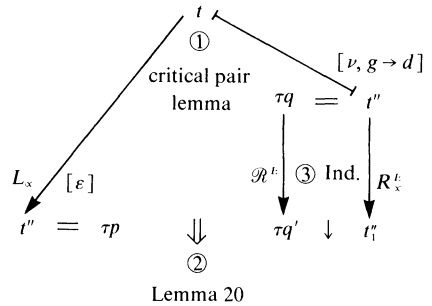


FIG. 4.7. *Local coherence proof of R_∞^E, Case* 5a.

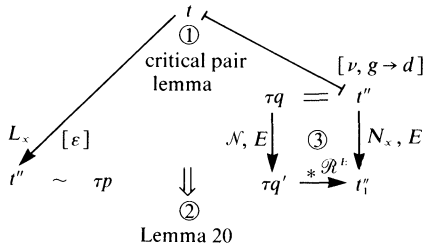


FIG 4.8. *Local coherence proof of* R^E_∞, *Case* 5b.

the Critical Pairs Lemma, there exists a critical pair of $\langle\langle p, q\rangle\rangle$ in $\mathrm{CP}(E, L_j)$ and a substitution τ such that $t' = \tau p$, $t'' = \tau q$. This is step 1 in Figs. 4.7 and 4.8. From the procedure CRITICAL-PAIRS, we know that $q \to^{\mathscr{R}^E}_{[\omega,\theta,k':l'\to r']} q'$. The rule $l' \to r'$ can be the added rule $q \to p\downarrow$ and in that case $p\downarrow = q'\downarrow$, or the pair (p, q') has been dropped into P. In this last case, we know from Lemma 20 that $p \downarrow_{\mathscr{R}/E} q'$, therefore $p \Downarrow q'$. As a consequence, in both cases, we have $\tau p \Downarrow \tau q'$. This is step 2 in Fig. 4.7 and Fig. 4.8. The rest of the proof is now split into three cases:

- *Case* 5a. $\omega \neq \varepsilon$. In that case, the induction hypothesis is applied to the multiset $\{\tau q/\omega, \tau q'/\omega\}$ yielding $\tau q/\omega\downarrow \sim \tau q'/\omega\downarrow$. As a consequence, the pair $(t'', \tau q')$ satisfies the same property. Since $\mathscr{R}$ is E-terminating, there must exist a term t''_1 such that $t'' = \tau q \to^{R^E_\infty} t''_1$ and $\tau q'\downarrow t''_1$. This is step 3 on Fig. 4.7.
- *Case* 5b. $\omega = \varepsilon$ and q is not an instance of l' but an E-instance. Then $l' \to r' \in \mathscr{N}$ and has been protected (it may be an extended rule). Since $l \to r$ is in L_∞, a rule $k' : l' \to r''$ must be in N_∞ with $r' \to^{*\,\mathscr{R}^E} r''$, therefore $\tau q' = \tau\theta r' \to^{*\,\mathscr{R}^E} \tau\theta r''$. Now $t'' = \tau q \sim \tau\theta l' \to^{N_\infty}_{[\varepsilon,\tau\theta,k']} \tau\theta r''$. This is step 3 on Fig. 4.8, with $t''_1 = \tau\theta r''$.
- *Case* 5c. $\omega = \varepsilon$ and $l' \to r' \in \mathscr{L}$ or $l' \to r' \in \mathscr{N}$ and q is a true instance of l'. Then we conclude from the already proved reducibility property that $t'' \to^{R^E_\infty} t''_1$, $\tau q' \Downarrow t''_1$ and we are done.

- *Case* 6. $v = \varepsilon$, $\nu \neq \varepsilon$, and $\nu \in \mathscr{G}(g)$ with $l \to r$ in L_∞. Similar reasoning as in Case 5.
- *Case* 7. $v = \varepsilon$, $\nu \neq \varepsilon$ and $\nu \in \mathscr{G}(g)$ with $l \to r$ in N_∞. By the fairness hypothesis and the E-Critical Pairs Lemma, there exist an iteration j, a pair $\langle\langle p, q\rangle\rangle = \langle\langle \theta(g[\nu \leftarrow r]), \theta d\rangle\rangle$ in $\mathrm{ECP}(N_j, E)$ and a substitution τ such that $\sigma \sim \tau\theta[\mathscr{V}(g) \cup \mathscr{V}(l)]$, therefore $t'' \sim \tau q$ with E-equality steps taking place outside of $\mathscr{G}(d)$ and $t' \sim \tau p$. From the CRITICAL PAIRS procedure, we know that $q \to^{\mathscr{R}^E}_{[\omega,\theta,k':l'\to r']} q'$ and, as in Case 5, $\tau p \Downarrow \tau q'$. This is step 1 in Fig. 4.9.The next step in the proof allows us to transform the $\mathscr{R}^E$-rewriting from τq to $\tau q'$ into

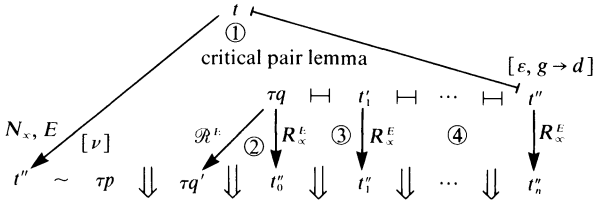


FIG. 4.9. *Local coherence proof of* R^E_∞, *Case* 7.

a R_∞^E-rewriting from τq to some t_0'' related to $\tau q'$. Two cases are to be distinguished:

- If $\omega = \varepsilon$, $k' : l' \to r' \in \mathcal{N}$ and q is not a true instance of l', then k' has been protected. Since $l \to r$ is in N_∞, a rule $k' : l' \to r''$ must be in N_∞ with $r' \xrightarrow{*}_{\mathcal{R}^E} r''$. Let now $t_0'' = \tau\theta r''$. Then $\tau q \to^{N_\infty, E}_{[\varepsilon, \tau\theta, k']} t_0''$, and $\tau q' = \tau\theta r' \xrightarrow{*}_{\mathcal{R}^E} t_0''$, therefore $\tau q' \Downarrow t_0''$.
- If k' applies at occurrence $\omega \neq \varepsilon$ or if q is a true instance of l', then the already proved reducibility property applies and yields $\tau q \to^{R_\infty^E} t_0''$ for some t_0'' and $\tau q' \Downarrow t_0''$. This is step 2 in Fig. 4.9.

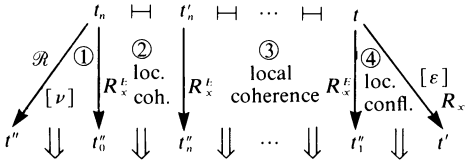


FIG. 4.10. *Local confluence proof of R_∞^E with $\mathcal{R}$, Case* 4.

The remainder of the proof is then very similar to the corresponding one in Theorem 16, applying the already proved local coherence property until t'' is reached, since no equality step from τq to t'' takes place at occurrence ε.

Let us now prove local confluence of R_∞^E with $\mathcal{R}$, specifically:

For any t' and t'' such that $t_n \to^{R_\infty^E}_{[\nu, \sigma, l \to r]} t'$ and $t_n \to^{\mathcal{R}}_{[\upsilon, \sigma, g \to d]} t''$, $t' \Downarrow t''$.

We only point out what differs from the proof of Theorem 16. Cases 1 and 2 are the same. Case 3 is the same when there is no overlapping, otherwise the fairness selection hypothesis and Lemma 20 are used as in the local coherence proof. Case 4 uses a first reducibility step to make appear a R_∞^E-reduction from t_n instead of a $\mathcal{R}$-reduction; the rest of the proof is entirely similar. □

It must be noted that the proofs of reducibility, local coherence and local confluence use the induction hypothesis and cannot be proved as independent lemmas. This explains the complexity of the whole proof. As a corollary of Theorem 25 and Proposition 21, we obtain under the same assumptions:

THEOREM 26.

- *R_∞ is R_∞^E-convergent modulo E.*
- *$\mathcal{R}$ is $\mathcal{R}^E$-convergent modulo E.*
- *The congruences $\overset{*}{\leftrightarrows}$, $\sim^{\mathcal{R} \cup E}$ and $\sim^{R_\infty \cup E}$ are equal.*

As in Theorem 16, we conjecture that some hypotheses could be removed, precisely the two hypotheses about the finiteness of the E-congruence classes and the well-foundedness of the strict E-subsumption preordering.

Let us finally prove that the E-completion algorithm can always be considered as a semi-decision procedure for $\overset{*}{\leftrightarrows}$, assuming that a decision procedure for $\sim$ is known. As in the standard completion algorithm, if the two terms to be checked for equality are not in the same congruence class, then the semi-decision procedure will loop forever if the E-completion procedure loops forever.

PROPOSITION 27. *$s \overset{*}{\leftrightarrows} t$ iff there exists a recursive call i such that $s \downarrow_{R_i^E} s'$, $t \downarrow_{R_i^E} t'$ and $s' \sim t'$ for some s' and t'.*

Proof. It follows the lines of the proof of the corresponding result in [33] obtained for the case of an empty E. □

This result can be extended in a straightforward way to semi-decide that two sets of axioms (R, E) and $(\mathcal{R}, E)$ generate the same equational theory, provided the

hypotheses of Theorem 25 are satisfied: we can start the procedure E-COMPLETION to check that all axioms in $\mathcal{R} \cup E$ are equational consequences of $R \cup E$. If this is the case, the process will stop after a finite number of calls, even if the E-COMPLETION procedure loops forever. Then we can start the symmetric check. However, this process of checking two sets of axioms for equivalence is time consuming and we will see in § 4.4 an efficient and elegant way of achieving the same goal when the E-COMPLETION procedure provides a R^E-Church–Rosser set of rules after a finite number of steps.

4.3. Protection versus inter-irreducibility. Let us now discuss some issues regarding our E-COMPLETION procedure, especially the tradeoff between protecting rules for coherence and reducing rules for the generated set of rules to have a uniqueness property as in the standard case [65].

Compared to the other known completion algorithms [54], [57], [58], [73], [32], [39], the main new difficulty here arises from the coherence property of the rewriting relation R,E. In Knuth and Bendix's completion algorithm, proved in [33], E is empty and of course, no coherence property is needed. The completion algorithm modulo E, designed in [32], uses standard rewritings with left-linear rules only. Our proof makes clear that we do not need to protect any rule if N is empty. On the other hand, for each rule that has an associative-commutative function symbol at the outermost occurrence of its left-hand side, Peterson and Stickel systematically introduce its associative-commutative extensions that are implicitly protected and thus enforce the coherence property.

Various strategies to deal with the problem of coherence can be imagined. Let us discuss some of them with their respective drawbacks and advantages:

- Our aim was to avoid introducing extensions of rules if the coherence property was ensured by an already existing rule. The lack of power of the relation R,E compelled us to protect some nonleft-linear rules. Once protected, a rule cannot be removed from the rewriting system, even if its left-hand side becomes reducible, unless the rule it was protecting has disappeared. Of course, the resulting rewriting system may not be entirely inter-reduced.
- However, systematically adding extensions can be very interesting for some cases: assume that whenever a nonleft-linear rule and an equation E-overlap at occurrence v, the adequate extension of this rule is added. We do not need computing a complete set of E-overlappings, since this extension automatically reduces all the corresponding E-critical pairs of this rule on the equation at occurrence v: we only need to know that E-unification is possible at this occurrence. In many cases this strategy will speed up the completion process. Note also that in this case, we do not need to protect any rule, except of course the extensions, since we are no longer interested in finding an existing rule which reduces the adequate member of the coherence pairs. As a main drawback of this method, extensions can be many and redundant, and cannot be reduced.
- Another approach consists in modifying the E-COMPLETION procedure by protecting both left-linear and nonleft-linear rules that reduce a coherence pair at the outermost occurrence under the same conditions as previously. This allows local coherence proofs without the use of reducibility in Theorem 25, Case 5c of the local coherence proof becoming similar to Case 5b which makes no use of reducibility. As a consequence, the reducibility proof can now be performed after the local coherence proof and can make use of it. As noted at the end of Case 4 of that proof, reducibility becomes satisfied now, even if left members of rules are reduced modulo E in the

SIMPLIFICATION procedure, provided the rule $l \to r$ is added to N. This will yield a set R_∞ of rules that can be expected to be more inter-reduced than previously.

• A fourth method can even be designed which yields a set R_∞ of completely inter-reduced rules. This can be done by another modification of the SIMPLIFICATION procedure: if the left-hand side of a rule k, protected for coherence of a rule k' with an equation $g = d$, becomes reducible, then k is removed from the rewriting system and the E-critical pairs of k with $g = d$ are computed again. If the process terminates, we are sure that all the critical pairs have been computed, and thus the coherence of k' is ensured. But this strategy does not satisfy the fairness assumption hypothesis when the procedure loops and the infinite set R_∞ is no longer guaranteed to be R_∞^E-Church-Rosser modulo E. However, we could modify the fairness selection hypothesis to take into account the fact that some rule would have to be chosen infinitely many times and some others only finitely many times with respect to a generalized fairness selection hypothesis. Such algorithms already exist which were designed for the purpose of implementing concurrent procedures.

Finally, let us emphasize that all our results remain valid, except the decidability results, if the theory E has a complete unification algorithm that may sometimes return an infinite set of unifiers, provided it returns a finite one at each call. For this case, we might want the E-unification procedure to be called only when it is really needed, in order to minimize the possibility of starting an infinite computation because of the E-unification process. This suggests use of the variant that adds extensions in a systematic way. Note in addition that we might require that the E-unifiers be computed a finite number at a time if there are infinitely many, in order to respect the fairness assumption hypothesis and still have a semi-decision procedure.

4.4. Complete sets of rules in normal form. As seen in the last section, the completion procedure may give as a result a Church-Rosser set of rules that are not fully inter-reduced. As a consequence, we cannot expect such a set to have a uniqueness property as in the standard case [13], [65], [59]: the result of the completion procedure will depend upon a particular presentation of the theory as well as upon the implementation of the choice functions used in the procedure. Our goal here is twofold:

• First, to define precisely what kind of uniqueness can be expected for a Church-Rosser set of rules.

• Second, to give an algorithm to inter-reduce any Church-Rosser set of rules.

Let us say that two sets R and $\mathcal{R}$ of rules are *E-equivalent*, or simply equivalent whenever E is known from the context, iff they generate the same congruence $\overset{*}{\leftrightarrow}_{\mathcal{R}}$.

Assume now a given E-reduction ordering $\geqq$ whose associated strict ordering $>$ is well founded. This ordering can be thought of as the ordering used in the E-COMPLETION procedure to guarantee the E-termination property of the set R of rules. We will say that $\geqq$ is the E-reduction ordering associated to R if this is the case.

As a consequence, whenever the previous goal is achieved, it follows that the E-equivalence problem for two Church-Rosser sets of rules with the same associated E-reduction ordering $\geqq$ is decidable. As a practical consequence, the fact that the resulting Church-Rosser set of rules produced by the E-COMPLETION procedure is not completely inter-reduced is not a severe drawback anymore: just apply the adequate normalization post-processing.

We now define which kind of Church-Rosser set of rules we want to get:

DEFINITION 28. Let R^E be any rewriting relation such that $R \subseteq R^E \subseteq R/E$. An *E-complete set of R^E-normalized reductions* R is a set of rules that has the following properties:

(1). R is E-terminating and R^E-Church-Rosser modulo E.

(2) Any left member of rules is R^E-irreducible except by itself at the outermost occurrence.

(3) The right members of rules are R^E-irreducible.

Any set of rules satisfying (2) and (3) is said to be *inter-reduced.*

We now show that such sets of reductions have a uniqueness property, when R^E is not R/E. To do so, we first introduce a useful technical lemma:

LEMMA 29. *Assume R is R^E-Church-Rosser modulo E and let $\geqq$ be the associated E-reduction ordering. Then a term t is R^E-irreducible iff it is minimal for $\geqq$ in its $\overset{*}{\leftrightarrows}$-congruence class.*

Proof. If t is R^E-irreducible, then for any $t' \overset{*}{\leftrightarrows} t$, $t' \overset{*}{\to}^{R^E} t'\downarrow \sim t\downarrow = t$. Hence, $t' \geqq t$. Conversely, if t is R^E-reducible to s, then $t > s$. □

We are now ready for the main theorem of E-complete sets of normalized reductions:

THEOREM 30. *Let R and $\mathcal{R}$ be two E-complete sets of R^E-normalized reductions where R^E is not R/E. Assume that R and $\mathcal{R}$ have the same associated E-reduction ordering $\geqq$ and generate (with the help of the equations in E) the same congruence $\overset{*}{\leftrightarrows}$. Then R and $\mathcal{R}$ are identical up to E-equality.*

Proof. We prove that for any rule $l \to r$ in $\mathcal{R}$, there exists a (unique) rule $l' \to r'$ in R such that $l \sim l'$ and $r \sim r'$. Since R and $\mathcal{R}$ (with the help of E) present the same equational theory and R is R^E-Church-Rosser and r is $\mathcal{R}^E$-irreducible, thus R^E-irreducible by Lemma 29, $l \overset{*}{\to}^{R^E} \circ \sim r$. Since l is $\mathcal{R}^E$-reducible thus R^E-reducible by Lemma 29 and since all the strict subterms of l are $\mathcal{R}^E$-irreducible, thus R^E-irreducible by the same lemma, $l \to^{R^E}_{[\varepsilon,\sigma,l'\to r'\in R]} \circ \overset{*}{\to}^{R^E} r$.[7] Assume now that σ is not E-equivalent to a variable renaming. The same reasoning shows that l' is $\mathcal{R}^E$-reducible at the outermost occurrence by a rule $l'' \to r''$ of $\mathcal{R}$ which cannot be the rule $l \to r$, since σ is not E-equivalent to a variable renaming. Now, since the second reduction applies at the outermost occurrence of l', l becomes reducible by $l'' \to r''$, which is impossible by the hypothesis that the rules of $\mathcal{R}$ are inter-reduced. Therefore σ is a variable renaming and $l \sim l'$. Since r' and r are supposed to be irreducible, we finally get $r' \sim r$ by the Church-Rosser property and we are done. The uniqueness of the rule $l' \to r'$ then follows from the inter-irreducibility of the rules in R. □

Similar results have already been obtained by [59], [16], for the case where $R^E = R/E$. In this last case however, the hypothesis that E-equivalence classes are finite is needed (see [16] for counter-examples when there are infinite classes). Their results will be used in the following.

The next three lemmas show how to transform an R^E-Church-Rosser set of E-terminating rules into an E-complete set of normalized reductions. This will be done by three different kinds of transformations applied to the starting set, each one preserving a Church-Rosser property. The first two transformations preserve the R^E-Church-Rosser property, while the last one only preserves the R/E-Church-Rosser property. As a consequence, the two first transformations can be usefully applied to the R^E-Church-Rosser set of rules given by the procedure E-COMPLETION. Actually, only the second one is needed, since right members of rules are fully simplified in the procedure SIMPLIFICATION.

LEMMA 31. *Assume R is an E-terminating and R^E-Church-Rosser set of rules. Let $\mathcal{R} = \{l \to r\downarrow \mid l \to r \in R\}$. Then $\mathcal{R}$ is equivalent to R and $\mathcal{R}^E$-Church-Rosser modulo E.*

Proof. $\mathcal{R}$ is clearly E-terminating since R is. In order to prove the result, we just prove that any term has the same normal forms for both rewriting relations. First, they define the same sets of normal forms, since left members of rules are the same and

[7] The reader can check that the reasoning does not apply if R^E is R/E.

are used with the same matching algorithm (which is implicit in our definition of $\mathcal{R}$). Let now t_1 be the $\mathcal{R}^E$-normal form of t. Then t_1 is the R^E-normal form of t since $t \overset{*}{\leftrightarrows} t_1$, R is R^E-Church-Rosser and t_1 is in R^E-normal form. □

LEMMA 32. *Assume R is an E-terminating and R^E-Church-Rosser set of rules, where the theory E is such that E-equivalence classes are finite and the strict E-subsumption preordering is well founded. Let $\mathcal{R}$ be obtained from $R = L \cup N$ by removing any rule from L whose left member is R^E-reducible, or any rule from N whose left member is R^E-reducible at the outermost occurrence by another rule of N. Then $\mathcal{R}$ is equivalent to R and is $\mathcal{R}^E$-Church-Rosser modulo E.*

Proof. $\mathcal{R}$ is clearly E-terminating since R is. As previously, we now prove that any term has the same normal forms for both rewriting relations. This is done here by noetherian induction on the relation $\mapsto$ defined in § 4.2. Let $t \to^{R^E}_{[v,\sigma,l\to r]} t' \overset{*}{\to}^{R^E} t{\downarrow}$. If $v \neq \varepsilon$, the result is easily obtained by induction on t/v. If $l \to r$ is in $\mathcal{R}$, the result is easily obtained by induction on t'.

For the remaining cases, $v = \varepsilon$, therefore $t \sim \sigma l$, and $l \to r$ has been removed, thus $l \to^{R^E}_{[v',\sigma',l'\to r']} l''$, with $l' \to r'$ in $\mathcal{R}$. If $l \to r \in L$, then $t = \sigma l \to^{\mathcal{R}} t'' = \sigma l[v' \leftarrow \sigma' r']$ and the result is obtained by the induction hypothesis applied to t'', together with the R^E-Church-Rosser property of R that forces t' and t'' to have *E-equal* R^E-normal forms. If $l \to r \in N$, then $t \sim \sigma l \to^{\mathcal{R}^E} t'' = \sigma l[v' \leftarrow \sigma' r']$, since $v' = \varepsilon$ in this case by definition of $\mathcal{R}$ and $l' \to r' \in N$. The result is obtained by induction hypothesis applied to t'' and the R^E-Church-Rosser property as before. □

Reducing the starting set of rules as long as possible using Lemmas 31 and 32 clearly yields a set of rules $\mathcal{R}$ satisfying requirements (1) and (3) of Definition 28. Moreover, it will also satisfy the part of requirement (2) concerning left members of rules in L. Now, either the whole set of rules satisfies requirement (2), i.e. left members of rules in N are also $\mathcal{R}^E$-irreducible and the resulting set of rules $\mathcal{R}$ is an E-complete set of $\mathcal{R}^E$-normalized reductions equivalent to the starting one, or it does not and we need to apply as many times as necessary the third transformation studied in the next lemma. If the latter is the case, we will get an E-complete set of $\mathcal{R}/E$-normalized reductions.

LEMMA 33. *Assume R is an E-terminating and R^E-Church-Rosser set of rules, where the theory E is such that E-equivalence classes are finite and the strict E-subsumption preordering is well founded. Let $\mathcal{R}$ be obtained from $R = L \cup N$ by removing any rule in N whose left member has a R^E-reducible strict subterm or is L-reducible at the outermost occurrence. Then $\mathcal{R}$ is equivalent to R and $\mathcal{R}/E$-Church-Rosser.*

Proof. Note first that the new set of rules is still E-terminating. We prove now the two other properties by induction on $\mapsto$ as in the proof of Lemma 32. The two first cases are the same and the case where $v = \varepsilon$ and $l \to r$ has been removed from R requires minor changes: since the E-equality step between t and σl cannot be absorbed anymore by the E-matching step used in the R^E-reduction relation it is absorbed here by the E-equality step used in the $\mathcal{R}/E$-reduction relation. This explains why we only get a $\mathcal{R}/E$-Church-Rosser property. □

Applying this last lemma results in an E-complete set of $\mathcal{R}/E$-normalized reductions that is unique up to E-quality by [16], allowing us to check for equivalence two different sets of axioms expected to be presentations of a same theory. But the complete set of normalized reductions can be used for this purpose only, since it is not $\mathcal{R}^E$-Church-Rosser anymore: for computing in the theory $\mathcal{R} \cup E$, we will use the $\mathcal{R}^E$-Church-Rosser set of rules obtained by applying the first two transformations to the set of rules given by the E-COMPLETION procedure, since the rewriting relation associated with it is more efficient.

5. Examples. Let us give a first example that emphasizes that splitting the set of rules into left-linear rules and nonleft-linear rules allows one to compute less complete sets of E-unifiers, since E-unification is only needed for nonleft-linear rules. It is the well-known example of abelian groups defined by the following set of axioms:

$$(x+y)+z=x+(y+z), \quad x+y=y+x,$$

$$x+0=x, \quad x+i(x)=0.$$

Using the rewriting relation $R^E = R,E$ where E is the subset of the two first axioms, the E-COMPLETION procedure generates the following R_∞:

$$i(0)\to 0, \quad x+0\to x,$$

whose extension is: $z+(x+0)\to z+x$,

$$i(i(x))\to x, \quad x+i(x)\to 0,$$

whose extension is: $z+(x+i(x))\to z$,

$$i(x+y)\to i(y)+i(x).$$

Using the rewriting relation $R^E = L\cup N,E$, The E-COMPLETION procedure generates R_∞ as the union of L_∞:

$$i(0)\to 0, \quad x+0\to x,$$

$$0+x\to x, \quad i(i(x))\to x,$$

$$i(x+y)\to i(y)+i(x)$$

and N_∞:

$$x+i(x)\to 0,$$

whose extension is: $z+(x+i(x))\to z$.

Notice that the additional rule $(0+x\to x)$ is needed when computing with the rewriting relation $L_\infty\cup N_\infty,E$, otherwise the pair $(0+x, x)$ would not be R^E-convergent although it is obviously R/E-convergent. Of course, both systems are equivalent to the same R/E-Church-Rosser set of normalized reductions:

$$i(0)\to 0, \quad x+0\to x,$$

$$i(i(x))\to x, \quad x+i(x)\to 0,$$

$$i(x+y)\to i(y)+i(x).$$

On the other hand, by using a set L of left-linear rules, one gets sometimes into a divergent process that would otherwise converge with all the rules considered to be in N. Let us recall the example of § 3 of a set of rules which is R,E-Church-Rosser modulo E but not $L\cup N,E$-Church-Rosser modulo E: Let $0, 1, -1$ be constant function symbols and $+$ a binary infix symbol which is associative and commutative. Let R be the set of left-linear rules:

(1) $0+x\to x$,
(2) $1+-1\to 0$,
(3) $(x+1)+-1\to x$.

Then R is R,E-Church-Rosser modulo associativity and commutativity. However, using our algorithm with $L = R$, the procedure generates an infinite set of left-linear rules (all supposed to be in L) among which are the following:

(4) $x+0 \to x$, by overlapping rule 1 and axiom of commutativity at the outermost occurrence.

(5) $-1+(x+1) \to x$, by overlapping rule 3 and axiom of commutativity at the outermost occurrence.

(6) $(1+x)+-1 \to x$, by overlapping rule 5 and axiom of commutativity at the outermost occurrence.

(7) $-1+(1+x) \to x$, by overlapping axiom of commutativity and rule 5 at occurrence 2 of rule 5.

(8) $(x+(y+1))+-1 \to x+y$, by overlapping the associativity axiom and rule 3 at occurrence 1 of rule 3.

This example shows that performing E-unification is costly but also powerful.

The theory of *equational term rewriting systems* presented here lacks many examples. We apologize for this drawback and explain the reason: interesting examples are simply intractable by hand. Only computer experiments can provide such examples. Some computer experiments, performed with the REVEUR 3 system can be found in [53]. REVEUR 3 is a version of the REVE Term Rewriting Laboratory developed by the EURECA group at CRIN and John Guttag's Group at MIT. REVE 1 has been implemented by Pierre Lescanne [61]. REVE 2 has been implemented by Randy Forgaard [23]. The main core of REVEUR 3 has been implemented by Claude and Hélène Kirchner [53].

6. Conclusion Let us now point out some interesting research directions for future work.

- First, the last open problem of infinite congruence classes should be addressed, since many interesting cases such as equipotency and identity fall in this category. Also the requirement that the strict E-subsumption preordering is well founded should be studied in detail: it may be possible that the property is true as soon as a minimal and complete unification algorithm exists for the theory E.
- Second, particular instances of our algorithm should be studied carefully as was done for associativity and commutativity [73]. We believe that various kinds of permutative axioms have similar properties. This would allow an improvement over previous algorithms [57], [39]. Performing various experiments should help a lot for studying such improvements.
- Third, we believe that our completion procedure can be improved: we conjecture that the transformation expressed in Lemma 32 could be incorporated into the algorithm (the one expressed in Lemma 31 is already achieved in the algorithm). This is not straightforward, since the set R of rules becomes R^E-Church-Rosser at the end, but does not have the property during the run of the algorithm. However, a rule can be considered to be *Church-Rosser* as soon as its critical pairs have been computed. As a consequence, dropping such rules into P when they become reducible in the sense of Lemma 32 could be sound, but this has to be proved.
- Last, we can imagine interesting rewriting relations other than those studied here: in OBJ [27], David Plaisted has implemented a very efficient reduction relation for the case of associative-commutative operators (and even for associative-commutative-idempotent operators having an identity): before applying an associative-commutative matching algorithm to find a redex, the term to be reduced is sorted in a lexicographic way. This technique allows a much more efficient associative-commutative

matching algorithm which makes the reductions much faster [75]. This technique should be studied carefully and possibly extended to other cases.

Acknowledgments. Many people at CRIN and at MIT have contributed to the implementation of REVE which is still going on. We are happy to thank them all here, especially Claude Kirchner. We are also grateful to Jose Meseguer for carefully reading the manuscript and to the referees for their pertinent comments and for exhaustively pointing out the typos.

REFERENCES

[1] A. AHO, R. SETHI AND J. D. ULLMAN, *Code Optimization and Finite Church-Rosser Theorems*, in Design and Optimization of Compilers, R. Rustin, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 89-105.

[2] A. M. BALLANTYNE AND D. S. LANKFORD, *New decision algorithms for finitely presented commutative semigroups*, Comput. Math. Appl., 7 (1981), pp. 159-165.

[3] G. M. BERGMAN, *The diamond lemma for ring theory*, Adv. in Math., 29 (1978), pp. 178-218.

[4] R. BOOK, *Confluent and other types of Thue systems*, J. Assoc. Comput. Mach., 29 (1982), pp. 171-182.

[5] B. BUCHBERGER AND R. LOOS, *Algebraic Simplifications*, in Computer Algebra, Symbolics and Algebraic Computations, Computing Supplementum 4, B. Buchberger, G. E. Collins and R. Loos, eds., Springer-Verlag, Berlin, West Germany, 1982, pp. 11-43.

[6] P. M. COHN, *Universal Algebra*, Harper and Row, New York, 1965.

[7] J. CORBIN AND M. BIDOIT, *A rehabilitation of Robinson's unification algorithm*, Proc. IFIP 83 Congress, North-Holland, R. E. A. Mason, ed. (1983), pp. 909-914.

[8] P. DERANSART, *Dérivation de Programmes* PROLOG *à partir de Spécifications Algébriques*, Rapport interne, INRIA, Le Chesnay, France, 1983.

[9] N. DERSHOWITZ, *A note on simplification orderings*, Inform. Process. Lett., 9 (1979), pp. 212-215.

[10] ———, *Ordering for term-rewriting systems*, Theoret. Comput. Sci., 17 (1982), pp. 279-301. Preliminary version in Proc. 20th Symp. Foundations Comput. Sci., San Juan, PR, 1979, pp. 123-131.

[11] ———, *Computing with term rewriting systems*, Tech. Rep., Univ. of Illinois, Urbana-Champaign, Illinois, 1983. Also in Information and Control, 63.

[12] N. DERSHOWITZ AND Z. MANNA, *Proving termination with multiset orderings*, Comm. ACM, 22 (1979), pp. 456-476. Also in Proc. 6th Internat. Colloquium on Automata, Languages and Programming, Graz, 1979, pp. 188-202.

[13] N. DERSHOWITZ AND L. MARCUS, *Existence and construction of rewrite systems*, Memo, The Aerospace Corporation, El Segundo, CA, 1982.

[14] N. DERSHOWITZ, J. HSIANG, N. A. JOSEPHSON AND D. A. PLAISTED, *Associate-commutative rewriting*, Proc. 8th Internat. Joint Conf. Artificial Intelligence, Karlsruhe, West Germany, 1983, pp. 940-944.

[15] N. DERSHOWITZ, *Termination of rewriting*, Tech. Rep. UIUCDCS-R-85-1220, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, Illinois, 1985.

[16] N. DERSHOWITZ AND A. TARLECKI, *A note on the uniqueness of term rewriting systems*, unpublished note.

[17] T. EVANS, *The word problem for abstract algebras*, J. London Math. Soc., 26 (1951), pp. 64-71.

[18] ———, *A decision problem for transformation of trees*, Canad. J. Math., 15 (1963), pp. 584-590.

[19] F. FAGES, *Associative-commutative unification*, in Proc. 7th Conf. Automated Deduction, Napa Valley, CA, Lecture Notes in Comput. Sci., Vol. 170, R. E. Shostak, ed., Springer-Verlag, Berlin, West Germany, 1984, pp. 194-208.

[20] ———, *Formes canoniques dans les algèbres Booléennes et application à la démonstration automatique en logique du premier ordre*, Thèse de 3eme cycle, Université de Paris 7, France, 1983.

[21] F. FAGES AND G. HUET, *Unification and matching in equational theories*, Proc. 5th Conf. Automata, Algebra and Programming, L'Aquila, 1983, Lecture Notes in Comput. Sci., Vol. 159, G. Ausiello and M. Protasi, eds., Springer-Verlag.

[22] M. FAY, *First order unification in equational theories*, Proc. 4th Conf. Automated Deduction, Austin, TX, Lecture Notes in Comput. Sci. Vol. 87, W. Bibel and R. Kowalski, eds., Springer-Verlag, Berlin, West Germany, 1979, pp. 161-167.

[23] R. FORGAARD AND J. V. GUTTAG, *A term rewriting system generator with failure-resistant Knuth-Bendix*, Tech. Rep., MIT Laboratory for Computer Science, Massachussets Institute of Technology, Cambridge, MA, 1984.

[24] L. FRIBOURG, *A superposition oriented theorem prover*, Proc. 8th Internat. Joint Conf. Artificial Intelligence, Karlsruhe, West Germany, 1983.

[25] J. A. GOGUEN, *How to prove algebraic inductive hypotheses without induction: with applications to the correctness of data type representations*, Proc. 5th Conf. Automated Deduction, Lecture Notes in Comput. Sci., Vol. 87, W. Bibel and R. Kowalski, eds., Springer-Verlag, Berlin, West Germany, 1980, pp. 356-373.

[26] J. A. GOGUEN AND J. MESEGUER, *Equality, Types, Modules and Generics for Logic Programming*, Logic Programming Symposium, Upsala, Sweden, 1984.

[27] J. A. GOGUEN, J. MESEGUER AND D. PLAISTED, *Programming with parameterized abstract objects in* OBJ, in Theory and Practice of Software Technology, D. Ferrari, M. Bolognani and J. Goguen, eds., North-Holland, New York, 1982, pp. 163-193.

[28] J. V. GUTTAG, E. HOROWITZ AND D. R. MUSSER, *Abstract data types and software validation*, Comm. ACM, 21 (1978), pp. 1048-1064.

[29] J. HSIANG, *Refutational theorem proving using rewriting systems*, Submitted to Artificial Intelligence.

[30] J. HSIANG AND N. DERSHOWITZ, *Rewrite methods for clausal and non clausal theorem proving*, Proc. 10th Internat. Colloquium on Automata, Languages and Programming, Barcelona, Spain, Lecture Notes in Comput. Sci., Vol. 154, Diaz, ed., Springer-Verlag, Berlin, West Germany, 1983, pp. 331-346.

[31] J. HSIANG AND D. PLAISTED, *Deductive program generation*, Tech. Rep., Computer Science Department, University of Illinois, Urbana-Champaign, Illinois, 1982.

[32] G. HUET, *Confluent reductions: Abstract properties and applications to term rewriting systems*, J. Assoc. Comput. Mach., 27 (1980), pp. 797-821. Preliminary version in Proc. 18th Symp. Foundations Comput. Sci., Providence, RI, 1977, pp. 30-45.

[33] ———, *A complete proof of correctness of the Knuth and Bendix completion algorithm*, J. Comput. System Sci., 23 (1981), pp. 11-21.

[34] G. HUET AND J. M. HULLOT, *Proofs by induction in equational theories with constructors*, J. Assoc. Comput. Mach., 25 (1982), pp. 239-266. Preliminary version in Proc. 21st Symp. Foundations Comp. Sci., IEEE, 1980.

[35] G. HUET AND D. OPPEN, *Equations and rewrite rules: A survey*, in Formal Language Theory: Perspectives and Open Problems, R. Book, ed., Academic Press, New York 1980, pp. 349-405.

[36] J. M. HULLOT, *Canonical forms and unification*, Proc. 5th Conf. Automated Deduction, Lecture Notes in Comput. Sci., Vol. 87, W. Bibel and R. Kowalski, eds., Springer-Verlag, Berlin, West Germany, 1980, pp. 318-334.

[37] ———, *Associative-commutative pattern matching*, Proc. 6th Internatl. Joint Conference on Artificial Intelligence, 1979.

[38] ———, *Compilation de formes canoniques dans les théories équationelles*, Thèse de 3eme cycle, Université de Paris Sud, Orsay, France, 1980.

[39] J. JEANROND, *Deciding unique termination of permutative rewrite systems: Choose your term algebra carefully*, Proc. 5th Conf. Automated Deduction, Lecture Notes in Comput. Sci., Vol. 87, W. Bibel and R. Kowalski, eds., Springer-Verlag, Berlin, West Germany, 1980, pp. 335-355.

[40] J. P. JOUANNAUD, *Church-Rosser computations with equational term rewriting systems*, Proc. Conf. Automata, Algebra and Programming, L'Aquila, 1983, Lecture Notes in Comput. Sci., Vol. 159, G. Ausiello and M. Protasi, eds., Springer-Verlag. Submitted to J. ACM.

[41] J. P. JOUANNAUD AND P. LESCANNE, *On multiset ordering*, Inform. Process. Lett., 15 (1982), pp. 57-62.

[42] J. P. JOUANNAUD AND M. MUNOZ *Termination of a set of rules modulo a set of equations*, Proc. 7th Conf. Automated Deduction, Napa Valley, CA, Lecture Notes in Comput. Sci., Vol. 170, R. E. Shostak, ed., Springer-Verlag, Berlin, West Germany, 1984, pp. 175-193.

[43] J. P. JOUANNAUD, H. KIRCHNER AND J. L. REMY, *Church-Rosser properties of weakly terminating equational term rewriting systems*, Proc. 8th Internat. Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, 1983.

[44] J. P. JOUANNAUD, C. KIRCHNER AND H. KIRCHNER, *Incremental construction of unification algorithms in equational theories*, Proc. 10th Internat. Colloquium on Automata, Languages and Programming, Barcelona, Spain, Lecture Notes in Comput. Sci., Vol. 154, J. Diaz, ed., Springer-Verlag, Berlin, West Germany, 1983, pp. 361-373.

[45] ———, *Algebraic manipulations as a unification and matching strategy for equations in signed binary trees*, Proc. 7th Internatl. Joint Conference on Artificial Intelligence, Vancouver, 1981, pp. 1016-1023.

[46] J. P. JOUANNAUD, P. LESCANNE AND F. REINIG, *Recursive decomposition ordering*, Proc. 2nd IFIP Workshop Conference on Formal Description of Programming Concepts, Garmish-Partenkirchen, D. Bjorner, ed., North-Holland, New York, 1983, pp. 33-348.

[47] S. KAMIN AND J. J. LEVY, *Two generalizations of the recursive path ordering*, Depart. Computer Science, University of Illinois, Urbana, IL, 1980, unpublished note.

[48] C. KIRCHNER, *A new equational unification method: A generalization of Martelli-Montanari algorithm*, Proc. 7th Conf. Automated Deduction, Napa Valley, Lecture Notes in Comput. Sci., Vol. 170, R. E. Shostak, ed., Springer-Verlag, Berlin, West-Germany, 1984, pp. 224-247.

[49] C. KIRCHNER AND H. KIRCHNER, *Résolution d'équations dans les algèbres libres et les variétés équationelles d'algèbres*, Thèse de troisième cycle, Université de Nancy I, France, 1982.

[50] H. KIRCHNER, *A general inductive completion algorithm and application to abstract data types*, Proc. 7th Conf. Automated Deduction, Napa Valley, Lecture Notes in Comput. Sci., Vol. 170, R. E. Shostak, ed., Springer-Verlag, Berlin, West Germany, 1984, pp. 282-302.

[51] C. KIRCHNER, *Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles*, Thèse d'état de l'Université de Nancy I, France, 1985.

[52] H. KIRCHNER, *Preuves par complétion dans les variétés d'algèbres*, Thèse d'état de l'Université de Nancy I, France, 1985.

[53] C. KIRCHNER AND H. KIRCHNER, *Implementation of a general completion procedure parameterized by built-in theories and strategies*, Proc. EUROCAL '85 Conf., Linz, Austria, 1985.

[54] D. KNUTH AND P. BENDIX, *Simple word problems in universal algebra*, in Computational Problems in Abstract Algebra, J. Leech, ed., Pergamon Press, Oxford, 1970, pp. 263-297.

[55] D. S. LANKFORD, *A unification algorithm for Abelian group theory*, Memo MTP-1, Mathematics Depart., Louisiana Tech. Univ., Ruston, LA, 1979.

[56] D. LANKFORD AND A. BALLANTYNE, *Decision procedures for simple equational theories with commutative axioms: Complete sets of commutative reductions*, Memo ATP-35, Dept. of Mathematics and Computer Science, Univ. of Texas, Austin, TX, 1977.

[57] ———, *Decision procedures for simple equational theories with permutative axioms: Complete sets of permutative reductions*, Memo ATP-37, Dept. of Mathematics and Computer Science, Univ. of Texas, Austin, TX, 1977.

[58] ———, *Decision procedures for simple equational theories with associative-commutative axioms: Complete sets of associative-commutative reductions*, Memo ATP-39, Dept. of Mathematics and Computer Science, Univ. of Texas, Austin, TX, 1977.

[59] D. S. LANKFORD AND G. BUTLER, Memo MTP-7, Mathematics Depart., Louisiana Tech. University, 1980.

[60] P. LECHENADEC, *Canonical forms in finitely presented algebras*, Proc. 7th Conf. Automated Deduction, Napa Valley, CA, Lecture Notes in Comput. Sci., Vol. 170, R. E. Shostak, Ed., Springer-Verlag, Berlin, West Germany, 1984, pp. 142-165.

[61] P. LESCANNE, *Computer experiments with the* REVE *term rewriting systems generator*, Proc. 10th Symp. Principles of Programming Languages, ACM, Austin, TX, 1983, pp. 99-108.

[62] ———, *Term rewriting systems and algebra*, Proc. 7th Conf. Automated Deduction, Napa Valley, CA, Lecture Notes in Comput. Sci., Vol. 170, R. E. Shostak, ed., Springer-Verlag, Berlin, West Germany, 1984, pp. 166-174.

[63] ———, *How to prove termination? An approach to the implementation of a new recursive decomposition ordering*, Proc. 6th Conf. Automata, Algebra and Programming, Bordeaux, France, 1984.

[64] M. LIVESEY AND J. SIEKMAN, *Unification of bags and sets*, Tech. Rep., Inst. fur Informatik I, Universitat Karlsruhe, West Germany, 1976.

[65] B. MÉTIVIER, *About the rewriting systems produced by the Knuth-Bendix completion algorithm*, Inform. Process. Lett., 16 (1983), pp. 31-34.

[66] P. MARCHAND, *Languages d'arbres, langages dans les algèbres libres*, Thèse d'Etat, Université de Nancy I, France, 1981.

[67] R. MILNER, *A theory of type polymorphism in programming*, J. Comput. System Sci., 17 (1978), pp. 348-375.

[68] D. MUSSER, *On proving inductive properties of abstract data types*, Proc. 7th Annual ACM Symp. Principles of Programming Languages, Las Vegas, CA, 1980, pp. 154-162.

[69] M. H. A. NEWMAN, *On theories with a combinatorial definition of 'equivalence'*, Ann. Math., 43 (1942), pp. 223-243.

[70] P. PADAWITZ, *Equational data type specification and recursive program scheme*, IFIP Working Conference on Formal Description of Programming Concepts II, D. Bjorner, ed., North-Holland, 1983.

[71] M. S. PATERSON AND M. N. WEGMAN, *Linear unification*, J. Comput. System Sci., 16 (1978), pp. 158-167.

[72] G. PETERSON, *A technique for establishing completeness results in theorem proving with equality*, this Journal, 12 (1983), pp. 82-100.

[73] G. PETERSON AND M. STICKEL, *Complete sets of reductions for some equational theories*, J. Assoc. Comput. Mach., 28 (1981), pp. 233-264.

[74] D. PLAISTED, *An associative path ordering*, NSF Workshop on the Rewrite Rule Laboratory, General Electric Research and Development, Schenectady, NY, 1983, pp. 123-136.

[75] ———, Personal communication, 1984.

[76] G. PLOTKIN, *Building-in equational theories*, Machine Intelligence, 7 (1972), pp. 73-90.

[77] J. C. RAOULT, *Finiteness results in term rewriting systems*, RAIRO Inform. Théor., 4 (1981), pp. 373-391.

[78] R. SETHI, *Testing for the Church-Rosser property*, J. Assoc. Comput. Mach., 21 (1974), pp. 671-679.

[79] R. SETHI, *Control flow aspects of semantics-directed compiling*, ACM Trans. Programming Languages And Systems, 5 (1985), pp. 554-595.

[80] J. SIEKMAN, *Universal unification*, Proc. 7th Conf. Automated Deduction, Napa Valley, CA, Lecture Notes in Comput. Sci., Vol. 170, R. E. Shostak, ed., Springer-Verlag, Berlin, West Germany, 1984, pp. 1-42.

[81] J. SIEKMAN AND P. SZABO, *A noetherian and confluent rewrite system for idempotent semigroups*, Semigroup Forum, 25 (1982), pp. 83-110.

[82] M. E. STICKEL, *A complete unification algorithm for associative-commutative functions*, J. Assoc. Comput. Mach., 28 (1981), pp. 423-434. Preliminary version in Proc. 4th Internat. Joint Conf. Artificial Intelligence, Tblissi, 1975.